UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Adaptive Signal Processing for Multichannel Sound using High Performance Computing

DOCTORAL THESIS

author

Jorge Lorente Giner

supervisors

Prof. Alberto González Salvador
Dr. Miguel Ferrer Contreras

Valencia, Spain
July 2015

To my parents and Mireia, for believing in me during this long and sometimes thankless way.

Abstract

The field of audio signal processing has undergone a major development in recent years. Both the consumer and professional marketplaces continue to show growth in audio applications such as immersive audio schemes that offer optimal listening experience, intelligent noise reduction in cars or improvements in audio teleconferencing or hearing aids. The development of these applications has a common interest in increasing or improving the number of discrete audio channels, the quality of the audio or the sophistication of the algorithms. This often gives rise to problems of high computational cost, even when using common signal processing algorithms, mainly due to the application of these algorithms to multiple signals with real-time requirements. The field of High Performance Computing (HPC) based on low cost hardware elements is the bridge needed between the computing problems and the real multimedia signals and systems that lead to user's applications. In this sense, the present thesis goes a step further in the development of these systems by using the computational power of General Purpose Graphics Processing Units (GPGPUs) to exploit the inherent parallelism of signal processing for multichannel audio applications.

The increase of the computational capacity of the processing devices has been historically linked to the number of transistors in a chip. However, nowadays the improvements in the computational capacity are mainly given by increasing the number of processing units and using parallel processing. The Graphics Processing Units (GPUs), which have now thousands of computing cores, are a representative example. The GPUs were traditionally used to graphic or image processing, but new releases in the GPU programming environments such as CUDA have allowed the use of GPUS for general processing applications. Hence, the use of GPUs is being extended to a wide variety of intensive-computation applications among which audio processing is included. However, the data transactions between the CPU and the GPU and viceversa have questioned the viability of the use of GPUs for audio applications in which real-time interaction between microphones and loudspeakers is required. This is the case of the adaptive filtering applications, where an efficient use of parallel computation in not straightforward. For these reasons, up to the beginning of this thesis, very few publications had dealt with the GPU implementation of real-time acous-

tic applications based on adaptive filtering. Therefore, this thesis aims to demonstrate that GPUs are totally valid tools to carry out audio applications based on adaptive filtering that require high computational resources. To this end, different adaptive applications in the field of audio processing are studied and performed using GPUs. This manuscript also analyzes and solves possible limitations in each GPU-based implementation both from the acoustic point of view as from the computational point of view.

One of the most used and simplest audio applications based on adaptive filtering is the adaptive channel identification. The adaptive channel identification system has been first designed and implemented on a GPU, and then, used in other adaptive applications to identify acoustic plants. Concretely, to this end, the Least Mean Square (LMS) algorithm has been implemented in the frequency domain. The size of both the input-data buffers and the adaptive filters, and also, how they can be managed in order to successfully exploit the GPU resources, is an important key in the design process. The goal is to propose a GPU implementation that can be easily adapted to any acoustic scenario, while freeing up CPU resources for other tasks. This application sets the bases for the real-time implementation of multichannel adaptive applications using a GPU as the main processor.

From the knowledge acquired in the channel identification application, two more audio applications based on adaptive filtering have been developed using the GPU as the main processor. First a multichannel Adaptive Equalization (AE) system has been developed. The prototype is based on the filtered-x LMS algorithm. Details of the parallelization of the algorithm have been analyzed. The experimental results have validated the real-time performance of the AE GPU implementation. Moreover, the application performance has also been analyzed from a computational point of view. The second application is a multichannel Active Noise Control (ANC) system. The ANC system has been implemented using two different algorithms: the LMS and the Normalized LMS with Orthogonal Correction Factors (NLMS-OCF). The experimental results have compared the performance of both multichannel ANC GPU implementations from different points of view: attenuation levels, convergence speed and computational aspects. Results of both applications have shown the usefulness of GPUs to develop versatile, scalable and low cost multichannel systems based on adaptive filtering.

Finally, this manuscript also introduces how to implement an ANC sys-

tem for networks with distributed processing. It also sets the bases for an implementation of distributed ANC systems using multi-GPU programming, where each GPU performs the processing of each node of the network.

Keywords: Multichannel Adaptive filtering, Adaptive Equalization, Active Noise Control, Distributed Processing, Graphics Processing Units.

Resumen

El campo de procesado de señales de audio ha experimentado un desarrollo importante en los últimos años. Tanto el mercado de consumo como el profesional siguen mostrando un crecimiento en aplicaciones de audio, tales como: los sistemas de audio inmersivo que ofrecen una experiencia de sonido óptima, los sistemas inteligentes de reducción de ruido en coches o las mejoras en sistemas de teleconferencia o en audífonos. El desarrollo de estas aplicaciones tiene un propósito común de aumentar o mejorar el número de canales de audio, la propia calidad del audio o la sofisticación de los algoritmos. Estas mejoras suelen dar lugar a sistemas de alto coste computacional, incluso usando algoritmos comunes de procesado de señal. Esto se debe principalmente a que los algoritmos se suelen aplicar a sistemas multicanales con requerimientos de procesamiento en tiempo real. El campo de la Computación de Alto Rendimiento basado en elementos *hardware* de bajo coste es el puente necesario entre los problemas de computación y los sistemas multimedia que dan lugar a aplicaciones de usuario. En este sentido, la presente tesis va un paso más allá en el desarrollo de estos sistemas mediante el uso de la potencia de cálculo de las Unidades de Procesamiento Gráfico (GPU) en aplicaciones de propósito general. Con ello, aprovechamos la inherente capacidad de paralelización que poseen las GPU para procesar señales de audio y obtener aplicaciones de audio multicanal.

El aumento de la capacidad computacional de los dispositivos de procesado ha estado vinculado históricamente al número de transistores que había en un chip. Sin embargo, hoy en día, las mejoras en la capacidad computacional se dan principalmente por el aumento del número de unidades de procesado y su uso para el procesado en paralelo. Las GPUs son un ejemplo muy representativo. Hoy en día, las GPUs poseen hasta miles de núcleos de computación. Tradicionalmente, las GPUs se han utilizado para el procesado de gráficos o imágenes. Sin embargo, la aparición de entornos sencillos de programación GPU, como por ejemplo CUDA, han permitido el uso de las GPU para aplicaciones de procesado general. De ese modo, el uso de las GPU se ha extendido a una amplia variedad de aplicaciones que requieren cálculo intensivo. Entre esta gama de aplicaciones, se incluye el procesado de señales de audio. No obstante, las transferencias

de datos entre la CPU y la GPU y viceversa pusieron en duda la viabilidad de las GPUs para aplicaciones de audio en las que se requiere una interacción en tiempo real entre micrófonos y altavoces. Este es el caso de las aplicaciones basadas en filtrado adaptativo, donde el uso eficiente de la computación en paralelo no es sencillo. Por estas razones, hasta el comienzo de esta tesis, había muy pocas publicaciones que utilizaran la GPU para implementaciones en tiempo real de aplicaciones acústicas basadas en filtrado adaptativo. A pesar de todo, esta tesis pretende demostrar que las GPU son herramientas totalmente válidas para llevar a cabo aplicaciones de audio basadas en filtrado adaptativo que requieran elevados recursos computacionales. Con este fin, la presente tesis ha estudiado y desarrollado varias aplicaciones adaptativas de procesado de audio utilizando una GPU como procesador. Además, también analiza y resuelve las posibles limitaciones de cada aplicación tanto desde el punto de vista acústico como desde el punto de vista computacional.

La identificación adaptativa de sistemas aplicada a identificar canales acústicos, es una de las aplicaciones de audio basadas en filtrado adaptativo más usadas a la vez que simples. En primer lugar, la aplicación de identificación adaptativa de sistemas se ha diseñado e implementado usando una GPU como procesador. Posteriormente, esta aplicación se ha utilizado en otras aplicaciones para identificar canales acústicos. Concretamente, para la implementación de esta aplicación, se ha usado el algoritmo de mínimos cuadrados o *Least Mean Square* (LMS) implementado en el dominio de la frecuencia. Tanto el tamaño de los filtros adaptativos como el de los vectores de datos de entrada además de cómo se deben gestionar dichos vectores es un aspecto muy importante del proceso de diseño para poder explotar con éxito todos los recursos computacionales que ofrece la GPU. El objetivo ha sido proponer una aplicación GPU que se pueda adaptar fácilmente a cualquier escenario acústico, mientras que al ejecutarse en la GPU, se liberen los recursos computacionales de la CPU para otras tareas. Con esta aplicación se han sentado las bases de la programación en GPU de aplicaciones de filtrado adaptativo multicanal en tiempo real.

A partir del conocimiento adquirido en la aplicación de identificación de canal, se han desarrollado dos aplicaciones multicanales de audio basadas en el filtrado adaptativo utilizando la GPU como procesador. En primer lugar, se ha desarrollado un sistema multicanal de Ecualización Adaptativa (AE). El prototipo está basado en el algoritmo LMS con filtrado-x. Se han analizado los detalles de la paralelización del algoritmo y posteriormente,

se han validado los resultados experimentales. Además, el rendimiento en tiempo real de la aplicación de AE implementada en la GPU ha sido analizado desde un punto de vista computacional. La segunda aplicación ha sido un sistema de Control Activo de Ruido (CAR) multicanal. El sistema CAR se ha implementado utilizando dos algoritmos diferentes: el LMS y el LMS Normalizadas con Factores de Corrección Ortogonal (NLMS-OCF). Se han presentado los resultados experimentales de ambas implementaciones del sistema CAR para poder comparar sus rendimientos desde diferentes puntos de vista: niveles de atenuación, velocidad de convergencia y aspectos computacionales. Los resultados de ambas aplicaciones, demuestran la utilidad de la GPU para la implementación de aplicaciones basadas en filtrado adaptativo así como el desarrollo de sistemas multicanales versátiles, escalables y de bajo coste.

Por último, este manuscrito también analiza los sistemas CAR para redes de nodos con procesado distribuido. Además, establece las bases para la implementación de dichos sistemas CAR distribuidos utilizando la programación en multiples GPUs donde cada GPU realiza el procesado de cada nodo de la red.

Palabras Clave: Filtrado Adaptativo Multicanal, Equalización Adaptativa, Control Activo de Ruido, Procesado Distribuido, Unidad de Procesamiento Gráfico.

Resum

El camp del processament de senyals d'àudio ha experimentat un desenvolupament important als últims anys. Tant el mercat de consum com el professional segueixen mostrant un creixement en aplicacions d'àudio, com ara: els sistemes d'àudio immersiu que ofereixen una experiència de so òptima, els sistemes intel·ligents de reducció de soroll en els cotxes o les millores en sistemes de teleconferència o en audiòfons. El desenvolupament d'aquestes aplicacions té un propòsit comú d'augmentar o millorar el nombre de canals d'àudio, la pròpia qualitat de l'àudio o la sofisticació dels algorismes que s'utilitzen. Això, sovint dóna lloc a sistemes d'alt cost computacional, fins i tot quan es fan servir algorismes comuns de processat de senyal. Això es deu principalment al fet que els algorismes se solen aplicar a sistemes multicanals amb requeriments de processat en temps real. El camp de la Computació d'Alt Rendiment basat en elements *hardware* de baix cost és el pont necessari entre els problemes de computació i els sistemes multimèdia que donen lloc a aplicacions d'usuari. En aquest sentit, aquesta tesi va un pas més enllà en el desenvolupament d'aquests sistemes mitjançant l'ús de la potència de càlcul de les Unitats de Processament Gràfic (GPU) en aplicacions de propòsit general. Amb això, s'aprofita la inherent capacitat de paral·lelització que posseeixen les GPUs per processar senyals d'àudio i obtenir aplicacions d'àudio multicanal.

L'augment de la capacitat computacional dels dispositius de processat ha estat històricament vinculada al nombre de transistors que hi havia en un xip. No obstant, avui en dia, les millores en la capacitat computacional es donen principalment per l'augment del nombre d'unitats de processat i el seu ús per al processament en paral·lel. Un exemple molt representatiu són les GPU, que avui en dia posseeixen milers de nuclis de computació. Tradicionalment, les GPUs s'han utilitzat per al processat de gràfics o imatges. No obstant, l'aparició d'entorns senzills de programació de la GPU com és CUDA, han permès l'ús de les GPUs per a aplicacions de processat general. D'aquesta manera, l'ús de les GPUs s'ha estès a una àmplia varietat d'aplicacions que requereixen càlcul intensiu. Entre aquesta gamma d'aplicacions, s'inclou el processat de senyals d'àudio. No obstant, les transferències de dades entre la CPU i la GPU i viceversa van posar en dubte la viabilitat de les GPUs per a aplicacions d'àudio en què es requereix la

interacció en temps real de micròfons i altaveus. Aquest és el cas de les aplicacions basades en filtrat adaptatiu, on l'ús eficient de la computació en paral·lel no és senzilla. Per aquestes raons, fins al començament d'aquesta tesi, hi havia molt poques publicacions que utilitzeren la GPU per implementar en temps real aplicacions acústiques basades en filtrat adaptatiu. Malgrat tot, aquesta tesi pretén demostrar que les GPU són eines totalment vàlides per dur a terme aplicacions d'àudio basades en filtrat adaptatiu que requereixen alts recursos computacionals. Amb aquesta finalitat, en la present tesi s'han estudiat i desenvolupat diverses aplicacions adaptatives de processament d'àudio utilitzant una GPU com a processador. A més, aquest manuscrit també analitza i resol les possibles limitacions de cada aplicació, tant des del punt de vista acústic, com des del punt de vista computacional.

La identificació adaptativa de sistemes aplicada a identificar canals acústics, és una de les aplicacions d'àudio basades en filtrat adaptatiu més utilitzades i simples. En primer lloc, l'aplicació d'identificació adaptativa de sistemes s'ha dissenyat i implementat utilitzant una GPU com a processador. Posteriorment aquesta aplicació s'ha utilitzat en altres aplicacions per a identificar canals acústics. Concretament, per a la implementació d'aquesta aplicació, s'ha utilitzat l'algorisme de mínims quadrats o emph Least Mean Square (LMS) implementat en el domini de la freqüència. Tant el tamany dels filtres adaptatius com el dels vectors de dades d'entrada, a més de com s'han de gestionar aquests vectors, és un aspecte molt important del procés de disseny ja que permet poder explotar amb èxit tots els recursos computacionals que ofereix la GPU. L'objectiu és proposar una aplicació GPU que es puga adaptar fàcilment a qualsevol escenari acústic, mentre que al executar-la amb la GPU, s'alliberen recursos de la CPU per a altres tasques. Amb aquesta aplicació s'han establert les bases de la programació en GPU d'aplicacions de filtrat adaptatiu multicanal en temps real.

A partir dels coneixements adquirits en l'aplicació d'identificació de canal, s'han desenvolupat dues aplicacions d'àudio basades en el filtrat adaptatiu utilitzant la GPU com a processador. En primer lloc, s'ha desenvolupat un sistema multicanal d'Equalització Adaptativa (AE). El prototip s'ha basat en l'algorisme LMS amb filtrat-x. S'han analitzat els detalls de la paral·lelització de l'algorisme, i posteriorment, s'han validat els resultats experimentals. A més, el rendiment en temps real de l'aplicació d'AE implementada amb la GPU ha estat analitzat des d'un punt de vista com-

putacional. Pel que fa a la segona aplicació, s'ha implementat un sistema de Control Actiu de Soroll (CAS) multicanal. El sistema CAS s'ha implementat per a dos algorismes distints: el LMS i el LMS Normalitzat amb Factors de Correcció Ortogonal (NLMS-OCF). A més a més, s'han presentat els resultats experimentals de les dues implementacions del sistema CAS per després poder comparar els rendiments des de diferents punts de vista: nivells d'atenuació, velocitat de convergència i alguns aspectes computacionals. Els resultats d'ambdues aplicacions han demostrat la utilitat de les GPU per a la implementació d'aplicacions basades en filtrat adaptatiu i el desenvolupament sistemes multicanal versàtils, escalables i de baix cost.

Finalment, aquest manuscrit també introdueix sistemes CAS per a xarxes de nodes amb processat distribuït. A més, estableix les bases per a la implementació d'aquests sistemes CAS distribuïts utilitzant la programació en múltiples GPUs on cada GPU realitza el processat de cada node de la xarxa.

Paraules Clau: Filtrat Adaptatiu Multicanal, Equalització Adaptativa, Control Actiu de Soroll, Processat Distribuït, Unitat de processat Gràfic.

Acknowledgements

There have been many people who have walked alongside me during the last five years. I would like to thank each and everyone of them. First and foremost, I offer my sincerest gratitude to my supervisors, Dr. Miguel Ferrer and Prof. Alberto Gonzlez, for supporting me throughout this thesis with their knowledge and advice, for the numerous hours of proofreading and for allowing me to work in my own way at the audio laboratory of the Telecommunications and Multimedia Applications Institute.

I am very grateful to Prof. José Ranilla Pastor from University of Oviedo, Dr. Mikael Sternad from the University of Uppsala and Dr. Jernimo Arenas García from the University Carlos III of Madrid for serving as reviewers of this thesis. Many thanks for having provided me very useful comments that helped to improve the quality of the Thesis. Special thanks also to Dr. Luis Vergara Domínguez from Technical University of Valencia and Dr. Jaume Segura García from University of Valencia for acting as members of the committee.

I would like to acknowledge and thank my research group (GTAC) for allowing me to conduct my research and providing any material requested. I owe my deepest gratitude to the university teachers Prof. Antonio Vidal, Dr. Maria de Diego, and Dr. Paco Martnez for their continuous support and collaboration, which have contributed in a very important way to the development of this thesis. Special thanks go to Dr. Gema Pi nero who supervised me the masters final project and advised me to start a research career. Thank you.

I would like to show my gratitude to all the people at the Universitat Politécnica de Valéncia that shared my daily work at the Telecommunications and Multimedia Applications Institue (iTEAM). Thanks to Prof. Jose J. Lpez, Dr. Maximo Cobos, Dr. Sandra Roger, Dr. Amparo Mart and Dr. Fernando Domene. In particular, I want to thank Dr. Jose Antonio Belloch for helping me to improve my GPU programming skills. Thanks also to my current and former colleagues at the iTEAM: Emanuel Aguilera,

Laura Fuster, Marian Simarro, Pablo Gutierrez and Cristian Anto nanzas.

Thanks to my dear friends Alvaro, Josep, Vicent, Estelles, Jornet, Alfredo, Juan Carlos, Josele, Jordi, Ana and Amparo for being always by my side and for making me smile even in my worse days.

Finally, and without hesitation, I would like to thank my family and Mireia, to whom this thesis is dedicated. A special feeling of gratitude to my parents: Clemente and Maria Jos, whose continuous support and words of encouragement has enabled me to complete this thesis. Last, but not least, I would like to express my sincere gratitude to Mireia, for their confidence, patience and fondness. Start a life with you has been the best thing that could have happened to me during my doctoral studies.

Jorge Lorente Giner
September 2015

Contents

List of Figures

List of symbols

x	Scalar
$\mathbf{x}$	Column vector in time domain
$\mathbf{X}$	Column vector in frequency domain
$(\cdot)^{\mathrm{T}}$	Transpose
$(\cdot)^{*}$	Complex conjugation
$(\cdot)^{\mathrm{H}}$	Conjugate transpose
$\lvert \cdot \rvert$	Absolute value
$\lVert \cdot \rVert$	ℓ_2 norm
B	Block size
L	Length of the adaptive filters
M	Length of the FIR filters that model the estimated secondary paths
F	L/B, number of partitions of the adaptive filters
P	M/B, number of partitions of the estimated secondary paths
I	Number of reference signals (reference sensors)
J	Number of secondary sources (actuators)
K	Number of error signals (error sensors)
R	Number of successive input vectors used in the weight update of the NLMS-OCF algorithm
D	Number of iterations between the successive input vectors used in the weight update of the NLMS-OCF algorithm
x_n	nth sample of signal x
$\mathbf{x}_n$	subscript n refers to the nth block iteration of signal x
$\mathbf{x}_{[ijk]n}$	subscript between brackets $[ijk]$ refers to the to the $(I{:}J{:}K)$ channel configuration
$\mathbf{x}^f$	superscript f refers to the fth partition of signal x
$\mathbf{x}^r$	superscript r refers to the rth input vectors used in the weight update of the NLMS-OCF algorithm

$\mathbf{0}_B$	vector of B zeros
$\mathrm{FFT}_B\{\mathbf{x}\}$	Fast Fourier Transform of size B of vector $\mathbf{x}$
$\mathrm{IFFT}_B\{\mathbf{x}\}$	Inverse Fast Fourier Transform of size B of vector $\mathbf{x}$
$\langle\cdot\rangle$	Dot (scalar) product
$\circ$	Hadamard (element-wise) product

Abbreviations and Acronyms

AE	Adaptive Equalization
ANC	Active Noise Control
APA	Affine Projection Algorithm
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
ASIO	Audio Stream Input/Output
BF	BeamForming
BFxLMS	Block Filtered-x Least Mean Square
BLAS	Basic Linear Algebra Subprograms
BLMS	Block Least Mean Square
Cg	C for graphics
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CUFFT	CUda Fast Fourier Transform
DA	Decorrelating Algorithm
DFT	Discrete-time Fourier Transform
DLMS	Delayed Least Mean Square
DSP	Digital Signal Processors
FAP	Fast Affine Projection
FBLMS	Frequency-domain Block Least Mean Square
FIR	Finfinite Impulse Response
FeLMS	Filtered-error Least Mean Square
FFT	Fast Fourier Transform
FLMS	Fast Least Mean Square
FBFxLMS	Frequency-domain Block-based Filtered-x Least Mean Square
FPBLMS	Frequency-domain Partitioned Block Least Mean Square
FPBFxLMS	Frequency-domain Partitioned Block Filtered-x Least Mean Square
FPBMFxLMS	Frequency-domain Partitioned Block Modified Filtered-x Least Mean Square

FPM-OCF	Frequency Partitioned Block Modified Filtered-X Normalized Least Mean Square algorithm with Orthogonal Correction Factors
FxLMS	Filtered-x Least Mean Square
FxNLMS	Filtered-x Normalized Least Mean Square
IFFT	Inverse Fast Fourier Transform
FPGA	Field-Programmable Gate Arrays
FURLMS	Filtered-U Recursive Least Mean Square
FxLMS	Filtered-x Least Mean Square
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
HPC	High Performance Computing
IIR	Infinite Impulse Response
LAPACK	(Linear Algebra PACKage
LMS	Least Mean Square
MFxLMS	Modified Filtered-x Least Mean Square
MIMO	Multiple Input Multiple Output
M-OCF	Modified filtered-x normalized least mean square with Orthogonal Correction Factors
NLMS	Normalized Least Mean Square
NLMS-OCF	Normalized Least Mean Square with Orthogonal Correction Factors
OCF	filtered-x normalized least mean square with Orthogonal Correction Factors
OpenGL	Open Graphics Library
OpenMP	Open Multi-Processing
PCI-E	Peripheral Component Interconnect Express
PRA	Partial Rank Algorithm
P2P	Peer-To-Peer
R-APA	Regularized Affine Projection Algorithm
RE	Room Equalization
RGBA	Red Green Blue Alpha
RLS	Recursive Least Squares
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SIMO	Single Input Multiple Output
SIMT	Single Instruction Multiple Thread
SM	Stream Multiprocessors

SP	Streaming Processors
UVA	Unified Virtual Addressing
WFS	Wave Field Synthesis

1 Introduction and Scope

Introduction and Scope 1

1.1 Background

This thesis fits into the field of Information Technology and Communications [1], particularly in the areas of Digital Signal Processing [2] and High Performance Computing (HPC) [3]. HPC generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop or personal computer in order to solve large problems in fields such as science, engineering, or business. HPC is generally related with the use of parallel processing to solve these large problems making use, for example, of the parallel resources of multicore Central Processing Units (CPU) or Graphics Processing Units (GPU).

Concretely, the main goal of this thesis is the development and implementation of adaptive signal processing algorithms [4] for multichannel spatial sound on GPU [5]. It is intended to integrate these algorithms into systems and prototypes to solve real computing-intensive problems of multichannel audio signal processing, with particular attention to current and future applications that involve: recording, transmission and process-

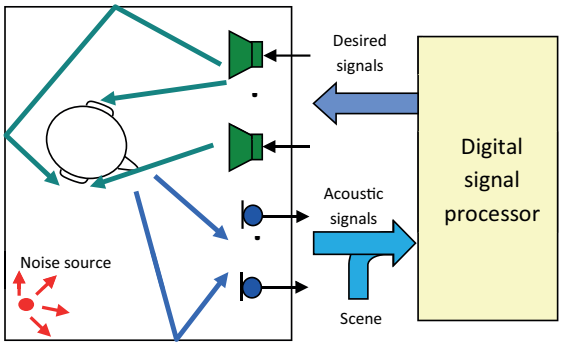


Figure 1.1. Basic Scheme of a multichannel recording-reproduction system.

ing of spatial sound. By spatial sound we mean sounds coming from one or several sources, which, implicitly or explicitly, have information of the recording environment, of the virtual stage and/or of the real reproduction stage (spatial information).

The main multichannel recording-reproduction problem can be described using the scheme depicted in the Figure 1.1, where a number of signals (to be reproduced) are processed through a multichannel processing system that takes into account certain features or parameters of the stage and of the sound scene to be reproduced. Generally speaking this is a discrete MIMO (Multiple Input, Multiple Output) system of sound signals with multiple loudspeakers and microphones. The number of listeners is variable, although in the figure only one is represented for simplicity. A large number of applications of spatial sound can be derived from the scheme of Figure 1.1, as applications that perform reproduction either through speakers or through headphones. The use of multiple microphones provides feedback information to the system, which is necessary to implement multiple applications based on adaptive algorithms like: the cancellation of undesired sounds [6], room equalization [7] or acoustic path identification [8].

There are two classical configurations or strategies to accomplish that the spatial sound is properly perceived by a listener through loudspeakers. The first one surrounds the listening area of a large number of sources and tries to recreate the sound field in a large zone of this space. It might be called global control of sound field. The second tries to recreate the sound

that would have been perceived in the actual listening in the ears of the listener. This option might be called local or localized control. Methods based on the first strategy are generally applicable to reproduction in large areas and large numbers of listeners, where investment in a large number of transducers is more justified. The second of the strategies is more appropriate for applications for domestic reproduction or for listeners with reduced mobility, since: the reproduction environments can have different characteristics, the number of listeners is small, the number of sound sources to be used can be reduced, and a greater precision is required for the recreation of the acoustic environment.

Due to the increasing demand for new acoustic sensations, which generally involves an increase of audio channels, the number of transducers (microphones or loudspeakers) used in the system showed in Figure 1.1 should be high. The increase of transducers involves an exponential increase in the number of operations that have to be performed by the computer. Moreover, many of these systems must be implemented in real time, which means that the sound signals that are picked up by the microphones must be processed continuously to generate the signals that have to be reproduced by the loudspeakers.

While the number of channels and the processing operations are increasing, the processing device has to perform all the operations in a very short time period to keep the multichannel system working in real time. Therefore, a powerful processing device is needed for these multichannel real-time systems. For this reason, this thesis is based on the use of the GPU as the main processor, so that multichannel signal processing algorithms can take benefit of GPUs SIMT (Single Instruction Multiple Thread) execution model. SIMT is a parallel execution model, used in some General Purpose GPU (GPGPU) platforms, where multithreading is simulated by SIMD (Single Instruction Multiple Thread) processors. The SIMD architecture allows running the same instruction over multiple data and therefore, obtaining massive parallelization for the multichannel acoustic processing.

In this context, the main work of this thesis has been the development of prototypes based on adaptive filtering algorithms for multichannel applications such as: Channel Identification (CI), Room Equalization (RE) and local Active Noise Control (ANC). All of them based on the new paradigms of computing and signal processing arising from the use of GPUs.

1.2 Motivation and objectives

The field of signal processing is highly suitable for the use of HPC based on low-cost hardware elements, because it is the needed bridge between the problems of computing and the real multimedia signals and systems that leads to user applications. Nowadays, signal processing has become a basic tool in many multichannel audio applications, such as noise control or equalization. Such applications often give rise to problems of high computational cost, even when using common signal processing algorithms. This is mainly due to the application of these algorithms to multiple signals and with real-time requirements. The present thesis aims to take a further step in the development of these systems using the computational power of GPGPUs to exploit the inherent parallelism of signal processing for multichannel audio applications. The development of new audio signal processing applications using GPGPUs as high performance computing elements, in which the heaviest part of the algorithms will be run, is a scientific and technological challenge that has been recently addressed. The results of researchers that combines signal processing with HPC should allow tackling potential applications that so far seemed unattainable for the consumer market, due to its computational cost.

Considering the motivation aspects, the main objective of this thesis is the following:

The development of prototypes based on adaptive filtering algorithms for massive multichannel sound applications working in real time.

To this end, the following global objectives are also considered:

- *Analyze the proposed adaptive algorithms and transform them seeking the most efficient implementation.*
- *Study the GPU architecture in order to use it as the main processor that computes and accelerates the massive processing tasks that require the cited applications.*
- *Solve drawbacks from both the acoustic and computational point of view.*
- *Implement GPU-based prototypes working under the real-time condition.*

Concretely, these global objectives can be particularized into several specific objectives, as follows:

- The assessment of the use of the GPU architecture for the implementation of different sub-systems of signal processing in spatial audio systems. This includes the testing of different parallel programming tools in order to analyze the benefits of using them for the parallelization of the signal processing algorithms. For example:

  Jacket. Tool for programming the GPU using Matlab environment [9].

  CUDA. It means Compute Unified Device Architecture. It is a programming language created by NVIDIA [10] to program the NVIDIA GPUs. Used for: create kernels that run in parallel using the multiple threads of the GPU, control the GPU memory allocations, and the data transactions between the CPU and GPU [11].

  CUFFT. CUDA implementation of the direct Fast Furier Transform (FFT) and inverse Fast Furier Transform (IFFT) [12].

  CULA. CUDA implementation of the LAPACK (Linear Algebra Package) mathematic library [13].

  CUBLAS. CUDA implementation of the BLAS (Basic Linear Algebra Subprograms) mathematic library [14].

- The development of signal processing sub-systems for multichannel sound on GPU. In particular, it is aimed to develop different algorithms based on block processing in the frequency domain for different adaptive applications such as: channel identification, room equalization and active noise control.

- The implementation of a single-channel identification application using a GPU as the main processor.

- The implementation of a room equalization application using a GPU as the main processor.

- The implementation of an ANC application using a GPU as the main processor.

- The study and implementation of an ANC system over a network of acoustic nodes with distributed processing.

1.3 Organization of the thesis

The remainder of this thesis describes the research that has been undertaken to develop the aims stated above. It is important to remark that this thesis involves two different disciplines: audio signal processing and computational science. Thus, chapter 2 is dedicated to review the state of the art of both the adaptive signal processing and the use of GPU in signal processing. Moreover, chapter 2 also introduces some basic knowledge of both disciplines, which could be considered as fundamental concepts within a mono-disciplinary thesis. However, this review helps to understand the thesis from a multi-disciplinary point of view. The chapters are organized and presented as follows:

- Chapter 2. This chapter is devoted to introduce basic knowledge of adaptive signal processing and GPU computing. Moreover a brief state of the art of both disciplines is also presented.

- Chapter 3. This chapter introduces the algorithms that have been used in the adaptive prototypes applications, which are: the Least Mean Squares (LMS) algorithm, and the Normalized LMS with Orthogonal Correction Factors (NLMS-OCF) algorithm.

- Chapter 4. This chapter presents the GPU implementation of a single-channel identification application based on the LMS algorithm.

- Chapter 5. This chapter deals with the GPU implementation of a multichannel room equalization application based on the LMS algorithm.

- Chapter 6. This chapter develops the GPU implementation of a multichannel active noise control application based on the LMS algorithms.

- Chapter 7. This chapter develops the GPU implementation of a multichannel active noise control application based on the NLMS-OCF algorithms.

- Chapter 8. This chapter proposes a distributed algorithm for an ANC system over a network of acoustic nodes.
- Chapter 9. Finally, the conclusions obtained throughout this thesis are presented, including some guidelines for future research lines. A list of published work related to this thesis is also given.

The GPU implementation of the different adaptive applications are explained from chapter 4 to chapter 7. Therefore, these chapters are organized following the same pattern:

1. The first section introduces the adaptive application.
2. The second section presents the proposed algorithm.
3. The third section describe the GPU implementation of the algorithm.
4. The results are given from a twofold perspective: the algorithm behavior and the computing results.

2 Preliminaries and basic concepts

Preliminaries and basic concepts 2

This chapter describes some necessary concepts for the understanding of this dissertation. It contains an introduction to the topic of adaptive filters, focusing on the three applications that are implemented in this thesis: channel identification, room equalization and active noise control. Moreover, due to the multi-disciplinary nature of this thesis, a brief sate of the art of the use of the GPU in signal processing as well as some basic concepts are also presented. Finally, the main issues related to the real-time implementation of the cited applications are stated.

2.1 Adaptive filtering algorithms

This thesis is concerned on the design and implementation of adaptive algorithms related to some applications of filtering, prediction and control [15]. When the system model is completely specified, standard design techniques can be employed to design the optimal filter or Wiener solution [16, 17]. However, this thesis is focused on designing techniques that are applicable when the system model is only partially known. These techniques have to self-adjust some kind of parameters related to the system. Then, it seems plausible that appropriate models could be estimated by analyzing actual

data. This is frequently done in practice, especially when the models are ill defined or time varying. This leads to adaptive filtering techniques [15, 4].

Adaptive filters [17, 18, 16] are self-designing systems which can adjust themselves to different environments. Since the invention, by Widrow and Hoff in 1959 [19], of one of the first adaptive filters, the so-called Least mean Square (LMS) algorithm [17, 16], many applications appeared to have the potential to use this fundamental concept. To cite some examples, they can be used in a wide variety of fields such as: control, communications, sonar, radar or biomedical engineering. However, all the applications have a common feature that is the target of filtering some input signal to match a desired response. The filter parameters are updated by making some measurements and applying them to the adaptive filtering algorithm such that the difference between the filter output and the desired response is minimized.

The adaptive filters usually use a recursive equation to adjust the filter coefficients iteratively. The reasons for solving the problem of adaptive filtering in an iterative manner are:

1. Iterative solutions do not require accumulation of signal samples. This results in a *memory saving*.

2. The accumulation of signal samples for post processing them to generate the filter output signal introduces a *delay* that could be unacceptable in some applications. The iterative procedure does not introduce such delay.

3. The iterative procedure results in an adaptive solution with some *tracking capabilities*. For example, if the signal statistics are time variant, then, the iterative adjustment of the filter coefficients will be able to adjust to the new statistics.

4. Iterative solutions are in general *simpler to implement*.

2.1.1 The adaptive filtering problem

Figure 2.1 shows the block diagram of a general adaptive filtering scheme. It also shows the signals involved in a general adaptive filtering problem. The signals are the following:

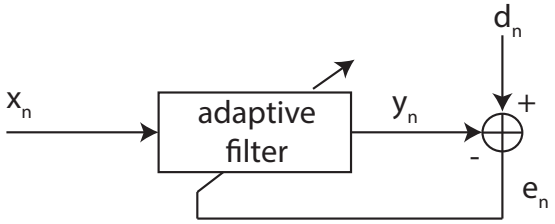


Figure 2.1. Block diagram of a general adaptive algorithm.

- x (Input signal). It is the input signal of the adaptive algorithm.
- y (Output signal). It is the output signal of the adaptive algorithm. It is used in different ways depending on the application.
- d (Desired signal). This signal is compared to the output signal in order to calculate de error signal.
- e (Error signal). It is the signal used by the adaptive algorithm for minimizing any of its parameters (usually its average power).

Particularizing for a system based on a sample-by-sample acquisition, a sample from a digital input signal x is fed into the adaptive filter that computes the corresponding output signal sample y. In general, the adaptive filter contains adjustable parameters whose values affect how y is computed. The output signal is compared to signal d, generally called "desired signal", by subtracting both samples. This difference signal is commonly known as the error signal, which is given at time instant n by $e_n = d_n - y_n$. The error signal is used in a procedure to alter or adapt the parameters of the adaptive filter from time n to time $n + 1$ in a defined manner. The goal of the adaptive procedure is to match the output signal of the adaptive filters with the desired response signal while the time instant n increases. In other words, the magnitude of signal e has to decrease over time. In this context, what is meant by "better" is specified by the adaptive algorithm used to adjust the parameters of the adaptive filter. The number and types of parameters within this system depend on the computational structure chosen for the system.

A wide variety of iterative algorithms have been developed in the literature for the operation of the adaptive filters, but, in practice, the choice

of one algorithm over another is based on one or more of the following parameters:

- **Rate of convergence**. This quantity describes the transient behavior of the algorithm. This is defined as the number of iterations required for the algorithm to converge close enough to the solution under stationary conditions.
- **Misadjusting**. This quantity describes the steady-state behavior of the algorithm. This is a quantitative measure of the amount by which the ensemble averaged final value of the mean-squared error deviates from the minimum mean-squared error produced by the desired filter.
- **Tracking capabilities**. When an adaptive algorithm performs in a time-variant environment, the algorithm is required to track statistical variations in the environment.
- **Computational requirements**. This parameter focuses on an implementation point of view. The parameters of interest are the number of operations (multiplications, divisions, and additions or subtractions) required for a complete iteration of the algorithm and the amount of memory needed to store the required data and also the program. These quantities influence the type and the price of the device needed to implement the adaptive filter, especially in multichannel scenarios.
- **Numerical Robustness**. The implementation of adaptive filtering algorithms on a digital computer, which inevitably operates using finite word-lengths, results in quantization errors. These errors sometimes can cause numerical instability of the adaptation algorithm. An adaptive filtering algorithm is said to be numerically robust when its digital implementation using finite-word-length operations is stable.

Ideally, it would be desirable to have a computationally-simple and numerically-robust adaptive algorithm with high performance in terms of convergence rate and misadjusting that can be easily implemented on a hardware device. However, in practice, these characteristics are incompatible in most cases and some kind of trade-off between the algorithm performance and the computational aspects is needed. Two examples of

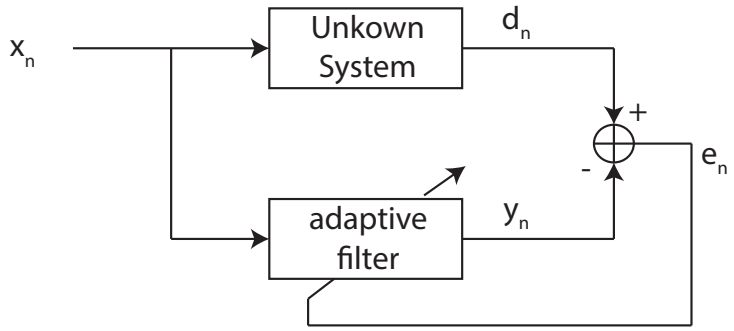


Figure 2.2. Block diagram of a system identification.

the mentioned trade-off, are the LMS [17, 16] and the RLS (Recursive Least Squares) [17, 16] algorithms. The first one is a computationally simple and numerically robust algorithm, but it exhibits a slow convergence as a drawback, especially for colored inputs. On the other hand, the RLS algorithm has a fast convergence while it is computationally complex and it is also known for having numerical problems [17]. These are two well-known algorithms of the vast repertoire of adaptive algorithms. As this thesis is focused on developing applications based on adaptive filtering, the goal is to achieve a better trade-off by implementing algorithms with good results in terms of convergence rate using parallel computation in order to reduce the computational drawbacks.

Concretely, in this thesis we are going to use the LMS algorithm and its variants to implement three adaptive applications: system identification (channel identification), inverse modeling (room equalization), and active noise control.

2.1.2 System identification

Figure 2.2 shows the general problem of system identification [20, 8]. In this block diagram, the unknown system represents a general input-output relationship that the adaptive filter has to identify.

In this model, the desired response signal, d, represents the output of the unknown system, while signal x represent its input. Here, the task of the adaptive filter is to accurately represent the signal d at its output. If $y = d$, then the adaptive filter has accurately modeled or identified the

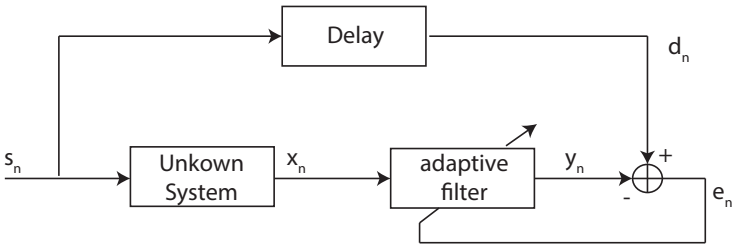


Figure 2.3. Block diagram of a channel post-equalization system.

unknown system that is fed by x. In practice, the adaptive filter can only be adjusted so that y closely approximates d over time.

The system identification task is at the heart of numerous adaptive filtering applications such as: channel identification in communication systems or plant identification in adaptive control of sound.

2.1.3 Channel equalization

Channel equalization was one of the first applications of adaptive filters and is described in the pioneering work of Lucky [7]. Today, it remains as one of the most popular uses of the adaptive filters. In [21], Qureshi provides a tutorial on Adaptive Equalization (AE). Generally speaking, channel equalization is a technique for transmitting signals across non-ideal communication channels. The transmitter sends a sequence at time instant n, s_n, that is known to both the transmitter and the receiver. The received signal is used as the input signal of the adaptive filter, x_n, which adjusts its characteristics so that its output signal y_n closely matches a delayed version of the known transmitted signal $s_{n-\Delta}$. A block diagram of the channel post-equalization system is shown in Figure 2.3.

2.1.4 Active noise control

The traditional passive noise controllers made by sound absorbing materials achieve high attenuation over a broad frequency range. However, they are inappropriate for many modern applications because they are relatively voluminous, costly, and ineffective for low frequencies. These problems are overcome by using ANC systems. The theory of ANC systems was proposed by Paul Leug in 1930's [22]. Leug suggested an electro-acoustic device to

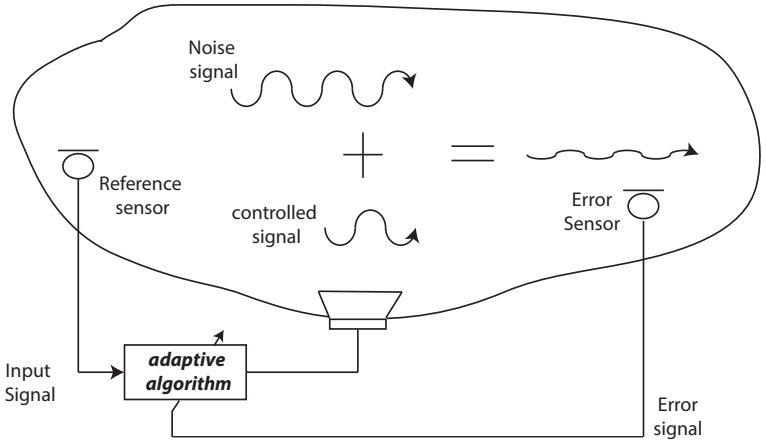


Figure 2.4. Components of a feedforward active noise control system.

reproduce sound of equal magnitude but opposite phase for cancellation of tonal sound in an acoustic duct. However, due to technological limitations, his idea was not feasible. In 1950's, Olsen applied analog electronic technology to invent the first realization of ANC. Olsen called it "Electronic Sound Absorber" [23]. All the early realizations of ANC systems had an analog amplifier which coupled a loudspeaker to a microphone by a simple negative feedback. The common drawback of these systems was that they were non-adaptive. The realization of the ANC systems was revolutionized by the use of the digital technology. Actually, today ANC controllers are implemented using adaptive digital filters instead of analog amplifiers. Widrow et al. developed the idea of adaptive ANC in 1975 [6], where successful applications of adaptive ANC systems in electrocardiography, telecommunication and speech enhancement were showed. After that, several researchers began to apply adaptive ANC for acoustic noise control.

Active noise control systems are based on the principle of destructive interference between a disturbance sound field called primary noise and a secondary sound field generated by a controlled secondary source called actuator. The target is to cancel, or at least minimize, the primary noise signal. To cancel the primary noise, the ANC system uses an adaptive algorithm to generate the secondary sound field from a reference signal that is correlated with the primary noise. There are two different approaches to adaptive ANC related with the way of obtaining a reference of the noise sig-

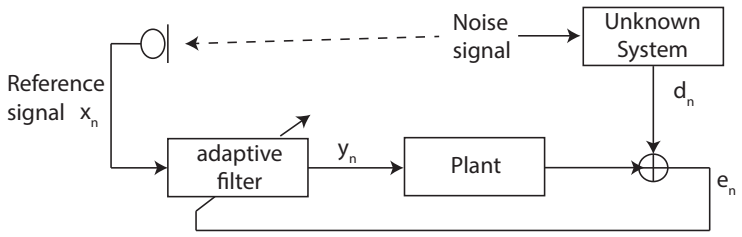


Figure 2.5. Block diagram of a feedforward active noise control system.

nal. The two approaches are: adaptive feedforward and adaptive feedback systems. In the adaptive feedforward ANC systems, the reference signal is obtained regardless of the error signal, for example placing a reference microphone outside the desired quiet zone to identify the noise field. On the other hand, the adaptive feedback ANC systems were suggested by Eriksson in 1991 [24]. In this approach, the noise field is identified using a feedback estimator without a reference microphone. The study of this thesis is focused on the first approach of feedforward control of sound. For the purpose of noise cancellation, the noise or undesired signal is monitored at a specific spatial point by a sensor that is called error sensor. Therefore, cancellation is only carried out at that specific spatial point and also at close points around the error sensor (see Figure 2.4). Nelson and Elliot in [25], and Kuo and Morgan in [26] provides two excellent tutorial reviews of ANC.

Figure 2.5 shows the block diagram of a feedforward ANC system, which combines elements of both the inverse modeling and system identification. The reference signal x is obtained by a sensor and therefore, is correlated with the noise signal. The output signal of the adaptive filter y passes through a plant before it is added to signal d to form the error signal e. The plant interferes with the operation of the adaptive filter by changing the amplitude and phase characteristics of signal y, which is represented in signal e. Thus, a prior knowledge of the plant is generally required in order to adapt the parameters of the filter properly.

2.1.5 The LMS algorithm

Although there are lots of adaptive algorithms in literature with a vast range of applications, the Least Mean Square (LMS) algorithm suggested

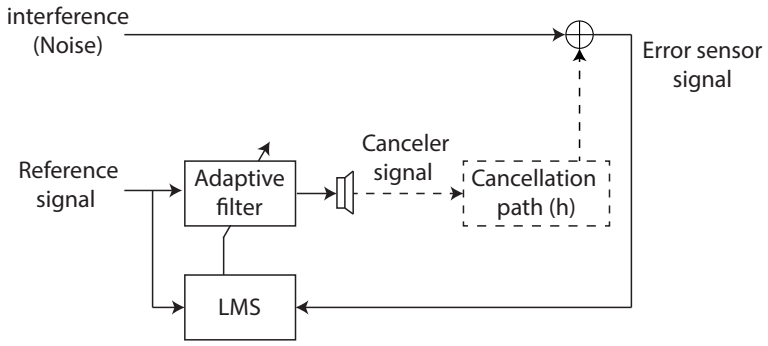


Figure 2.6. Block diagram of a noise canceler using LMS algorithm.

by Widrow and Hoff in 1960 [19], is one of the most widely used. The LMS algorithm has been widely adopted in commercial products mainly because its stability and computational simplicity. However, one of its primary drawback is that suffers from slow convergence.

The LMS algorithm, whose equations are introduced in chapter 3, has been cited and worked upon many researches over the years. As a result, many modifications have been proposed and presented in literature. Here, we are going to cite the most representative. To begin with, the Normalized LMS (NLMS) algorithm was proposed by Nagumo and Noda [27] and Albert and Gardner in 1967 [28]. This modification improves the convergence speed and the stability with time-variations of the power of the input signal. The LMS algorithm was also presented in frequency domain with block processing in [29]. It was first called the Fast LMS (FLMS) algorithm. In [30, 31] and the references therein the reader will find a complete analysis of the Frequency-domain Block LMS (FBLMS) algorithm. Although the LMS algorithm and its variations was being used in many applications, there was a problem with applications of adaptive cancelation of interferences. The authors found that there was an additional filter in the cancelation path from the adaptive filter to the summing junction, as depicted by h in Figure 2.6. Analysis of this problem showed that the LMS algorithm depends on the phase response of h and makes it quite unstable. Authors proposed two main solutions: one was to employ an inverse filter in the cancellation path and the other was to introduce a filter in the reference path in order

to compensate the effects of h. The first technique is known as *filtered-error* [32, 33], while the second technique is known as *filtered-reference* or *filtered-x* [4], because the reference signal is commonly denoted by x. The Filtered-error LMS (FeLMS) was later modified by Dubjert [34] and Elliot [35] in order to increase the convergence rate. However, the Filtered-x LMS (FxLMS) solution is the most widely used in literature.

During the last decades, the FxLMS algorithm has been developed for special cases resulting in a wide range of algorithms. When h is a simple delay, the FxLMS takes on a special form known as the Delayed LMS (DLMS) algorithm. Analysis of this algorithm can be found in [36, 37]. In [38], Eriksson introduced a generalization of the FxLMS to the case of Infinite Impulse Response (IIR) filters, and he called it the Filtered-U Recursive LMS (FURLMS) algorithm. The FURLMS algorithm is also analyzed in [39]. Moreover, a technique called delayless subband adaptive filter [40] was developed to address the problem of having to model the cancelation-path filter with a large number of adaptive coefficients. This technique computes the adaptive weights in each subband and transforms the composite set of weights to an equivalent fullband Finite Impulse Response (FIR) filter. These are some of the most representative techniques, but the number of variations of the FxLMS algorithm is too numerous to list here.

As commented before, the FxLMS algorithm suffers from slow convergence due to the delay in the cancellation path. Therefore, some efforts have been done to overcome this problem. First, Bjarnason developed the Modified FxLMS (MFxLMS) algorithm [41], which significantly improves the convergence rate of the FxLMS. However, although the MFxLMS algorithm improves the convergence rate, another family of algorithms was developed to raise the convergence speed of LMS-type algorithms, including the MFxLMS. This family of algorithms is referred to in literature as APA family [42]. The APA family takes the name from the most widely used algorithm of the family: the Affine Projection Algorithm (APA). The APA was proposed by Ozeki and Umeda in 1984 [43]. Since then, lots of contributions have analyzed the APA, see for example [44], [45] and [46]. The basic idea of this algorithm is to update the weights on the basis of multiple input signal vectors, in contrast to the FxLMS which uses a single input signal vector to update the coefficient weights. Therefore, its high computational cost is the major drawback of the algorithm. To re-

duce the computational complexity, a simplified version, called Fast Affine Projection (FAP) algorithm, was proposed by Gay and Tavathia in 1995 [47]. From [43], some variants of the APA algorithm have been proposed by different authors, such as the Regularized APA (R-APA) [48], the Partial Rank Algorithm (PRA) [49], the Decorrelating Algorithm (DA) [44] or the Normalized Least Mean Square with Orthogonal Correlation Factors (NLMS-OFC)[50][45].

The NLMS-OCF algorithm has been used in this thesis because we find its practical implementation easier than the APA implementation, and, as considered in [45], the NLMS-OCF algorithm is a generalization of the APA that allows other than unit delay between the input vectors. Therefore, in chapter 7 the NLMS-OCF algorithm is applied to a multichannel ANC system.

2.2 Graphics processing unit applied to digital signal processing

During the last decades, digital signal processing problems has been implemented using specific hardware devices such as Digital Signal Processors (DSP) and Field-Programmable Gate Arrays (FPGA). As the adaptive filtering problems are inside the field of digital signal processing, these two devices where the most widely used for prototypes based on adaptive filtering. DSPs and FPGAs are presented in the following lines.

Digital signal processor

A DSP is a specialized microprocessor with an optimize architecture for the operational needs of digital signal processing. In fact, hardware engineers use DSP to mean Digital Signal Processor, just as algorithm developers use DSP to mean Digital Signal Processing. Therefore, the DSP are designed considering the most common operations in digital signal processing: additions, multiplications and delays (store in memory). This kind of device has seen a tremendous growth in the last decades, finding use in everything from mobile phones to advanced scientific instruments.

Field-programmable gate arrays

An FPGA is an integrated circuit designed to be configured by a customer or a designer after manufacturing. As opposed to Application Specific Integrated Circuits (ASICs), where the device is built for the particular design, the FPGAs can be programmed to the desired application or functionality requirements. An FPGA contains programmable logic components called logic blocks, and a hierarchy of reconfigurable interconnections that allows to wire the blocks according to the application to be developed. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

However, the science filed of high performance computing is expanding its scope to many scientific and engineering problems, such as signal processing algorithms, to develop user applications in the promising market of processing, transmission and reproduction of multimedia content. The incorporation into the market of processors with multiple cores (multicore architectures) and the increasing use of GPUs in general purpose applications is at the same time a challenge and a great opportunity.

In the recent years, the widespread use of GPUs for general data processing instead of its traditional use for image processing has enabled the emergence of GPUs as a low-cost and high-performance solution for general-purpose computations, where the field of digital signal processing is not an exception. The computing power of the new GPU architectures should allow complex problems that require intensive computation to be solved even in personal computers.

2.2.1 State of the art

In the beginning, the GPUs were created only for graphics processing. Since then, GPUs have evolved (both in its hardware implementation, as in the programming interface) to a very powerful processor capable of developing general purpose processing tasks. Apart from image processing, which traditionally has been the main application of the GPUs, in literature, one can find many references to the use of GPUs for general purpose processing [51]. The first GPU implementations of non-graphics applications were programmed using Cg (C for graphics) shading language [52], which is a

graphical programming language. Using this language, Whalen [53] implemented seven common functions of audio processing on GPU: chorus, compress, delay, high-pass filter, low-pass filter, noise-gate and normalization. From then on, a wide variety of audio applications have been already implemented on GPU. Matsuyama et al describe in [54] a method for automatically generating sounds in real time. In [55], Gallo and Tsingos implemented a multichannel sound propagation model and a spatial filtering on the GPU. The signals that contained the different sounds were stored as RGBA (Red Green Blue Alpha) textures where each component had a bandpass copy of the original signal. There were also GPU implementations on acoustic modeling of rooms, for example in [56], where Robert et al made use of the 3D textures of the GPU and OpenGL (Open Graphics Library) [57] objects to get the maximum parallelism in the computing operations. However, the general purpose programming of GPUs using OpenGL was very complicated since the operations had to be programmed like graphics operations.

In June of 2007, NVIDIA, one of the leading manufacturers of graphics cards, launched CUDA. CUDA refers both to a compiler and to a set of development tools created by NVIDIA. It enables the programmers to use a variation of the C programming language to implement algorithms in the NVIDIA graphic cards. The emergence of CUDA, which made the GPU programming easier, together with the emergence of new powerful graphics cards, made that many algorithms of different disciplines were implemented and tested on GPU. Furthermore, many of these new cards were no longer manufactured for graphics as most of them no longer had external output. From that moment, the use of GPUs for parallelizing computationally expensive signal processing algorithms became a topic of interest. It was evidenced in the issues of IEEE Signal Processing Magazine of November 2009 [58] and March 2010 [59], which had almost exclusive dedication to signal processing on GPUs.

At that time, the acoustics researchers began to consider the GPU as a powerful platform to implement audio and acoustic applications [60]. However, most of the results were simulations. Trebien and Oliveria proposed in [61] a method for synthesizing sound at the same time that they were performing multichannel audio processing on the GPU. The GPU implementation showed speed-ups of up to four orders of magnitude over the CPU. Cowan and Kapralos were pioneers in implementing auraliza-

tion algorithms on the GPU [62, 63]. The auralization of sound consists in recording the sound in an environment and then reproducing it with the same environment characteristics in which it was recorded. To carry out the auralization, high computational capacity is needed because it requires massive filtering. In [64], the authors implemented on GPU some methods based on wave-based finite-difference. In [65] and [66], Webb and Bilbo, use the GPU in the field of room acoustics. Regarding digital treatment of sound, two techniques like Wave Field Synthesis (WFS) [67] and Beamforming (BF) [68] have also been implemented on GPU [69]. In [70], one can find a summary of the GPU implementation in different sound applications.

Furthermore, the new GPUs have increased their processing capacity and their use is being considered in the field of multichannel algorithms based on adaptive filtering. However, efficient use of parallel computation in the adaptive filtering context is not straightforward due to the feedback loops. Moreover, the data transactions among GPU, CPU and the audio card are critical for the real-time performance. For these reasons, very few publications deal with the GPU implementation of real-time acoustic applications based on adaptive filtering. As an example, in [71], Schneider et al implemented a frequency adaptive algorithm for echo cancellation on a GPU. Due to the lack of studies in this area, this thesis is mainly focused on GPU implementations of multichannel adaptive algorithms.

2.2.2 Graphics processing unit

Traditionally, a GPU is defined as a programmable logic chip that renders images, animations and video for the computer's screen. However, since GPUs perform parallel operations on multiple sets of data, they are increasingly used as processors for non-graphics applications that require repetitive or intensive computations. From a conceptual point of view and following Flynn's taxonomy [72], a GPU can be considered as a SIMD machine. A SIMD example could be a computer in which a single set of instructions is executed on different data sets. SIMD implementations usually work synchronously, with a common clock signal, so that an instruction unit sends the same instruction to all of the processing elements, which execute this instruction on their own data simultaneously.

A block diagram of the organization of the GPU architecture is shown in Figure 2.7. A GPU is composed by multiple Stream Multiprocessors (SM). Each SM consists of multiple pipelined cores called Streaming Pro-

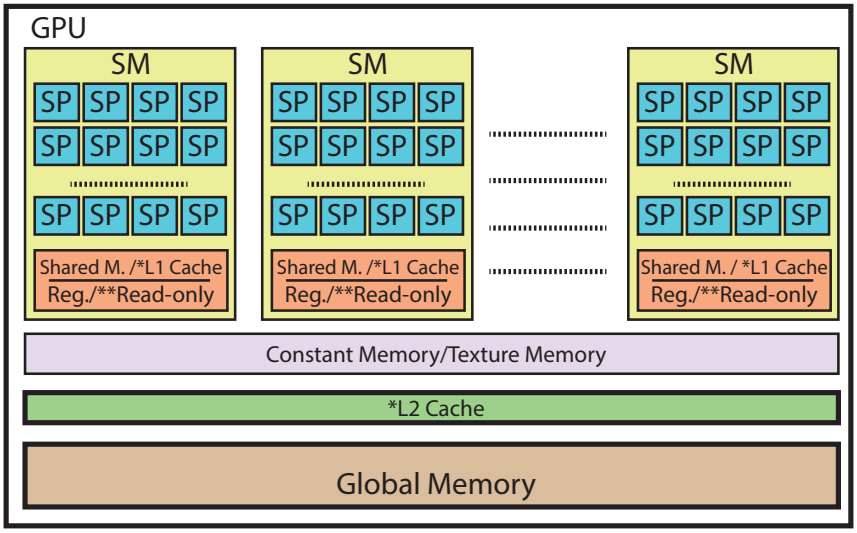


Figure 2.7. A GPU has multiple Stream Multiprocessor (SM) that are composed of multiple pipelined cores (SP). The number of SPs depends on the compute capability and the number of SMs depends on the kind of the device. A GPU device has off-chip device memories and on-chip memories. *(in devices with compute capability 2.x and 3.x) **(only in devices with compute capability 3.x).

cessors (SP). The number of SP cores in each SM depends on the CUDA capability. The oldest GPUs with compute capability 1.2 or 1.3 (Tesla architecture) own 8 SP cores per multiprocessor. GPUs with compute capability 2.0 (architecture) has 32 pipelined cores, while newest GPUs which its compute capability is 3.0 or 3.5 (Kepler architecture) own even 192 pipelined cores. Moreover, a GPU device has a large amount of off-chip device memory (*global-memory*) and a fast on-chip memory (*shared-memory*, *registers*). The *shared-memory* is normally used when multiple threads must share data. There are also read-only cached memories called *constant-memory* and *texture-memory*. The first memory is optimized for broadcast (for example when all the threads read the same memory location), while the second one is more oriented to graphics. Furthermore, the GPU devices of compute capability 2.x and greater come with an L1/L2 cache hierarchy that is used to cache *global-memory*. Cache of level L1 is located on-chip memory. The same occurs to the *read-only cache* that is

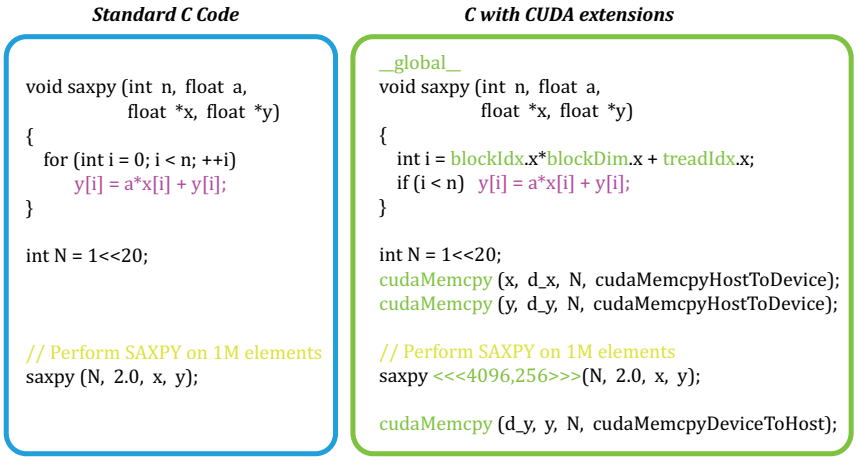


Figure 2.8. A simple example of code written first in plain C and then in C with CUDA extensions.

only present in GPU devices of compute capability 3.x.

2.2.3 Compute unified device architecture

CUDA is a parallel computing platform and programming model created by NVIDIA. Using CUDA, the GPUs can be used for general purpose processing (not exclusively graphics). This approach is known as GPGPU. The CUDA platform is accessible to software developers through CUDA-accelerated libraries and extensions to standard programming languages, including C, C++ and Fortran. These extensions let the developer express massive amounts of parallelism and direct the compiler to the portion of the application that maps to the GPU. A simple example of code is shown in Figure 2.8, written first in plain C and then in C with CUDA extensions.

This software makes profit from the high amount of execution threads which are available on a GPU. In CUDA, the code that has to be executed on the GPU is written defining a kernel function. Each kernel has a grid configuration, which defines the number of threads and how they are distributed and grouped. Threads are grouped into blocks, and the blocks configure the grid. Both the blocks of threads and the grid of blocks are organized in three dimensions. Thus, a thread identification will be defined by a position within a block (`ThreadIdx.x`, `ThreadIdx.y`, and `ThreadIdx.z`),

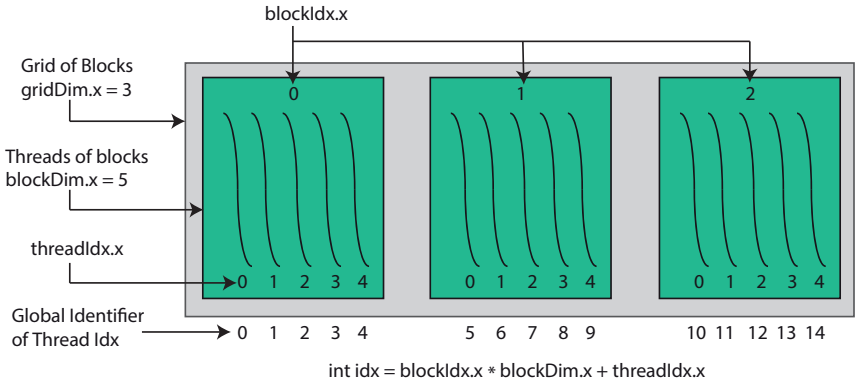


Figure 2.9. Distribution of the threads inside the cuda grid.

and this block will be defined within a grid (`BlockIdx.x`, `BlockIdx.y`, and `BlockIdx.z`). Parameters `BlockDim.x`, `BlockDim.y`, and `BlockDim.z` indicate the dimensions of a block in the same way as `gridDim.x`, `gridDim.y`, and `gridDim.z` indicate the dimensions of the grid. Figure 2.9 shows a CUDA kernel configured by a one-dimensional grid composed of 3 blocks, and each block composed of 5 one-dimensional threads.

The dimensions of the grid and the blocks are limited by the compute capability of the GPU. Figure 2.10 shows a table that is taken from the NVIDIA CUDA programming guide [11] that outlines some CUDA features that depend on the capability of the GPU device. Moreover, all the GPU devices that allow CUDA technology with its corresponding capability are published in [73]. Together with CUDA technology, the GPU developers can download the NVIDIA GPU Computing SDK (Software Development Kit) from the NVIDIA web site. The SDK collect different CUDA projects that could be taken as examples for the GPU developers.

2.2.4 Multi-GPU programming with multicore

Open Multi-Processing (OpenMP) [74] is an Application Program Interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran. OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer. It works by using parallel regions that are

	Compute Capability						
Technical Specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.5**
Maximum dimensionality of grid of thread blocks	2				3		
Maximum x-dimension of a grid of thread blocks	65535					2^{31}-1	
Maximum y- or z-dimension of a grid of thread blocks	65535						
Maximum dimensionality of thread block	3						
Maximum x- or y-dimension of a block	512				1024		
Maximum z-dimension of a block	64						
Maximum number of threads per block	512				1024		
Warp size	32						
Maximum number of resident blocks per multiprocessor	8					16	
Maximum number of resident warps per multiprocessor	24		32		48	64	
Maximum number of resident threads per multiprocessor	768		1024		1536	2048	
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K	64 K	

Figure 2.10. CUDA features that depend on the capability of the GPU device.

specified by the programmer. The CPU code runs sequentially and at some point hits a section where work can be distributed into several processors that perform the computations (CPU core spans several CPU threads). Afterwards, when all the computations are completed, all the CPU threads converge to a single thread again, which is called the *master* thread.

CUDA and openMP can work together in the following way. If a machine has a multicore processor and several GPUs, the parallelization can be achieved by defining a number of threads in the parallel region equal to the number of GPUs. In this sense, each CPU thread deals with a GPU. This is very important since a CPU thread is bound with a GPU context. Thus, all subsequent CUDA calls are executed in its corresponding GPU context [75].

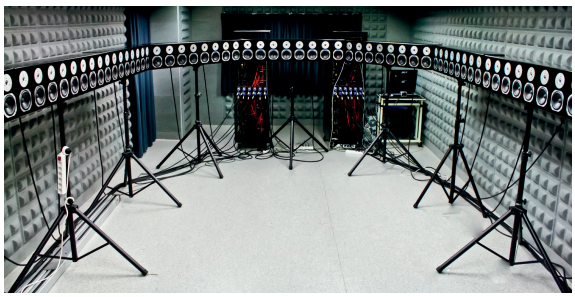


Figure 2.11. A photo of the listening room

CUDA Architecture	Fermi
CUDA Capability	2.0
Number of SMs	16
Number of cores per SMs	32
Maximum number of threads per block	1024

Table 2.1. Characteristics of the GPU (GTX-580M).

2.3 Implementation aspects

This section is devoted to describe the main issues related to the real-time implementation of the adaptive applications that has been developed in this thesis. It begins with a description of the listening room used to test the prototypes developed during this thesis. Then, a hardware description of the prototypes is given. Finally, some constraining conditions such as the real-time condition and the causality condition are analyzed.

2.3.1 Description of the listening room

The prototypes that have been developed during this thesis have been tested inside a listening room. The listening room belongs to the audio and communication signal processing group [76]. The dimensions of the listening room are: 9 meters long and 4.5 meters wide. Figure 2.11 shows an image of the room, while Figure 2.12 depicts an example of an estimated response in frequency domain.

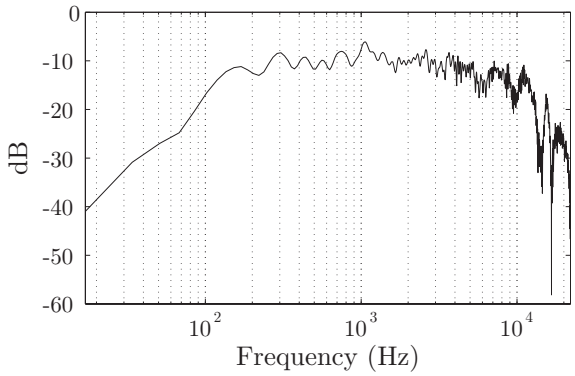


Figure 2.12. Example of a frequency response of the listening room

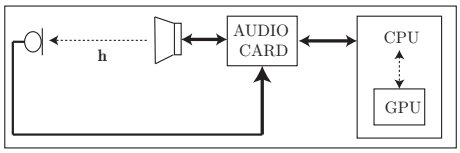


Figure 2.13. Prototype description of the GPU implementation of the single-channel identification system.

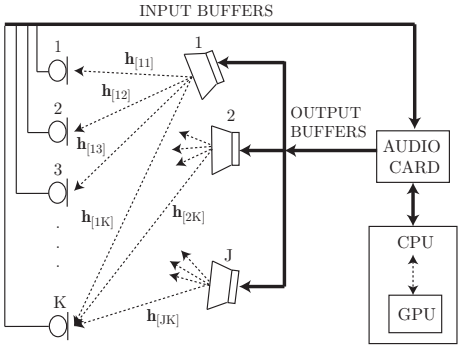


Figure 2.14. Prototype description of the GPU implementation of the multichannel room equalization system.

2.3.2 Description of the prototypes

As commented before, the three applications that have been developed are: a single-channel adaptive identification system, a multichannel adap-

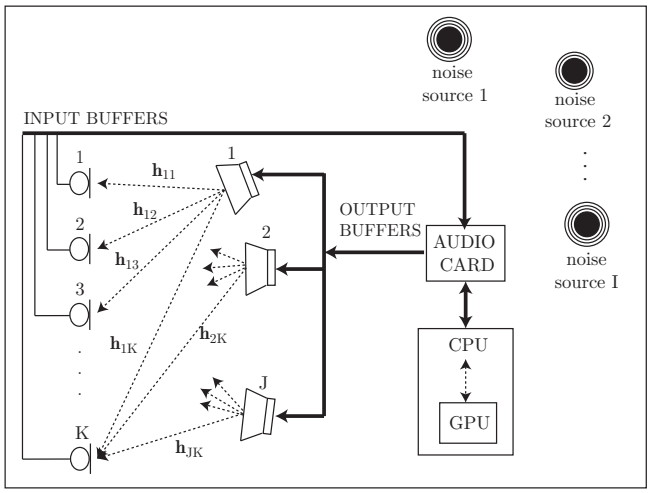


Figure 2.15. Prototype description of the GPU implementation of the multichannel ANC system.

tive equalization system and a multichannel active noise control system. Figures 2.13, 2.14 and 2.15 depicts the prototype description of these applications, respectively. With regards to the ANC prototype, the ANC system has been implemented using different algorithms. However, regardless the algorithm, the prototype is the same in all the cases. The hardware implementations of the three prototypes have similarities. They are composed by an audio card, a CPU and a GPU. The CPU is an Intel Core i7 (3.07 GHz). The GPU is a GeForce GTX 580 with Fermi architecture [11], whose main characteristics are summarized in Table 2.1. The audio card is a MOTU 24I/O. The MOTU audio card 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 are linked to the loudspeakers. It should be noted that the signal processing task is carried out by the GPU, while the CPU controls the data transfer between the input/output buffers and the GPU. The communication between the CPU and the GPU is depicted in Figure 2.16.

The operation of the GPU-based prototypes consists of the following tasks that are executed in each new iteration:

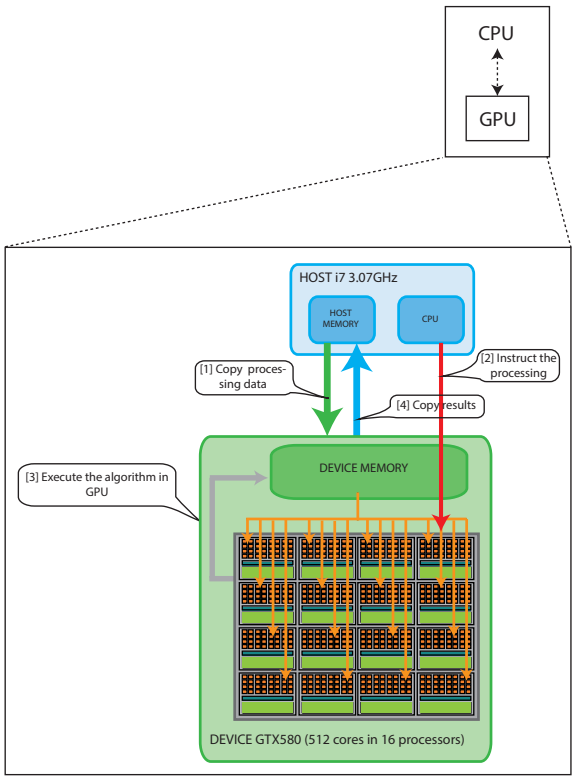


Figure 2.16. Communication between the CPU and the GPU.

1. Collect the input-data buffers 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 output-data buffers and send them back to the CPU in order to be rendered through the loudspeakers.

The sampling rate (f_s) and the block size (B) are two important parameters of the audio card. The block size describes the number of transferred discrete-time samples per iteration and thereby determines the latency of the algorithm. The latency is the time that is spent from when the input-data buffer is filled up until this data buffer is processed and sent back to the

size of B	$\mathbf{t_{buff} = B/f_s = t_{proc_{max}}}$
1	0,023 ms
32	0,73 ms
64	1,45 ms
128	2,9 ms
256	5,8 ms
512	11,6 ms
1024	23,2 ms
2048	46,4 ms

Table 2.2. Maximum processing time for a sampling rate of $f_s = 44.1$ kHz and different sizes of B

output-data buffer. The time that is spent to fill up the input-data buffers is defined as B/f_s, and we will refer to it as the buffering time (t_{buff}). The choice of the parameters B and f_s is critical for the real-time performance of the system because there is a condition that must be satisfied. We call this condition as the real time condition.

2.3.3 The real-time condition

The real-time condition defines the condition that any adaptive application has to fulfil in order to work in real time. It is given by: $t_{proc} < t_{buff}$, where t_{proc} is the execution delay measured from the moment the input-data buffer is sent to the GPU until the output-data buffer comes back to the CPU. This includes transfer delays between the CPU and the GPU and the data processing delay of the GPU.

The audio card that has been used in the prototypes offers different values of B between $B = 16$ and 2048. Moreover, the audio card offers three different sampling rates: 44.1, 44.8, and 96 kHz. It has been chosen a sampling frequency of $f_s = 44.1$ kHz in all the prototypes. Although it is the lowest rate, it is a fairly high rate for the common sounds involved in the cited applications. On the one hand, this sampling frequency offers us the possibility of controlling sounds up to 22.05 kHz, but on the other hand, as we have to satisfy the real time condition, it limits the processing time of the algorithm to a few milliseconds. Table 2.2 shows the buffering time, and therefore, the maximum processing time of the algorithm varying the

block size from $B = 1$ (sample-by-sample acquisition) to $B = 2048$. The table shows that depending on the size of B, the maximum time that the algorithm has to perform the operations varies from tens of microseconds to tens of milliseconds, which are very small time values. However, despite the algorithm has to be processed in such a short time, results of each chapter will show that the GPU is able to handle big multichannel ANC systems in real time.

For the specific case of the ANC system, there is another condition related with these parameters that has also to be satisfied. It is called the causality condition.

2.3.4 The causality condition

The ANC system has to satisfy the causality condition in order to perform properly [77]. It is given by: $t_{buff} + \tau_s < \tau_n$. The variable τ_s defines the maximum delay of the secondary paths that join the actuators with the error sensors. On the other hand, τ_n defines the minimum delay of the paths that join the noise source with the error sensors. This condition guarantees the causality of the system.

Causality is not a constraint when a harmonic excitation is considered because of the deterministic nature of the signal. However, it is important in broadband noise control. In this thesis, the multichannel ANC prototypes have been mounted inside of a listening room where both real-time and causality conditions are fulfilled by choosing the correct distances and parameters. If the location of the noise source and the environment where the noise has to be canceled do not offer the possibility to fulfill the causality conditions, the algorithm has to work with a lower block size, and, consequently, with a lower t_{buff} in order to satisfy the causality condition.

3 Description of the Algorithms

3 Description of the Algorithms

This chapter presents the algorithms that are going to be used in the adaptive applications developed in this thesis. Concretely, all the prototypes developed in this thesis are based in two algorithms: the Least Mean Square (LMS) algorithm and the Normalized Least Mean Square algorithm with Orthogonal Correlation Factors (NLMS-OCF). In a first stage, we present the LMS algorithm and its variants: the Normalized LMS (NLMS) algorithm, the Block LMS (BLMS) algorithm, the Frequency-domain BLMS (FBLMS) algorithm also called Fast BLMS, and the Partitioned FBLMS (FPBLMS) algorithm. In a second stage, the NLMS-OCF is presented.

Moreover, it is important to note that this chapter provides the basis for understanding the algorithms used in each of the applications developed in the following chapters. However, these applications use the algorithms presented in this chapter with some modifications depending on the requirements of each application.

3.1 Least mean squares

Following the nomenclature of the general adaptive filter described in the Figure 2.1, we define an L-tap adaptive filter at time n as

$$\mathbf{w}_n = [w_n(0) \;\; w_n(1) \;\; \ldots \;\; w_n(L-1)]^T, \tag{3.1}$$

where its output at time n is given by

$$y_n = \sum_{i=0}^{L-1} w_n(i) x_{n-i}. \tag{3.2}$$

The tap weights $w_n(0)$, $w_n(1)$, ..., $w_n(L-1)$ are selected so that the error signal

$$e_n = d_n - y_n, \tag{3.3}$$

is minimized in the mean-square sense. When the signals x_n and d_n are both stationary, the LMS algorithm converges to a set of tap weights which, on average, are equal to the Wiener-Hopf [16] solution. Therefore, this algorithm is in practice an scheme for realizing Wiener filters without explicitly solving the Wiener-Hopf equation.

The conventional LMS algorithm is a stochastic implementation of the steepest-descent algorithm, replacing the cost function $\xi = \mathrm{E}[e_n^2]$ by its instantaneous coarse estimate $\hat{\xi} = e_n^2$. Substituting $\hat{\xi} = e_n^2$ for ξ in the steepest-descent recursion, we obtain the adaptive filter tap weights update

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \mu \nabla_n^2. \tag{3.4}$$

where μ is the algorithm step-size parameter and ∇ is the gradient operator. The ith element of the gradient vector at time n, ∇e_n^2, is

$$\frac{\partial e_n^2}{\partial w_n(i)} = 2e_n \frac{\partial e_n}{\partial w_n(i)}, \tag{3.5}$$

and, substituting e_n from Eq. (3.3), and taking into account that d_n is independent of $w_n(i)$, we obtain

$$\frac{\partial e_n^2}{\partial w_n(i)} = -2e_n \frac{\partial y_n}{\partial w_n(i)}. \tag{3.6}$$

1. Filter output
   $y_n = \mathbf{w}_n^T \mathbf{x}_n$
2. Error estimation
   $e_n = d_n - y_n$
3. Tap-weight update
   $\mathbf{w}_{n+1} = \mathbf{w}_n + 2\mu e_n \mathbf{x}_n$

Table 3.1. Summary of the LMS algorithm.

Moreover, substituting y_n from Eq. (3.2)

$$\frac{\partial e_n^2}{\partial w_n(i)} = -2e_n x_{n-i}. \tag{3.7}$$

Finally, from these equations, we obtain that $\nabla e_n^2 = -2e_n \mathbf{x}_n$, where $\mathbf{x}_n = [x_n \;\; x_{n-1} \;\; \ldots \;\; x_{n-L+1}]^T$. Therefore, substituting it in Eq. (3.4) we get that the equation of the adaptive filter tap weights update is

$$\mathbf{w}_{n+1} = \mathbf{w}_n + 2\mu e_n \mathbf{x}_n. \tag{3.8}$$

This equation is referred to as the LMS recursion, which is a simple recursive procedure for the filter coefficients adaptation. The adaptation is performed after the arrival of every new sample of signal x, and its corresponding desired output sample, from signal d. The algorithm has three steps, which correspond with Eqs. (3.2), (3.3) and (3.8). Equation (3.2) is used to obtain the output signal. Equation (3.3) is performed to estimate the error signal. Equation (3.8) calculates the adaptation of the filter coefficients. Table 3.1 summarize the three steps required to complete each iteration of the LMS algorithm.

3.1.1 Normalized LMS

The normalized LMS (NLMS) algorithm is a variation of the LMS algorithm which takes into account the variations in the signal level at the filter input. To this end, the NLMS algorithm uses a normalized step-size parameter.

The step-size parameter is normalized by an estimate of the power of the filter input signal. It results in a stable as well as fast converging adaptation algorithm.

Considering the LMS recursion

$$\mathbf{w}_{n+1} = \mathbf{w}_n + 2\mu_n e_n \mathbf{x}_n, \tag{3.9}$$

where, now, the step-size parameter μ_n, is time-varying and defined as

$$\mu_n = \frac{\overline{\mu}}{2\mathbf{x}_n^T \mathbf{x}_n} = \frac{\overline{\mu}}{2\|\mathbf{x}_n\|^2}, \tag{3.10}$$

where $\overline{\mu}$ is a constant. The NLMS recursion can be expressed as

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \frac{e_n \mathbf{x}_n}{\|\mathbf{x}_n\|^2}. \tag{3.11}$$

3.1.2 Block LMS

The block LMS (BLMS) algorithm works on the basis of block processing. A block of samples of the filter input and the desired output are collected and then processed together to obtain a block of output samples. Therefore, the filter tap weights are updated once after the collection of every block of data samples. As a consequence, the subindex n denotes the block index.

The adaptation of the filter coefficients is given by

$$\mathbf{w}_{n+1} = \mathbf{w}_n + 2\mu_B \frac{\sum_{i=0}^{B} e_{Bn-i} x_{Bn-i}}{B}, \tag{3.12}$$

where B is the block length and μ is the algorithm step-size parameter. The calculation of the error samples at the nth block iteration is given by $e_{Bn+i} = d_{Bn+i} - y_{Bn+i}$ for $i = 0, 1, \ldots, B-1$, and the output samples are calculated using the filter coefficients vector from the previous block: $y_{Bn+i} = \mathbf{w}_n^T \mathbf{x}_{Bn+i}$.

The formulation of the BLMS is presented below. Let us define the matrix $\mathbf{X}$ as

$$\mathbf{X}_n = [\mathbf{x}_{Bn} \quad \mathbf{x}_{Bn-1} \quad \ldots \mathbf{x}_{Bn-B+1}], \tag{3.13}$$

1. Output
   $\mathbf{y}_n = \mathbf{X}_n \mathbf{w}_n$
2. Error estimation
   $\mathbf{e}_n = \mathbf{d}_n - \mathbf{y}_n$
3. Tap-weight update
   $\mathbf{w}_{n+1} = \mathbf{w}_n + 2\frac{\mu}{B}\mathbf{X}_n^T \mathbf{e}_n$

Table 3.2. Summary of the BLMS algorithm.

and the following columns vectors

$$\mathbf{x}_n = [x_{Bn} \quad x_{Bn-1} \quad \dots x_{Bn-B+1}]^T, \tag{3.14}$$

$$\mathbf{d}_n = [d_{Bn} \quad d_{Bn-1} \quad \dots d_{Bn-B+1}]^T, \tag{3.15}$$

$$\mathbf{y}_n = [y_{Bn} \quad y_{Bn-1} \quad \dots y_{Bn-B+1}]^T, \tag{3.16}$$

$$\mathbf{e}_n = [e_{Bn} \quad e_{Bn-1} \quad \dots e_{Bn-B+1}]^T. \tag{3.17}$$

The output and the error vectors are calculated as

$$\mathbf{y}_n = \mathbf{X}_n \mathbf{w}_n, \tag{3.18}$$

$$\mathbf{e}_n = \mathbf{d}_n - \mathbf{y}_n. \tag{3.19}$$

Finally, the filter coefficients update is done as in Eq. (3.12). Note that

$$\sum_{i=0}^{B} e_{Bn-i} x_{Bn-i} = \mathbf{X}_n^T \mathbf{e}_n, \tag{3.20}$$

thus, substituting Eq. (3.20) in Eq. (3.12) we obtain

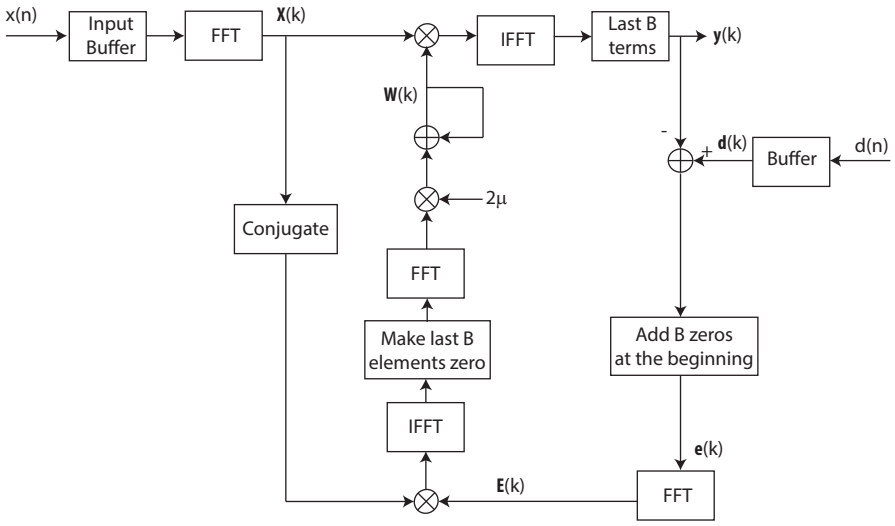


Figure 3.1. Block diagram of the FBLMS algorithm.

$$\mathbf{w}_{n+1} = \mathbf{w}_n + 2\frac{\mu}{B}\mathbf{X}_n^T\mathbf{e}_n. \tag{3.21}$$

Equations (6.16), (3.19) and (3.21) define one iteration of the BLMS algorithm, which is summarized in Table 3.2.

3.1.3 Fast BLMS

The Fast BLMS (FBLMS) algorithm is a fast implementation of the BLMS algorithm in the frequency domain. The implementation of the FBLMS algorithm is depicted in Figure 3.1. The Formulation of the algorithm is presented below.

The input vector $\mathbf{x}_n$, which is defined the same way it was in the BLMS algorithm ($\mathbf{x}_n = [x_{Bn} \quad x_{Bn-1} \quad \dots x_{Bn-B+1}]^T$), contains the last B samples of the input signal at the nth block iteration. However, let us define now $\mathbf{X}_n$ as the vector that contains the FFT of the concatenation of the last two vectors of $\mathbf{x}_n$: $\mathbf{X}_n = \mathrm{FFT}_{2B}\{[\mathbf{x}_{n-1}^T \ \mathbf{x}_n^T]^T\}$. The filtered output vector at the nth block iteration is calculated as

$$\mathbf{Y}_n = \mathbf{X}_n \circ \mathbf{W}_n, \tag{3.22}$$

1. Output
   $\mathbf{Y}_n = \mathbf{X}_n \circ \mathbf{W}_n$, where
   $[\times^T \quad \mathbf{y}_n^T]^T = \mathrm{IFFT}_{2B}\{\mathbf{Y}_n\}$.
2. Error estimation
   $\mathbf{e}_n = \mathbf{d}_n - \mathbf{y}_n$, where
   $\mathbf{E}_n = \mathrm{FFT}_{2B}\{[\mathbf{0}_B^T \quad \mathbf{e}_n^T]^T\}$.
3. Tap-weight update
   $\tilde{\boldsymbol{\mu}}_n = \mathbf{E}_n \circ \mathbf{X}_n^*$, where
   $[\boldsymbol{\phi}_n^T \quad \bar{\boldsymbol{\phi}}_n^T]^T = \mathrm{IFFT}_{2B}\{\tilde{\boldsymbol{\mu}}_n\}$, and
   $\mathbf{W}_{n+1} = \mathbf{W}_n + 2\mu\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^T \quad \mathbf{0}_B^T]^T\}$.

Table 3.3. Summary of the FBLMS algorithm.

where $\mathbf{W}_n$ is the FFT of size $2B$ of the coefficients of the adaptive filter $\mathbf{w}$ at the nth block iteration, and $\circ$ denotes the element-wise product of tho vectors. The valid samples of the adaptive filter output $\mathbf{y}_n$ are the last B samples of the IFFT of vector $\mathbf{Y}_n$.

The error vector is obtained as

$$\mathbf{e}_n = \mathbf{d}_n - \mathbf{y}_n. \tag{3.23}$$

The error vector is changed to frequency domain as follows

$$\mathbf{E}_n = \mathrm{FFT}_{2B}\{[\mathbf{0}_B^T \quad \mathbf{e}_n^T]^T\}, \tag{3.24}$$

where $\mathbf{0}_B$, is a column vector of B zeros. Once the error vector is calculated, the filter coefficients are updated in the frequency domain by performing the following equations

$$\tilde{\boldsymbol{\mu}}_n = \mathbf{E}_n \circ \mathbf{X}_n^*, \tag{3.25}$$

$$[\boldsymbol{\phi}_n^T \quad \bar{\boldsymbol{\phi}}_n^T]^T = \mathrm{IFFT}_{2B}\{\tilde{\boldsymbol{\mu}}_n\}, \tag{3.26}$$

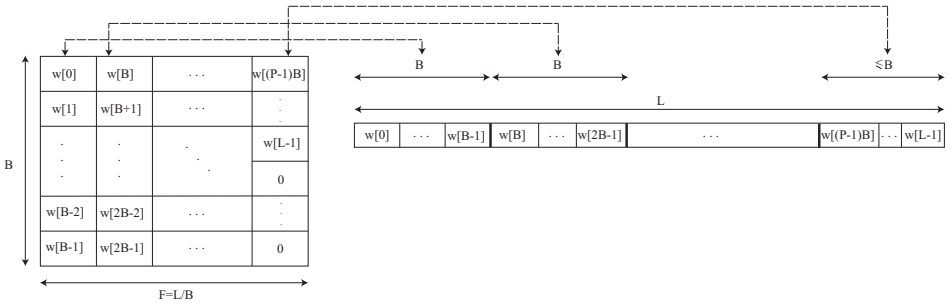


Figure 3.2. Scheme of the partition of an adaptive filter of size L into F partitions of size B.

$$\mathbf{W}_{n+1} = \mathbf{W}_n + 2\mu \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^T \quad \mathbf{0}_B^T]^T\}, \tag{3.27}$$

where $\tilde{\boldsymbol{\mu}}_n$ is a $2B$-size column vector and variables $\boldsymbol{\phi}_n$ and $\bar{\boldsymbol{\phi}}_n$ are column vectors composed by B elements. The FBLMS algorithm is summarized in Table 3.3.

3.1.4 The partitioned FBLMS

When the filter length L is larger than the block length B, an efficient implementation of the FBLMS algorithm can be derived by partitioning the adaptive filters. This leads to the Partitioned FBLMS (FPBLMS).

Let us assume that $L = F \cdot B$, where F is the number of partitions. The partition of the filters is depicted in Figure 3.2. Now, as the adaptive filters are partitioned, the Eq. (3.22) has to be rewritten as

$$\mathbf{Y}_n = \sum_{f=1}^{F} \mathbf{X}_{n-f+1} \circ \mathbf{W}_n^f, \tag{3.28}$$

where superscript f denotes the partition number and $\mathbf{W}_n^f$ is the FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}_n$. Figure 3.3 represents the arrangement of $\mathbf{X}_n$ while Figure 3.4 depicts the implementation of Eq. (3.28) when $F = 3$.

As in the FBLMS algorithm, the valid samples of the adaptive filter output $\mathbf{y}_n$, are the last B samples of the IFFT of vector $\mathbf{Y}_n$.

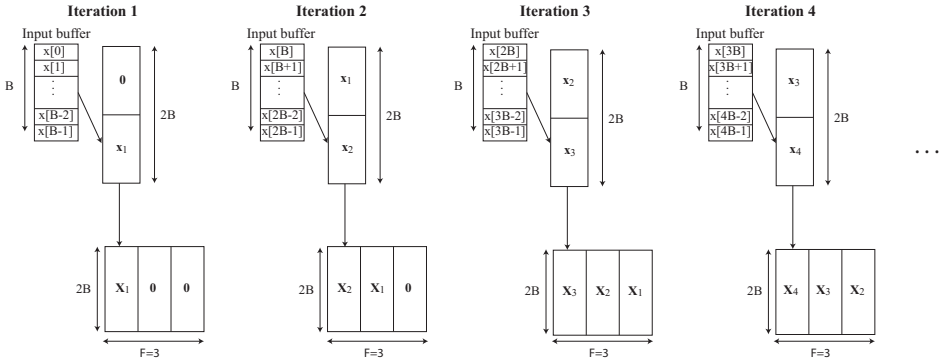


Figure 3.3. Arrangement of input matrix $\mathbf{X}_n$ when $F = 3$.

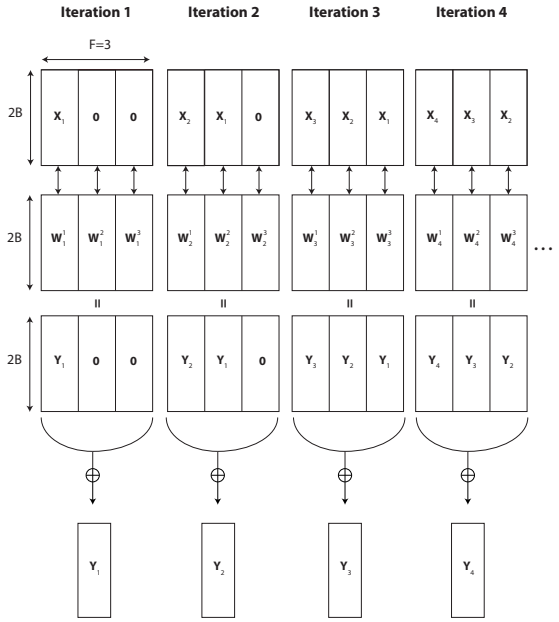


Figure 3.4. Implementation of Equation (3.28).

The equations for obtaining the error vector in frequency domain are Eq. (3.23) and (3.24), but now, the filter coefficients are updated each partition independently by performing the following equations

$$\tilde{\boldsymbol{\mu}}_n^f = \mathbf{E}_n \circ \mathbf{X}_{n-f+1}^*, \tag{3.29}$$

1. Output
   $\mathbf{Y}_n = \sum_{f=1}^{F} \mathbf{X}_{n-f+1} \circ \mathbf{W}_n^f$, where
   $[\times^T \quad \mathbf{y}_n^T] = \mathrm{IFFT}_{2B}\{\mathbf{Y}_n\}$
2. Error estimation
   $\mathbf{e}_n = \mathbf{d}_n - \mathbf{y}_n$, where
   $\mathbf{E}_n = \mathrm{FFT}_{2B}\{[\mathbf{0}_B^T \quad \mathbf{e}_n^T]^T\}$.
3. Tap-weight update
   $\tilde{\boldsymbol{\mu}}_n^f = \mathbf{E}_n \circ \mathbf{X}_{n-f+1}^*$, where
   $[\boldsymbol{\phi}_n^{fT} \quad \bar{\boldsymbol{\phi}}_n^{fT}]^T = \mathrm{IFFT}_{2B}\{\tilde{\boldsymbol{\mu}}_n^f\}$, and
   $\mathbf{W}_n^f = \mathbf{W}_n^f + 2\mu\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^{fT} \quad \mathbf{0}_B{}^T]^T\}$.

Table 3.4. Summary of the FPBLMS algorithm.

$$[\boldsymbol{\phi}_n^{fT} \quad \bar{\boldsymbol{\phi}}_n^{fT}]^T = \mathrm{IFFT}_{2B}\{\tilde{\boldsymbol{\mu}}_n^f\}, \tag{3.30}$$

$$\mathbf{W}_n^f = \mathbf{W}_n^f + 2\mu\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^{fT} \quad \mathbf{0}_B{}^T]^T\}. \tag{3.31}$$

The FPBLMS algorithm is summarized in Table 3.4.

3.2 The normalized least mean square with orthogonal correction factors

To better understanding of the NLMS-OCF algorithm, let us start with a reminder of the NLMS algorithm:

- Filter output
  $y_n = \mathbf{w}_n^T \mathbf{x}_n$
- Error estimation
  $e_n = d_n - y_n$
- Tap-weight update
  $\mathbf{w}_{n+1} = \mathbf{w}_n + \mu \frac{\mathbf{x}_n e_n}{\|\mathbf{x}_n\|^2}$.

If the desired output $d_n = \mathbf{w}_n^{0T}\mathbf{x}_n$, comes from a FIR system that can be modeled with weights $\mathbf{w}^0$, and there is no measurement of noise, the minimum achievable mean square estimation error is zero. Under this condition, for a $\mu = 1$ and replacing $e_n = \mathbf{w}_n^{T0}\mathbf{x}_n - \mathbf{w}_n^T\mathbf{x}_n = (\mathbf{w}_n^0 - \mathbf{w}_n)^T\mathbf{x}_n$:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \frac{\mathbf{x}_n \cdot e_n}{\|\mathbf{x}_n\|^2} = \mathbf{w}_n + \frac{\mathbf{x}_n(\mathbf{w}_n^0 - \mathbf{w}_n)^T\mathbf{x}_n}{\|\mathbf{x}_n\|^2}. \tag{3.32}$$

Considering that $\frac{(\mathbf{w}_n^0 - \mathbf{w}_n)^T\mathbf{x}_n}{\|\mathbf{x}_n\|^2}$ is an scalar that we call P, the weight update equation is:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mathbf{x}_n \cdot P. \tag{3.33}$$

A geometric interpretation of Eq. (3.32) is shown in Figure 3.5. If $\mathbf{w}_n$ are the weights of the current iteration, the new weights $\mathbf{w}_{n+1}$ are the point that is nearest (in $\|\cdot\|_2$ sense) to the desired weights ($\mathbf{w}^0$) along the direction specified by the input vector $\mathbf{x}_n$. In other words, $\mathbf{w}_{n+1}$ is the projection of $\mathbf{w}^0$ in the direction specified by $\mathbf{x}_n$.

Considering L as the number of coefficients of $\mathbf{w}^0$, if the L consecutive input vectors $\mathbf{x}_{n+l}$ for $l = 0, \ldots, L-1$ used in the tap weight update are orthogonal, the NLMS algorithm minimizes the distance from the true weights $\mathbf{w}^0$ along L orthogonal directions and thereby converges to the desired weights in exactly L iterations. This is illustrated in Figure 3.6. For simplicity, it is illustrated in a bi-dimensional space using filters that are composed of two weights. In this case, as the consecutive input vectors $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$ are orthogonal, the NLMS algorithm converges to the desired weights in two iterations. However, when the input signal is strongly colored, the successive input vectors tend to be almost parallel to each other and, therefore, the estimated weights improve very little during successive iterations. As an example, see Figure 3.7. This behavior motivated to introduce orthogonal correction factors to the NLMS algorithm to accelerate its convergence.

The basic idea of the NLMS-OCF algorithm is to update the weights on the basis of multiple input vectors in contrast to the NLMS algorithm that uses a single input vector. If the successive input vectors used in the weight update are orthogonal in nature, the convergence of the algorithm is accelerated. When the successive input vectors are not orthogonal, this al-

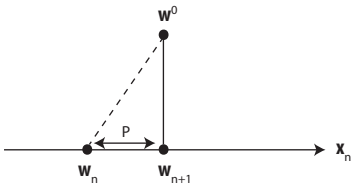


Figure 3.5. Geometric interpretation of the filter coefficients update by the NLMS algorithm.

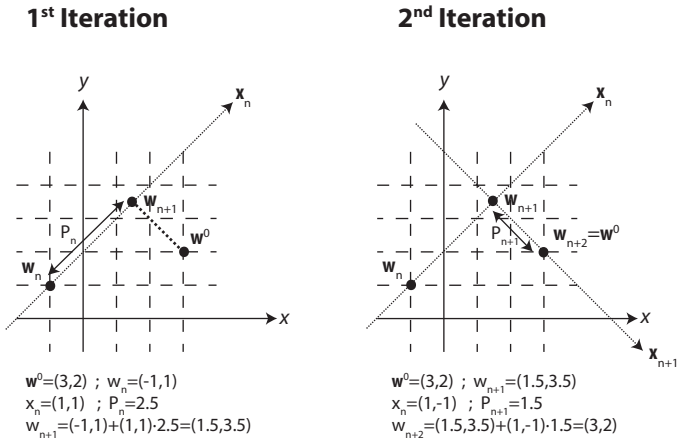


Figure 3.6. Geometric interpretation of the exact convergence when using orthogonal directions.

gorithm proposes to generate appropriate orthogonal input vectors. To this end, the equation of the filter coefficients update of the NLMS is modified as follows

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu_0 \mathbf{x}_n + \mu_1 \mathbf{x}_{1_n} + \ldots + \mu_R \mathbf{x}_{R_n}, \quad n \geq R < L \tag{3.34}$$

where the vectors $\mathbf{x}_n, \mathbf{x}_{1_n}, \mathbf{x}_{2_n}, \ldots, \mathbf{x}_{R_n}$ are orthogonal to each other. The procedure to generate these orthogonal input vectors and the corresponding step-sizes is explained below.

As in NLMS, $\mathbf{x}_n$, is the input vector at the nth instant, and, μ_0 is chosen as

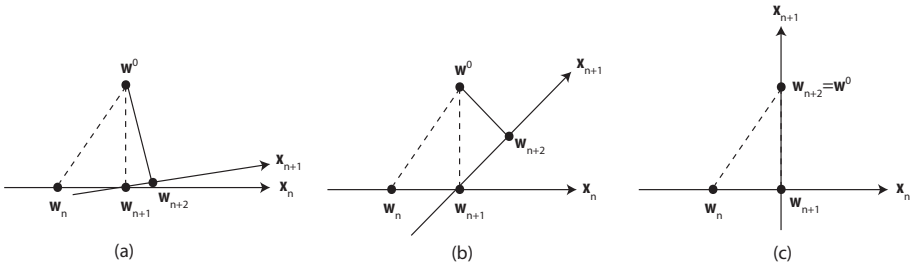


Figure 3.7. Geometric interpretation of the filter coefficients update by the NLMS algorithm. (a) Almost parallel input, small improvement. (b) Differently oriented input, large improvement. (c) Orthogonal input, most improvement.

$$\mu_0 = \frac{\overline{\mu} e_n}{\|\mathbf{x}_n\|^2}. \tag{3.35}$$

Considering $\mathbf{w}_{1_{n+1}} = \mathbf{w}_n + \mu_0 \mathbf{x}_n$ as the new estimate for the weights obtained after the correction along $\mathbf{x}_n$. Using Gram-Schmidt procedure [78], we write $\mathbf{x}_{n-D}$ $(D > 0)$ as the sum of a component along $\mathbf{x}_n$ and a component orthogonal to $\mathbf{x}_n$. Let the orthogonal component be $\mathbf{x}_{1_n}$. Then the step-size along $\mathbf{x}_{1_n}$ is

$$\mu_1 = \begin{cases} \dfrac{\overline{\mu} e_{1_n}}{\|\mathbf{x}_{1_n}\|^2}, & \text{if } \|\mathbf{x}_{1_n}\|^2 \neq 0 \\ 0, & otherwise. \end{cases} \tag{3.36}$$

where $e_{1_n} = d_{n-D} - \mathbf{x}_{n-D}^t \mathbf{w}_{1_{n+1}}$ is the error in estimating d_{n-D} using $\mathbf{w}_{1_{n+1}}$.

Considering R orthogonal correction factors, the above steps can be generalized for a given orthogonal correction factor $0 \leq r \leq R$. Vector $\mathbf{x}_{r_n}$ is the component of $\mathbf{x}_{n-rD}$ that is orthogonal to $\mathbf{x}_n$, $\mathbf{x}_{n-1D}$, $\mathbf{x}_{n-2D}$, $\ldots$, $\mathbf{x}_{n-(r-1)D}$ and it can be computed using the Gram-Schmidt procedure. The corresponding step-size μ_r is calculated as follows

$$\mu_r = \begin{cases} \dfrac{\overline{\mu} e_{r_n}}{\|\mathbf{x}_{r_n}\|^2}, & \text{if } \|\mathbf{x}_{r_n}\|^2 \neq 0 \\ 0, & otherwise. \end{cases} \tag{3.37}$$

Choose an arbitrary $\mathbf{w}_n$ and the number of Orthogonal Correction Factors $R < L$. Repeat the following steps at each new iteration.

1. $e_n = d_n - \mathbf{x}_n^T \mathbf{w}_n$
2. $\mu_0 = \frac{\overline{\mu} e_n}{\|\mathbf{x}_n\|^2}$
3. $\mathbf{w}_{1_{n+1}} = \mathbf{w}_n + \mu_0 \mathbf{x}_n$
4. $\mathbf{x}_{0_n} = \mathbf{x}_n$

For all $1 \leq r \leq R$, do

6. Compute $\mathbf{x}_{r_n}$ as the vector obtained from $\mathbf{x}_{n-rD}$ that is orthogonal to $\mathbf{x}_{0_n}$, $\mathbf{x}_{1_n}$, $\mathbf{x}_{2_n}$, ..., $\mathbf{x}_{r-1_n}$, using the Gram-Schmidt procedure [78].

   $\mathbf{x}_{r_n} = \mathbf{x}_{n-rD} - \sum_{i=0}^{r-1} \frac{\mathbf{x}_{n-rD}^T \mathbf{x}_{i_n}}{\|\mathbf{x}_{i_n}\|^2} \mathbf{x}_{i_n}$
7. $e_{r_n} = d_{n-rD} - \mathbf{x}_{n-rD}^T \mathbf{w}_{r_{n+1}}$
8. $\mu_r = \begin{cases} \dfrac{\overline{\mu} e_{r_n}}{\|\mathbf{x}_{r_n}\|^2}, & \text{if } \|\mathbf{x}_{r_n}\|^2 \neq 0 \\ 0, & otherwise. \end{cases}$
9. $\mathbf{w}_{r+1_{n+1}} = \mathbf{w}_{r_{n+1}} + \mu_r \mathbf{x}_{r_n}$

end for

15. $\mathbf{w}_{n+1} = \mathbf{w}_{R+1_{n+1}}$

Table 3.5. Summary of the NLMS-OCF algorithm.

where

$$e_{r_n} = d_{n-rD} - \mathbf{x}_{n-rD}^T \mathbf{w}_{r_{n+1}}, \tag{3.38}$$

and

$$\mathbf{w}_{r+1_{n+1}} = \mathbf{w}_n + \mu_0 \mathbf{x}_n + \mu_1 \mathbf{x}_{1_n} + \ldots + \mu_r \mathbf{x}_{r_n} = \mathbf{w}_{r_{n+1}} + \mu_r \mathbf{x}_{r_n}. \quad (3.39)$$

The NLMS-OCF algorithm is summarized in Table 3.5.

Channel identification 4

Channel identification 4

One of the simplest audio applications that require real-time feedback is the adaptive channel identification. This chapter discusses the design and implementation of an adaptive channel identification system on a GPU. Concretely, the LMS algorithm has been implemented in the frequency domain. The size of the 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. The goal is to propose a GPU implementation that can be easily adapted to any other acoustic scenario, while freeing up CPU resources for other tasks. The channel identification application will be necessary for other audio applications like room equalization (chapter 5) or active noise control (chapters 6 and 7).

4.1 Introduction

As commented in section 2.2.1, in recent years, the number of scientific contributions and research projects related to the use of GPUs for signal processing applications has significantly increased. However, most of the applications that have been using GPUs for accelerating the audio processing consist in convolving multiple input signals with static filters. On

the other hand, in the adaptive filtering context, the GPUs have been rarely used. This is because there are some constraining aspects regarding the GPU implementation of an adaptive application. These constraints are related with the iterative execution of the signal acquisition, the data transactions between CPU, GPUs and audio cards, and the loudspeaker reproduction. For this reason, this thesis has studied the viability of using the GPUs for real-time adaptive filtering applications dealing with these constraining aspects.

The purpose of this chapter is to describe how to use GPUs for real-time audio applications based on adaptive algorithms where the adaptive filter coefficients vary over time in order to reach a target. The target depends on the application. Among the wide variety of adaptive filtering applications, this thesis has mainly focused on the active noise control application. However, as shown in Figure 2.5, the ANC systems have to previously estimate or identify the plant between the error sensors and the secondary sources. This leads to a system identification problem. The system identification is such a fundamental discipline, with such wide-ranging and cross-disciplinary applications that has been widely studied in last decades, resulting in many techniques that have been presented in literature. In [79], the author describes the development of the system identification from the early works of [80] and [81] up to nowadays. It describes the evolution of the identification techniques from the traditional methods such as the maximum likelihood method [80] to the identification of nonlinear systems, going through the sub-spaced identification, the identification for control or the frequency-domain identification to cite some examples.

Although the adaptive algorithms for channel identification are not the system identification technique most used nowadays, it is a simple technique based on adaptive filtering. For this reason, and to begin with the viability study of the use of GPUs in the adaptive filtering context, an adaptive single-channel identification application based on the LMS algorithm has been developed using a GPU as the main processor. Therefore, this application has been chosen for its simplicity in order to set the basis of the use of the GPU in real-time adaptive applications. Then in the following chapters, this study is expanded to multichannel applications such as adaptive equalization (chapter 5) or active noise control (chapters 6 and 7).

On the other hand, because of the inherent capability of parallelization of the GPUs, not all the adaptive algorithms can run efficiently on these

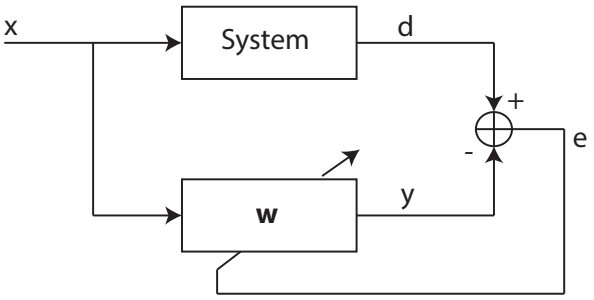


Figure 4.1. Block diagram of an adaptive channel identification system.

systems. As an example, the adaptive algorithms that work sample-by-sample are particularly difficult to optimize in GPUs. Therefore, in this study we have used a block-based algorithm that optimizes its computational cost through its implementation in the frequency domain. However, a block-based algorithm introduces the disadvantage of a higher latency. In order to illustrate how to overcome the difficulties that appear due to the use of GPUs in adaptive systems, we have implemented a single-channel identification system using the FBLMS algorithm, which is a computationally efficient version in the frequency domain of the BLMS algorithm. As it is explained in the following chapters, most of the actions described in this chapter, can be extended to other adaptive applications.

4.2 Description of the algorithm

Figure 4.1 shows the general scheme of an adaptive channel identifier. Let us consider that the block labeled with 'system' is an electro-acoustic system that can be described by an impulse response between two points in space (usually located in a room). Moreover, we assume that the 'system' can be modeled by a finite impulse response filter. Under these conditions, signal x is defined as the reference signal, while signal d is the result of exciting the 'system' with signal x. Furthermore, we consider the vector $\mathbf{w}$ as the adaptive FIR filter that models the 'system'. Consequently, signal y is the adaptive filter output, and finally, e is the error signal.

In the case of our study, the system is excited with a broadband signal x. The target is the cancellation of the error signal, which is defined as:

Figure 4.2. Reordering of the data input blocks.

$e = d - y$. The cancellation of the error signal means that the coefficients of the adaptive filter fit the coefficients of the unknown system. The adaptive algorithms minimize the error signal e or some function related with this signal, such that w at steady state is a good estimation of the impulse response of the system that has to be identified.

The LMS algorithm is one of the most commonly used adaptive algorithms due to its good performance as well as simplicity and robustness. For this reasons, the adaptive channel identification system has been implemented using the LMS algorithm. Moreover, as long as the parallel resources of a GPU are better exploited working with block-data buffers than sample-by-sample acquisition, the LMS algorithm has been implemented with a block-data processing, resulting in the BLMS algorithm. Finally, the BLMS algorithm has been efficiently implemented in the frequency domain providing the FBLMS algorithm presented in chapter 3. Thus, a computational cost reduction is achieved with respect to the BLMS algorithm in time domain by efficiently computing the Discrete-time Fourier Transform (DFT) with the Fast Fourier Transform (FFT). The implementation of the FBLMS algorithm, is explained below.

Let us define the input signal vector at the nth block iteration as $\mathbf{x}_n = [x_n \;\; x_{n-1} \;\; \dots \;\; x_{n-B+1}]^T$. The input-data vector has to be arranged as in Figure 4.2, forming a vector of size $2B$ that is composed of the concatenation of the last two vectors of the input signal. Then, the concatenation of the input vectors is transformed into the frequency domain obtaining vector $\mathbf{X}_n$. It is done by applying an FFT of size $2B$: $\mathbf{X}_n = \mathrm{FFT}_{2B}\{[\mathbf{x}_{n-1}^T \;\; \mathbf{x}_n^T]^T\}$.

The filtered output vector at the nth block iteration is calculated as

$$\mathbf{Y}_n = \mathbf{X}_n \circ \mathbf{W}_n, \tag{4.1}$$

where $\mathbf{W}_n$ is the FFT of size $2B$ of the coefficients of the adaptive filter $\mathbf{w}$ at the nth block iteration, and $\circ$ denotes the element-wise product of the vectors. The valid samples of the adaptive filter output $\mathbf{y}_n$ are the last B samples of the IFFT of vector $\mathbf{Y}_n$.

The error vector is obtained as

$$\mathbf{e}_n = \mathbf{d}_n - \mathbf{y}_n. \tag{4.2}$$

The error vector is transformed into the frequency domain by performing an FFT of size $2B$ to the error signal $\mathbf{e}_n$ filling its first B elements with zeros. It is done as follows

$$\mathbf{E}_n = \mathrm{FFT}_{2B}\{[\mathbf{0}_B^T \;\; \mathbf{e}_n^T]^T\}, \tag{4.3}$$

where $\mathbf{0}_B$, is a columns vector of B zeros. Once the error vector is calculated, the filter coefficients are updated in the frequency domain by performing the following equations

$$\tilde{\boldsymbol{\mu}}_n = \mathbf{E}_n \circ \mathbf{X}_n^*, \tag{4.4}$$

$$[\boldsymbol{\phi}_n^T \;\; \bar{\boldsymbol{\phi}}_n^T]^T = \mathrm{IFFT}_{2B}\{\tilde{\boldsymbol{\mu}}_n\}, \tag{4.5}$$

$$\mathbf{W}_{n+1} = \mathbf{W}_n + 2\mu\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^T \;\; \mathbf{0}_B^T]^T\}. \tag{4.6}$$

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 filter with a maximum of L coefficients, which in turn imposes a constraint when setting the size of B. One solution to estimate larger response systems regardless the size of B, is to split up the adaptive filter coefficients into blocks called partitions [82]. Let us call define $F = L/B$ as the number of partitions. Thus, the L coefficients are partitioned into F partitions of size B. This means that the algorithm can be parallelized in order to work simultaneously with the F partitions of size B 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 by performing the adaptation of each partition of the filters at the same time. The resulting implementation is called the partitioned FBLMS (FPBLMS) algorithm, which is also explained in chapter 3.

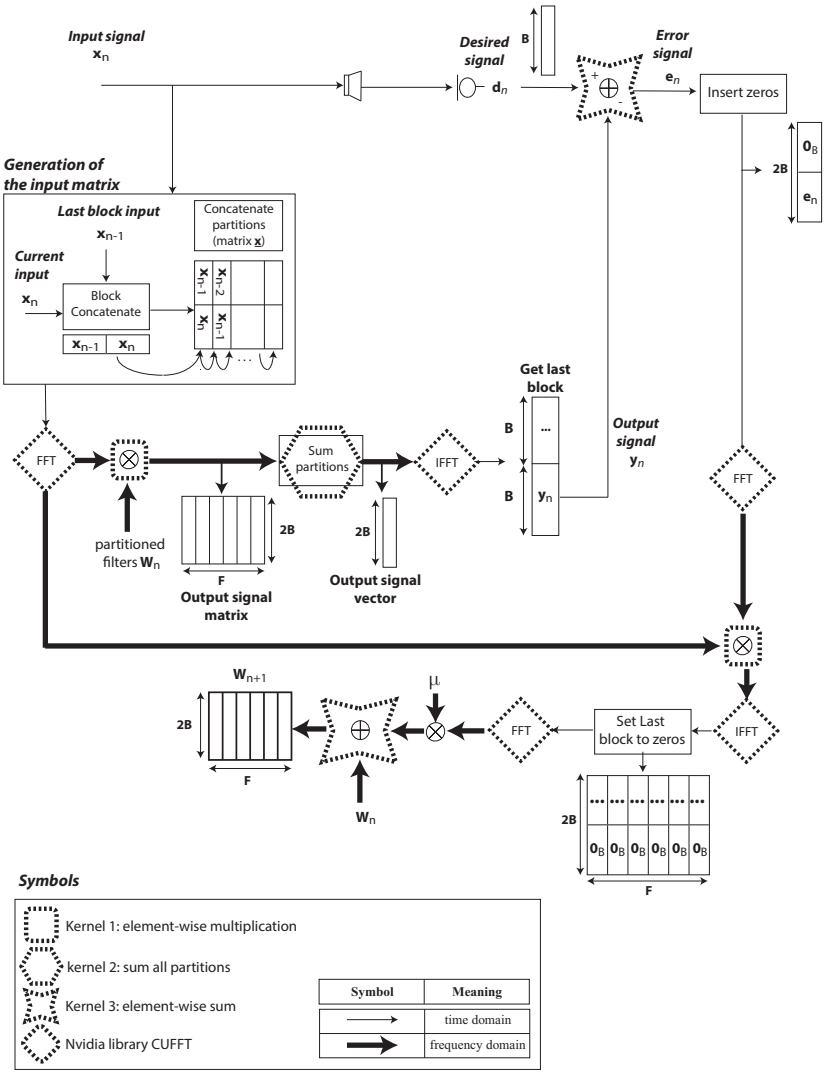


Figure 4.3. Block diagram of the GPU implementation of the FPBLMS algorithm for a channel identification application at the nth block iteration.

4.3 GPU implementation of the prototype

This section is devoted to explain the main issues of the GPU implementation of the channel identification prototype. The prototype description

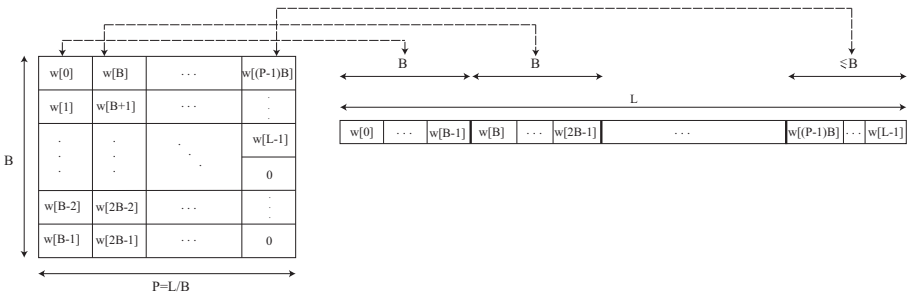


Figure 4.4. Partition of the adaptive filter.

as well as the real-time constraining aspects has been considered in section 2.3.

Figure 4.3 shows the GPU implementation of one iteration of the FPBLMS algorithm for a single-channel identification application. First of all, as the length of the adaptive filter needed in the application is longer than the block sizes offered by the audio card, the FBLMS algorithm has been implemented by partitioning the adaptive filter, resulting in the FPBLMS algorithm. The partitioning of the adaptive filters occurs in applications where the filter size is longer than the input-data buffer or in applications where the latency time requires minimizing the input-data buffer [82, 83]. In our case, by partitioning the filter, we also obtain the best performance from the resources of the GPU by making use of the SIMD GPU architecture. This means that all the partitions of the filter can be updated concurrently. Considering a filter size of L, and a block size of B, the adaptive filter $\mathbf{w}$ is uniformly partitioned in $F = L/B$ blocks of B samples. If L is not a multiple of B, the last block is padded with zeros. Figure 4.4 illustrates the partition of the filter. As the update of the filters is done in frequency domain, an FFT of size $2B$ of each partition has to be done. This results in a coefficient matrix of size $2B \times F$ called $\mathbf{W}$

Once the filter coefficients are arranged as in Figure 4.4, the channel estimation is carried out by performing the FPBLMS algorithm. Table 4.1 shows a pseudo-code of the steps needed for the GPU implementation. Moreover, the GPU uses the kernels described in Table 4.2. All the steps are detailed below:

Step 1 First, as the adaptive filter matrix in frequency domain, $\mathbf{W}$, is

	CHANNEL ESTIMATION PSEUDO CODE
Step 1	Build input data matrix $\underline{\mathbf{x}}$ (see figure 4.3).
Step 2	FFT of size $2B$ of each column of matrix $\underline{\mathbf{x}}$.
Step 3	Perform an element-wise matrix multiplication: $\mathbf{Y} = \underline{\mathbf{X}} \circ \mathbf{W}$.
Step 4	Summation of all the columns of the matrix $\mathbf{Y}$, all the columns are reduced to a single one.
Step 5	Carry out an IFFT of size $2B$ of the result obtained in step 4. Discard the first B samples and save the last B samples as the output vector in time domain $\mathbf{y}$.
Step 6	Calculate the B elements of the error signal vector in time domain as: $\mathbf{e} = \mathbf{d} - \mathbf{y}$.
Step 7	Perform an FFT of size $2B$ of the error signal vector after filling its first B elements with zeros: $\mathbf{E} = \mathrm{FFT}_{2B}\{[\mathbf{0}_B^T \ \ \mathbf{e}^T]^T\}$.
Step 8	Calculate the IFFT of the autocorrelation between the error signal vector and the reference signal vector as: $\boldsymbol{\phi} = \mathrm{IFFT}_{2B}\{\mathbf{E} \circ \underline{\mathbf{X}}^*\}$.
Step 9	Discard the last B elements of each column of matrix $\boldsymbol{\phi}$ and fill it with zeros. Then perform an FFT of size $2B$ of each columns of the resulting matrix to obtain the step size variable needed for updating the filter coefficients: $\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^T \ \ \mathbf{0}_B^T]^T\}$.
Step 10	Update the filter coefficients as: $\mathbf{W} = \mathbf{W} + \mu\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^T \ \ \mathbf{0}_B^T]^T\}$.

Table 4.1. The steps of the GPU implementation of the FP-BLMS algorithm.

of size $2B \times F$, the algorithm builds an input data matrix $\underline{\mathbf{x}}$ of size $2B \times F$ in order to fit the element-wise multiplication when filtering the input data with the adaptive filter. The building of matrix $\underline{\mathbf{x}}$ is depicted in Figure 4.5. Matrix $\underline{\mathbf{x}}$ is formed by filling each column of the matrix with the input-data buffer of the corresponding iteration. Concretely, the first column of matrix $\underline{\mathbf{x}}$ at the nth block iteration is filled with the B samples of the input-data buffer of the *(n-1)*th iteration followed by the B samples of the input-data buffer of the nth iteration. The second column would be filled with the input-data buffer of the *(n-2)*th iteration followed by the input-data buffer of the *(n-1)*th iteration. The same way, the third column would be filled with the input-data buffer of the *(n-3)*th iteration followed by the input-data buffer of the *(n-2)*th iteration, and so on. In the first

Kernel 1	*This kernel launches bi-dimensional blocks of threads. It launches $2B \cdot F$ threads in total, where the dimension of the blocks is (x, y, 1), and the dimensions of the grid is (F/x, 2B/y, 1). The kernel uses each thread for processing each sample. Thus, each thread performs a complex multiplication between elements of each matrix.*
Kernel 2	*This kernel uses one-dimensional blocks of threads. The dimension of the blocks is (1, y, 1) and the dimension of the grid is (1, 2B/y, 1), where 'y' is the number of threads in each block. Therefore, the kernel launches $2B$ threads in total. Each thread carries out F sums. The result is a column vector of $2B$ elements. Each element contains the reduction sum of the elements of each row.*
Kernel 3	*This kernel launches B threads divided in one-dimensional blocks. The dimension of the blocks is (1, y, 1) and the dimension of the grid is (1, B/y, 1). Each thread performs a subtraction between the values of vectors $\mathbf{d}$ and $\mathbf{y}$.*
Kernel 4	*It is similar to Kernel 1. The difference is that in this case $\mathbf{E}$ is a vector, so each column of matrix $\underline{\mathbf{X}}$ is element-wise multiplied by the same vector $\mathbf{E}$. The dimensions of the blocks are (1,y,1) while the dimensions of the grid is (1,2B/y,1). Therefore, it launches 2B threads, where each thread perform an element-wise multiplication. Also note that before the element-wise multiplication, the kernel carries out a conjugation of the elements of matrix $\underline{\mathbf{X}}$.*
Kernel 5	*It has the same thread configuration of kernel 1, but each thread performs a sum instead of a multiplication.*

Table 4.2. Kernel configurations

iterations, the matrix $\underline{\mathbf{x}}$ is filled by zeros, but after F iterations, all columns are filled with input-data buffers. Then, at iteration $F + 1$ the column filled in the first iteration is rewritten using the current input data buffer.

Step 2 Once the input matrix $\underline{\mathbf{x}}$ is arranged, it must be filtered with the adaptive filter. For this purpose, matrix $\underline{\mathbf{X}}$ is formed by performing an FFT of size $2B$ of each column of matrix $\underline{\mathbf{x}}$. Then, an element-wise multiplication between the input matrix $\underline{\mathbf{X}}$ and the filter coefficients matrix $\mathbf{W}$ is carried out. Matrix $\mathbf{W}$ is a $2B \times F$ matrix whose columns are the FFT of size $2B$ of each partition of B time-

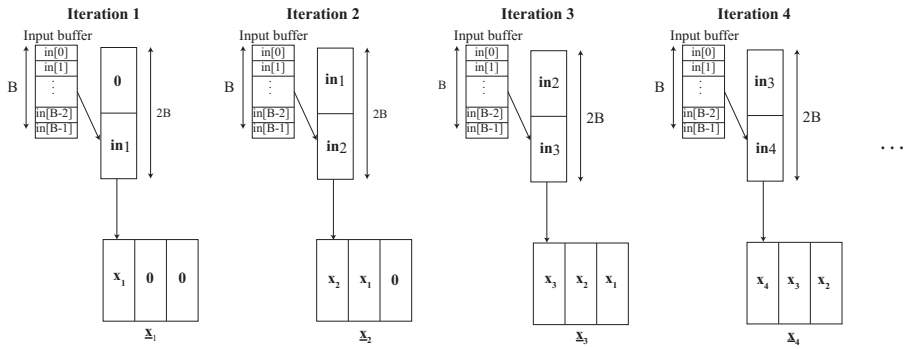


Figure 4.5. Example of input-data matrix when P=3.

coefficients padded with B zeros. For the FFT implementation, we have used the NVIDIA cuFFT library [11]. This library allows multiple one-dimensional FFTs to be carried out simultaneously. Therefore, it allows to perform the F FFTs of the F partitions concurrently. When the F FFTs have been carried out, the element-wise multiplication is performed launching the CUDA kernel 1 (see kernel 1 in Table 4.2).

Figure 4.6 illustrates two schemes of the complex element-wise matrix multiplication:

1. The elements of the fth column are ordered in the same position in both matrices (scheme 'A').
2. The elements of the fth column are not ordered in the same position in the two matrices (scheme 'B').

The scheme 'A' shows the direct implementation, where the input-data matrix is ordered so that the elements of the fth column of matrix $\underline{\mathbf{X}}$ are element-wise multiplied by elements of the fth column of matrix $\mathbf{W}$. This scheme has the disadvantage that all the columns except one of the matrix $\underline{\mathbf{X}}$ are moved at each iteration, so it involves a copy of $2B(F-1)$ elements in GPU memory at each iteration. The copied data is represented in grey, and the current input buffer is marked by an arrow. On the other hand, the scheme 'B' shows an optimized multiplication, where the current input buffer is placed in the corresponding column avoiding GPU memory transactions. Therefore, in order to achieve the same final result in both

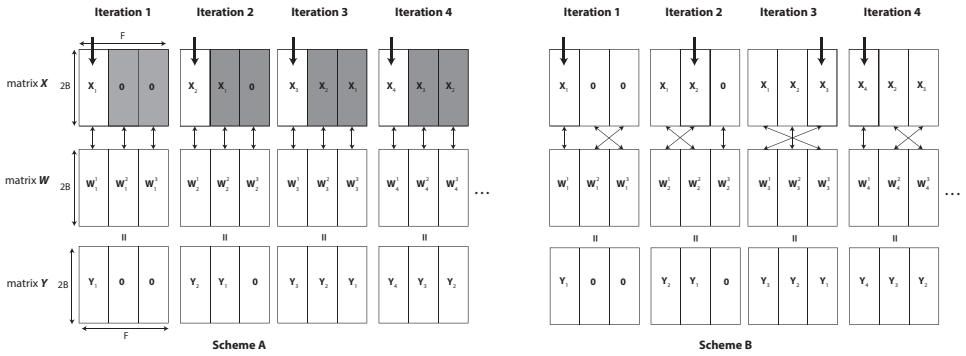


Figure 4.6. Two different schemes of an element-wise multiplication for the case F=3.

schemes, we have to redefine the thread memory access of the GPU for multiplying the corresponding elements of the two matrices.

Step 3 The multiplication of Step 2 generates an output matrix $\mathbf{Y}$ of size $2B \times F$. Step 3 performs a sum of the F partitions of the output matrix $\mathbf{Y}$, reducing all the columns to a single one and resulting in a vector of size $2B$. See Figure 4.7 and kernel 2 in Table 4.2.

Step 4 Next step consists in performing a one-dimensional IFFT of size $2B$ of the vector obtained in step 3. The last B samples correspond to the output signal of the adaptive filter in time domain, $\mathbf{y}$. 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 B at each iteration as: $\mathbf{e} = \mathbf{d} - \mathbf{y}$. Kernel 3 implements this subtraction.

Step 6 This step calculates the error vector in frequency domain, $\mathbf{E}_n$. To this end, first, the error vector of size B in time domain, $\mathbf{e}$, is padded with B zeros. Then an FFT of size $2B$ is performed. $\mathbf{E} = \mathrm{FFT}_{2B}\{[\mathbf{0}_B^T \;\; \mathbf{e}^T]^T\}$.

Step 7 Step 7 computes the IFFT of the correlation between the error vector $\mathbf{E}$ and each column of the input matrix $\underline{\mathbf{X}}^*$. At the nth iteration, it is done performing the equation: $\boldsymbol{\phi}_n^f = \mathrm{IFFT}_{2B}\{\mathbf{E}_n \circ \mathbf{X}_{n-f+1}^*\}$. The resulting matrix $\boldsymbol{\phi}$ is of size $2B \times F$. Kernel 4 is implemented to perform the $2B$ element-wise multiplication of the vector $\mathbf{E}$ with each column of the matrix $\underline{\mathbf{X}}^*$.

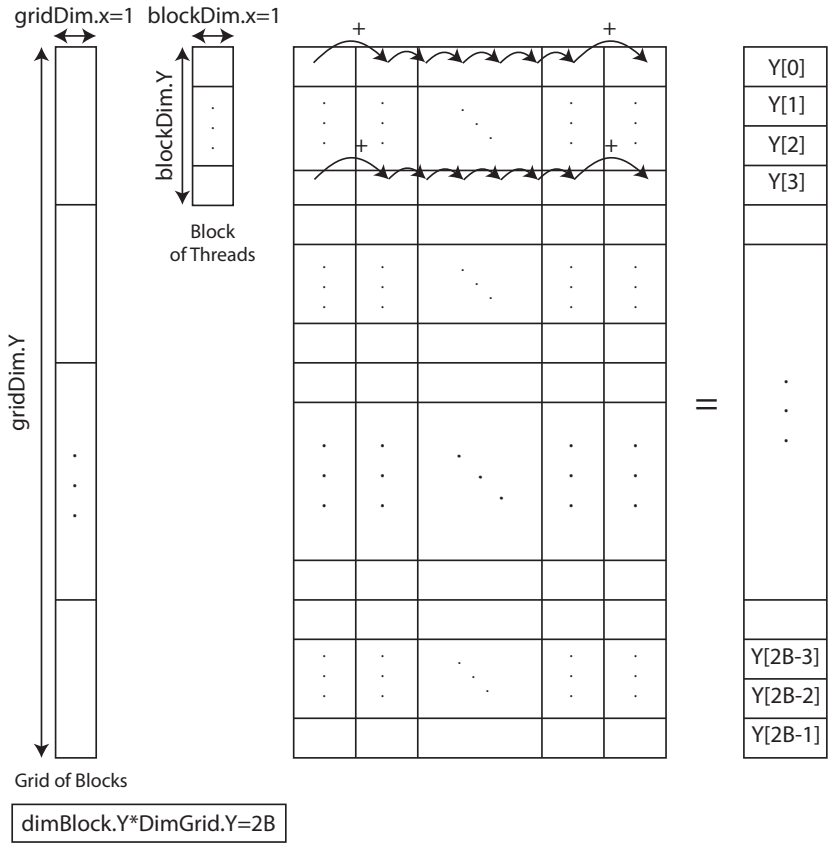


Figure 4.7. Representation of the CUDA Kernel 2.

Step 8 The last B elements of each column of matrix $\boldsymbol{\phi}$ are discarded and padded with zeros. Then, an FFT of size $2B$ of each columns of the resulting matrix is performed: $\mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^{fT} \quad \mathbf{0}_B^T]^T\}$.

Step 9 Finally kernel 5 (see Table 4.2) is implemented to update each partition of the filter matrix. It is done at the nth iteration performing the equation: $\mathbf{W}_{n+1}^f = \mathbf{W}_n^f + \mu \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}_n^{fT} \quad \mathbf{0}_B^T]^T\}$.

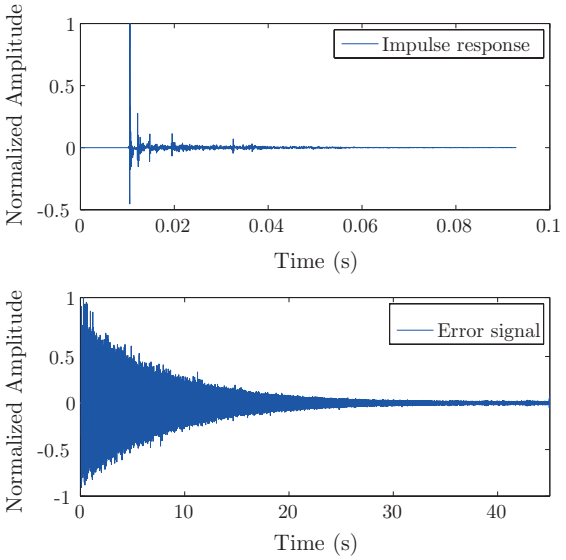


Figure 4.8. Estimated channel and residual error for $B = 128$ and $L = 4096$.

4.4 Results

This section analyzes the performance of the real-time implementation from three points of view:

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

The experiments have been carried out inside a listening room with dimensions of 9×4.5 meters. The main characteristics of the listening room have been described in section 2.3.

4.4.1 Algorithm behavior

Figure 4.8 shows the estimated channel and the residual error of the estimation. This results are obtained using a block size of $B = 128$ and a filter

length of $L = 4096$. The residual error shows the speed of convergence of the algorithm (note that it can be adjusted varying the step parameter μ). Theoretically, the algorithm has to identify an acoustic channel between a loudspeaker and a microphone. However, in practice, the channel that has to be identified is an electro-acoustic channel composed by the acoustic channel between a microphone and a loudspeaker, and the electric channel composed by the transducers and the audio card. In our prototype, the audio card introduces a delay that depends on the buffer size, and therefore, the estimated electro-acoustic channel depends on the size of B, specifically this delay is equal to $B + 22$ samples. Therefore, this implies that the result of the electro-acoustic estimated channel is the channel between the loudspeaker and microphone plus the delay introduced by the audio card, which makes the resulting channel variable with the size of B. Figure 4.9 shows the same channel estimated with different buffer sizes. In this figure, the delay introduced by the audio card is depicted with a dot line.

4.4.2 Implementation aspects

This subsection is devoted to analyze some computational aspects. To this end, the throughput and the latency time are studied. The latency time is defined as the delay from which the processing begins until a response is given. It is calculated as t_{proc}+t_{buff}. On the other hand, the throughput is defined as the number of input samples processed per second (throughput=B/Latency). Table 4.3 shows the latency time and throughput for different thread configurations and different B sizes. The best thread configurations are highlighted for each size of B. The table shows that the throughput does not vary significantly when B varies. This is because t_{proc} is much shorter than t_{buff}. Moreover, 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 processed per second.

4.4.3 Multichannel performance

A multichannel extension of this application means that several channels are identified at once. This multichannel extension could be formulated in two ways: in the first approach several single channel measurements can be performed in order to identify all the acoustic paths involved in a multichannel reproduction system, a second approach is based on measuring

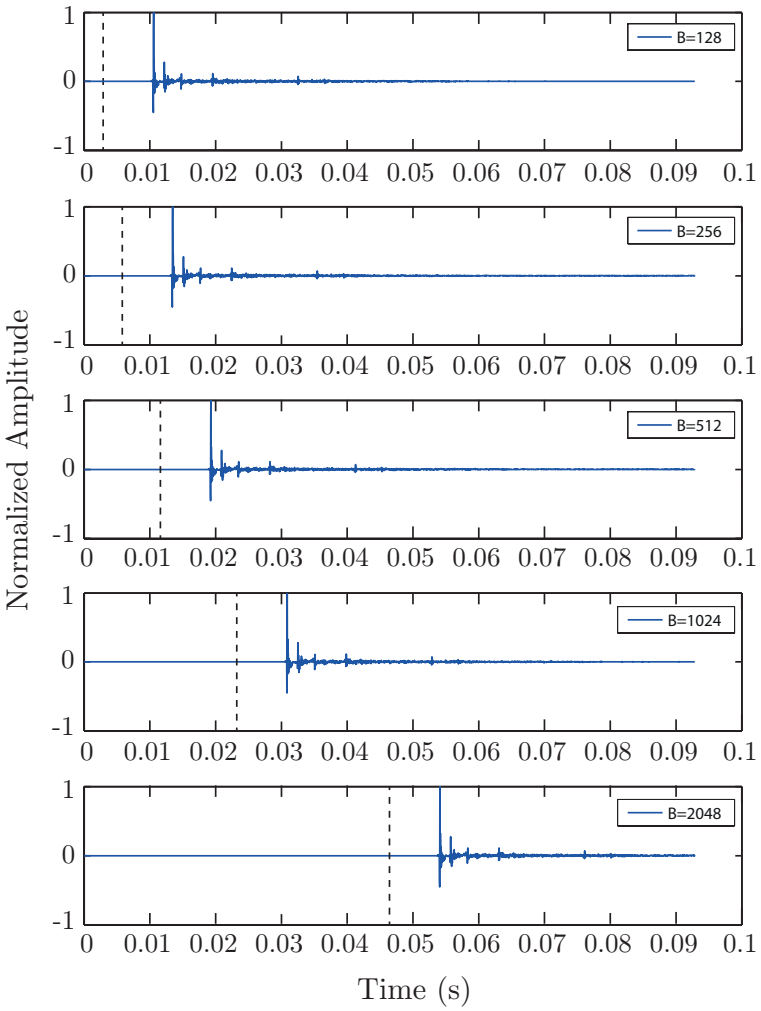


Figure 4.9. Estimations of the same channel for different input-data buffer size.

all the acoustic paths simultaneously. The second approach involves multiple microphones and loudspeakers, defining a MIMO system. In this case, the identification of all the channels is not trivial because each microphone receive data from all the loudspeakers, and therefore, some kind of signal multiplexing has to be performed in order to be able to deconvolve several impulse responses within the same measurement. Thus, two further

L size	**B size**	**t_{buff} (ms)**	**Threads per block**	**t_{proc} (ms)**	**Latency (ms)**	**Throughput (samples/s)**
4096	128	2.90	**128**	**0.44**	**3.34**	$\mathbf{3.83 \cdot 10^4}$
	256	5.80	128	0.54	6.34	$4.03 \cdot 10^4$
			256	**0.46**	**6.26**	$\mathbf{4.08 \cdot 10^4}$
	512	11.61	128	0.45	12.06	$4.24 \cdot 10^4$
			256	**0.43**	**12.04**	$\mathbf{4.52 \cdot 10^4}$
			512	0.46	12.07	$4.24 \cdot 10^4$
	1024	23.22	128	0.68	23.90	$4.28 \cdot 10^4$
			256	0.57	23.79	$4.30 \cdot 10^4$
			512	**0.52**	**23.74**	$\mathbf{4.31 \cdot 10^4}$
			1024	0.61	23.83	$4.29 \cdot 10^4$
	2048	46.44	128	0.60	47.04	$4.35 \cdot 10^4$
			256	**0.54**	**46.98**	$\mathbf{4.36 \cdot 10^4}$
			512	0.61	47.05	$4.35 \cdot 10^4$
			1024	0.71	47.15	$4.34 \cdot 10^4$

Table 4.3. Latency and throughput analysis obtained for different input-data buffer size.

possibilities can be considered: time or frequency multiplexing.

On the one hand, for the time multiplexing case, signals that have been used for years to measure single channel impulse responses, like the ones based on maximum length sequences (MLS) [84, 85] or frequency sweeps [86, 87], can be used in a very useful manner. On the other hand, the frequency multiplexing method was suggested by Potchinkov in [88] for the particular case of two channels: an audio channel and its corresponding unwanted cross-talk channel. In [89], both multiplexing techniques for multichannel identification are analyzed in further detail. However, the multiplexing techniques for the multichannel extension case are not the study of this thesis.

Although the extension of an adaptive single-channel identification system to a MIMO system is not trivial, the extension to a SIMO system where all the channels between one loudspeaker and multiple microphones are

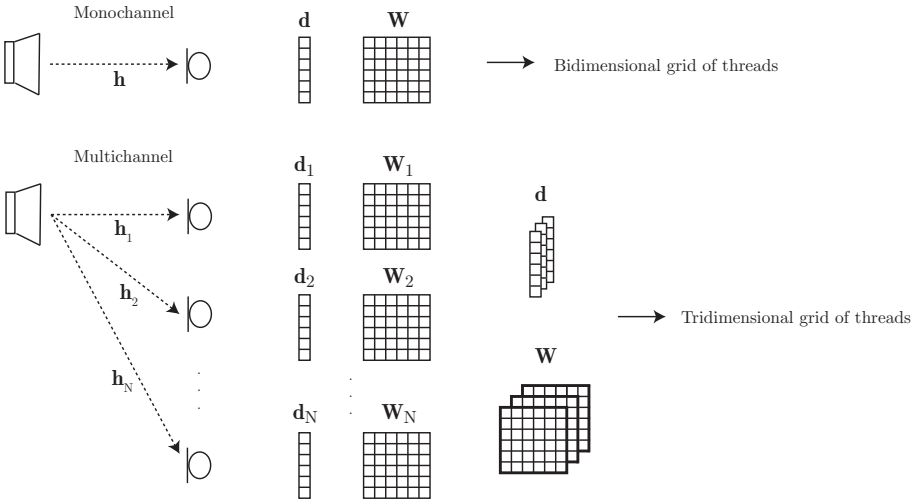


Figure 4.10. Extension of the GPU channel estimator to a multi-channel system.

identified at once is straightforward. Concretely, it can be easily obtained through a simple data arrangement. Figure 4.10 shows this multi-channel extension with 1 loudspeaker and N microphones. In this case, the same reference signal is used to calculate the paths between the loudspeaker and the different microphones. In the SIMO system, vector $\mathbf{d}$ becomes a three-dimensional vector where each channel occupies a plane of the matrix, being each plane a column vector. Consequently, the error vector becomes a matrix of the same dimensions. The same way, the system has a three-dimensional matrix $\mathbf{W}$ in which each plane corresponds to the filter of each channel. These three-dimensional data matrices can be handled by the GPU through CUDA, which let the user to configure three-dimensional grids of three-dimensional blocks of threads. Therefore, the same kernels of Table 4.2 can be used but launched with a three-dimensional configuration of the blocks and the grid. As an example, usigna SIMO system that identifies N channels, the dimensions of the blocks and the grid of Kernel 1 would be modified as follows: dimension of the blocks (x, y, z), dimensions of the grid (F/x, 2B/y, N/z). A similar modification would have to be done for the other kernels.

In order to see the parallel properties of the GPU, Table 4.4 compares the number of channels that the system is able to identify in real-time with

Input-data buffer size (B)	**Buffer filling time t_{buff} (ms)**	**Processing time t_{proc} (ms)**	**Nº of channels (estimation t_{buff}/t_{proc})**	**Nº of channels (real measurement)**
128	2.90	0.44	6	88
256	5.80	0.46	12	159
512	11.61	0.43	26	254
1024	23.22	0.52	44	362
2056	46.44	0.54	85	473

Table 4.4. Estimation of the number of channels that the system could identify for different input-data buffer sizes.

an estimate of this number of channels. It has been done for different block sizes. The estimate is calculated as follows: taking into account that the time t_{proc} is the time used to estimate a single channel, and the time t_{buff} is the maximum processing time to achieve the real-time condition, the ratio t_{buff}/t_{proc} is a direct estimate of the number of channels that, at least, the application could be able to identify. In other words, we consider that if the time spent to identify a single channel is 'X', the time spent to identify 2 channels would be '$2X$'. However, 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 involved, the processing time used by the system to identify X channels should be less than X-times the single-channel processing time. This hypothesis is confirmed by the results of the last column of Table 4.4, which shows that, in practice, the number of channels that can be identified without losing the real-time condition is much higher, reaching 473 channels for B=2056.

4.5 Conclusions

This chapter analyzes the viability of the use of GPUs for real-time adaptive applications. To this end, a simple adaptive application such as a single-channel identification system has been implemented using a GPU as the main processor. The identification system has been based on the LMS algorithm. Concretely, a partitioned BLMS algorithm in the frequency domain called FPBLMS algorithm has been implemented in order to efficiently ex-

ploit the SIMD GPU architecture. For this purpose, each CUDA kernel has been designed seeking the most efficient implementation avoiding memory copies in GPU. Moreover, some CUDA aspects as the number of threads per block, the distribution of threads within the blocks, the number of blocks in the grid and the distribution of the blocks in the grid have also been analyzed.

The results have shown a good performance in terms of the algorithm behavior. Moreover, the computing results have shown that it is important to carry out a precious analysis of the distribution of the threads inside the blocks and blocks inside the grid in order to efficiently configure each Kernel. On the other hand, a study of the implementation of a multichannel version of the application has been presented. In this sense, a SIMO system has been implemented and evaluated, demonstrating that depending on the input-data buffer size, the application can identify even hundreds of 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 the main processor, and that they are capable of managing multiple channels without overloading the CPU. In this regard, the following chapters are devoted to develop other multichannel adaptive applications.

Finally, the main results of this study were published in the Waves Journal [90] and opened a research line on GPU implementations of computationally complex adaptive algorithms for different adaptive applications such as multichannel room equalization or multichannel active noise control.

5 Equalization

5 Equalization

The second prototype that has been implemented using a GPU is an equalization system. Concretely, a multichannel Adaptive Equalization (AE) system. Therefore, this chapter 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. Moreover, some experimental results are presented to validate and computationally analyze the real-time performance of the AE GPU implementation. Finally, results show the usefulness of GPUs to develop versatile, scalable and low-cost multichannel AE systems.

5.1 Introduction

The equalization systems are used to compensate for transmission-channel impairments such as frequency-dependent phase and amplitude distortion, which can result in the interference of the transmitted signals with one another. Besides correcting for channel frequency-response anomalies, the equalizer can cancel the effects of multipath signal components. The equalization systems have become ubiquitous in many diverse applications including, voice, data, and video communication via various transmission

media. As an example, multichannel equalization could find 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 supplied to separate the listeners that are in the same listening space. Generally speaking, these systems intend to reproduce some desired signals at certain points of the listening space [91, 92]. The behavior of the acoustic system at a particular listening position (usually monitored by a microphone) is characterized by its impulse response. Thus, 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 a good approximation of the desired signal to be obtained at the microphone. In this chapter, the equalization is used to compensate the effects of a room. This approach is commonly known as room equalization.

The equalization systems employ two main techniques to formulate the filters: the fixed equalization and the adaptive equalization methods. In the fixed equalization methods, the equalizer typically calculates the coefficient of an inverse filter in order to compensate for the channel response. In these methods the equalization filters are computed once and usually in a previous stage to the rendering one. This task in multichannel scenarios is not straightforward, and efficient computational methods are needed. Moreover, the design of the inverse filter depends on the assumption of knowing the channel transfer function. But, in some digital communications applications, the channel transfer function is not known at enough level to incorporate filters that remove the channel effects. Furthermore, there are also non-stationary channels like in wireless communication, where the channels impulse response may vary with time. These restrictions lead to the use of the adaptive equalizers that iteratively compute the filters. An adaptive equalizer is an equalization filter that automatically adapts to time-varying properties of the communication channel. In the adaptive equalization context, although literature contains several interesting algorithms, the LMS algorithm in time or frequency domain, is the most widely used. However, most of these works only present results for a Single-Input Multiple-Output (SIMO) system, that is, an AE system that computes only one adaptive filter for all the microphones signals [93, 94]. It seems insufficient to perform equalization in multichannel scenarios with massive audience, where

the sound has to be equalized at each listener position. In these massive scenarios, Multiple-Input Multiple-Output (MIMO) systems [95] become essential to compensate room effects. Besides, the multichannel inversion in Wave Field Synthesis is another example of the use of MIMO systems in equalization [96].

On the other hand, with regards to the topic of this thesis, the AE MIMO systems seem suitable to be implemented using a GPU. This assertion is due to the fact that the AE MIMO systems involve a high number of long compensation adaptive filters, and, therefore, their implementation require a high computational capacity. Moreover, the processing of each channel could be done in parallel.

The previous chapter has set the bases of the real-time implementations of adaptive applications using a GPU as the main processor. To this end, a single-channel and a multichannel identification system based on the FPBLMS algorithm were presented and implemented on a GPU. Now, in this chapter, the FPBLMS algorithm is used with a filtered-x structure (FPBFxLMS) to implement a multichannel AE system on the GPU. The use of a filtered-x structure [97, 98] is necessary for the use of the LMS algorithm in adaptive equalization. Moreover, as explained in the previous chapter, the choice of the other aspects of the algorithm is conditioned by the efficient use of the parallel computation. To sum up, the block processing, the frequency domain and the partition of the filters are used due to hardware requirements of the audio card and to better exploit the parallel resources of the GPU. Finally, these reasons lead us to the use of the FPBFxLMS algorithm.

5.2 The FPBFxLMS algorithm applied to room equalization

In chapter 3, the FPBLMS algorithm was presented for a single-channel identification application. In this section, the FPBLMS algorithm is applied to a multichannel room equalization application. To this end, the FPBLMS algorithm is implemented with a filtered-x scheme, resulting in the FPBFxLMS algorithm. For simplicity, the block diagram of a single channel AE system is depicted in Figure 5.1. However, notation in Table 5.1 will be used to describe the algorithm for a generic multichannel AE sys-

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
$x_{[i]_n}$	nth sample of the ith reference signal
$\mathbf{x}_{[i]_n}$	$[x_{[i]_{Bn}} \quad x_{[i]_{Bn-1}} \quad \dots \quad x_{[i]_{Bn-B+1}}]^T$
$y_{[j]_n}$	nth sample of the jth actuator signal
$\mathbf{y}_{[j]_n}$	$[y_{[j]_{Bn}} \quad y_{[j]_{Bn-1}} \quad \dots \quad y_{[j]_{Bn-B+1}}]^T$
$e_{[k]_n}$	nth sample of the kth microphone signal
$\mathbf{e}_{[k]_n}$	$[e_{[k]_{Bn}} \quad e_{[k]_{Bn-1}} \quad \dots \quad e_{[k]_{Bn-B+1}}]^T$
$m_{[k]_n}$	nth sample of the kth microphone signal
$\mathbf{m}_{[k]_n}$	$[m_{[k]_{Bn}} \quad m_{[k]_{Bn-1}} \quad \dots \quad m_{[k]_{Bn-B+1}}]$
$d_{[k]_n}$	nth sample of the kth desired signal
$\mathbf{d}_{[k]_n}$	$[d_{[k]_{Bn}} \quad d_{[k]_{Bn-1}} \quad \dots \quad d_{[k]_{Bn-B+1}}]^T$
$\mathbf{s}_{[jk]}$	M-length estimation of the acoustic path that links the jth secondary source with the kth monitoring sensor
$\mathbf{S}^p_{[jk]}$	FFT of size $2B$ of the pth partition of the acoustic path $\mathbf{s}_{[jk]}$
$\mathbf{w}_{[ij]}$	Coefficients of the adaptive filter of length L that links the ith input signal with the jth secondary source
$\mathbf{W}^f_{[ij]_n}$	FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}_{[ij]}$ during the nth block iteration

Table 5.1. Notation of the description of the algorithms

tem with I input signals, J secondary sources and K monitoring sensors (I:J:K). The subscript between brackets is referred to the I:J:K channel configuration. Furthermore, the subscript and superscript of the following notation denotes block iteration and number of partition respectively. Finally, the estimated acoustic paths $\mathbf{s}_{[jk]}$ have been previously modeled by FIR filters.

The adaptive filter output is calculated in frequency domain at the nth

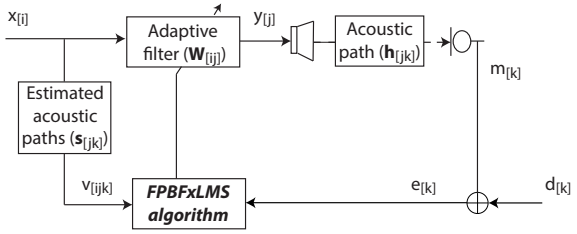


Figure 5.1. Block diagram of a single channel AE system based on the FPBFxLMS algorithm.

block iteration as

$$\mathbf{Y}_{[j]_n} = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}^f_{[ij]_n} \circ \mathbf{X}_{[i]_{n-f+1}}, \tag{5.1}$$

where $\mathbf{X}_{[i]_n} = \mathrm{FFT}_{2B}\{[\mathbf{x}^T_{[i]_{n-1}} \quad \mathbf{x}^T_{[i]_n}]^T\}$, and $\circ$ denotes the element-wise product of two vectors. The adaptive filter output at the nth block iteration $\mathbf{y}_{[j]_n}$ are the last B samples of $\mathrm{IFFT}_{2B}\{\mathbf{Y}_{[j]_n}\}$.

The algorithm error signals are calculated by subtracting the desired signals to the microphone signals. Therefore if the algorithm error signals tend to zero, the microphone signals converge to the desired signals. The kth error vector at the nth block iteration is calculated in time domain as:

$$\mathbf{e}_{[k]_n} = \mathbf{m}_{[k]_n} - \mathbf{d}_{[k]_n}. \tag{5.2}$$

Then, the kth error vector at the nth block iteration is padded with B zeros and transformed into the frequency domain by performing an FFT of size $2B$:

$$\mathbf{E}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{0}^T_B \quad \mathbf{e}^T_{[k]_n}]^T\}. \tag{5.3}$$

Moreover, the input signals are filtered through the estimated secondary paths. They are calculated at the nth iteration in frequency domain as

$$\mathbf{V}_{[ijk]_n} = \sum_{p=1}^{P} \mathbf{S}^p_{[jk]} \circ \mathbf{X}_{[i]_{n-p+1}}. \tag{5.4}$$

Let us define vector $\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n}$ as the DFT of the correlation between the input signals filtered through the estimated secondary paths, and the error signals. It is calculated for the fth partition at the nth iteration as:

$$\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n} = \mathbf{E}_{[k]_n} \circ \mathbf{V}^*_{[ijk]_{n-f+1}}, \tag{5.5}$$

for f=1,2,...,F. Finally, the update of the coefficients of the fth partition of the ijth adaptive filter is calculated in frequency domain as follows

$$\mathbf{W}^f_{[ij]n+1} = \mathbf{W}^f_{[ij]n} + \mu \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[{\boldsymbol{\phi}^f_{[ijk]n}}^T \quad \mathbf{0}_B^T]^T\}, \tag{5.6}$$

for f=1,2,...,F. Constant μ is the step-size parameter and vector $\boldsymbol{\phi}^f_{[ijk]_n}$ corresponds to the first B samples of the $2B$-IFFT of the corresponding partition $\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n}$

$$[\boldsymbol{\phi}^{f^T}_{[ijk]_n} \quad \bar{\boldsymbol{\phi}}^{f^T}_{[ijk]_n}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n}\}, \tag{5.7}$$

for f=1,2,...,F.

5.3 GPU implementation of the prototype

This section is devoted to explain the GPU implementation of the multichannel AE prototype. Details of the multichannel AE prototype have been shown in section 2.3. Figure 5.2 shows the GPU implementation of an iteration of the FPBFxLMS algorithm. The implementation makes use of five optimized GPU kernels that are described below. The goal is to design generic kernels that perform simple operations like element-wise multiplications or sums so that the kernels can be used in the GPU implementation of other multichannel applications. Therefore, these kernels will be used in the GPU implementation of the multichannel ANC systems described in chapter 6 and 7. Moreover, some GPU kernels are similar to those used in chapter 4. However, in this chapter, the kernels are designed and launched with three-dimensional blocks of threads and three-dimensional grid of blocks due to the multichannel nature of the application. Furthermore, the cuFFT library of NVIDIA has been used for carrying out simultaneously multiple FFTs. The GPU kernels are the following:

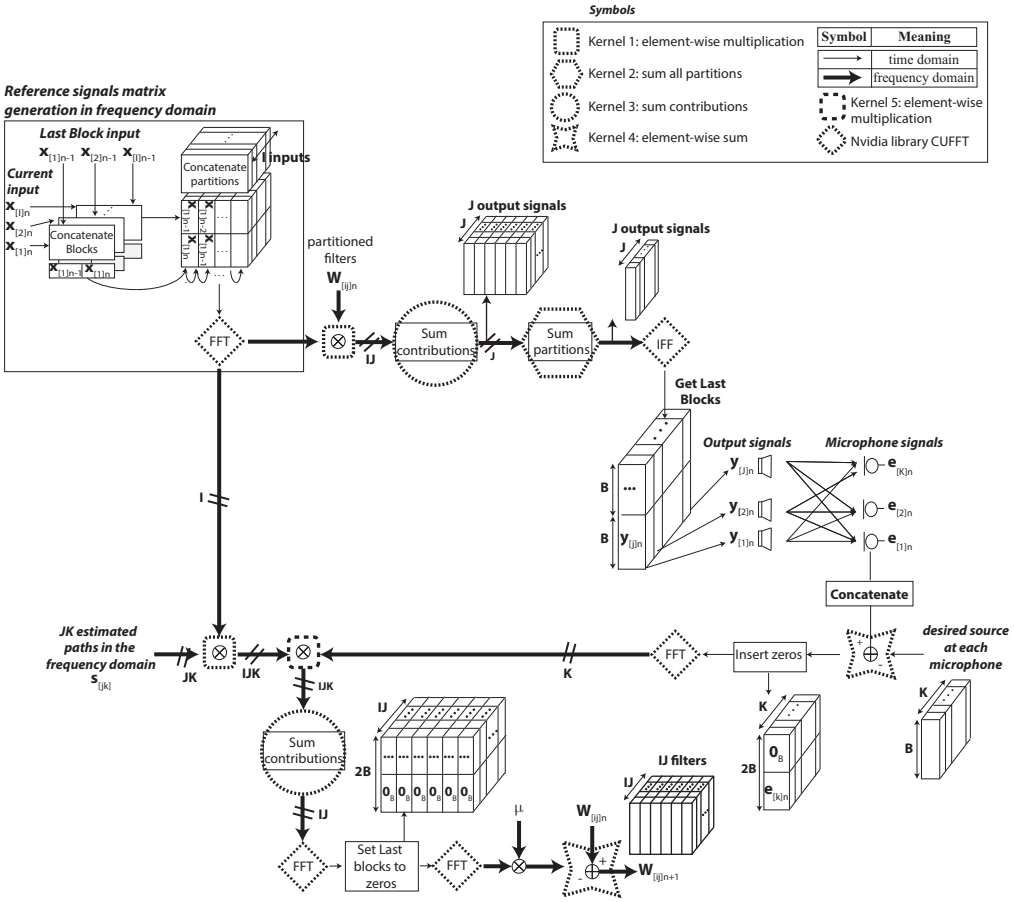


Figure 5.2. Block diagram of the GPU implementation of the FPBFxLMS algorithm for a multichannel room equalization application.

K1 Kernel 1 performs an element-wise multiplication of two matrices. This Kernel is similar to Kernel 1 of chapter 4 using the scheme 'B' depicted in Figure 5.3. However, due to the multichannel nature of this application, the matrices involved in the multiplication are three-dimensional. Each plane of the matrix corresponds to a different channel. The first dimension is the number of partitions, the second dimension is the size of B, and the third dimension is the number of channels. Therefore, this kernel performs the same computations than Kernel 1 of chapter 4, but applying it to all the channels of the system.

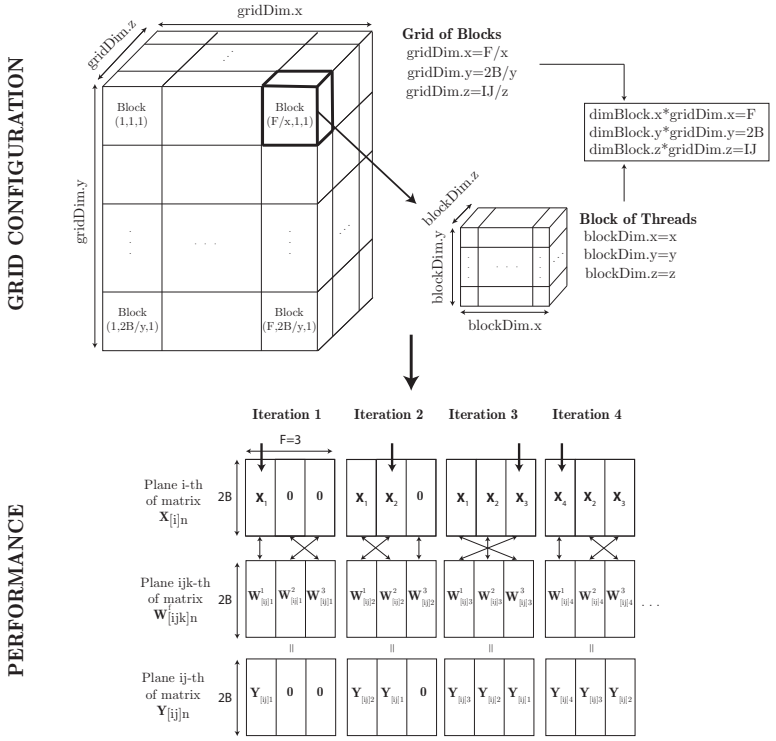


Figure 5.3. GPU kernel of an element-wise multiplication.

As the matrices involved in the multiplication are three-dimensional, this kernel launches a three-dimensional grid of three-dimensional blocks of threads. For example, for the particular case in which the input matrix $\mathbf{X}_{[i]}$ is element-wise multiplied with the adaptive filter matrix $\mathbf{W}_{[ij]}$ (see Figure 5.3), the blocks are dimensioned with (x,y,z)[1] threads, and the grid is dimensioned 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.

K2 Kernel 2 is launched to simultaneously reduce all the columns of each plane (channel) to a single one by computing the sum of the elements in each row. The number of columns that are reduced to a single one is F or P, depending on the number of partitions. This kernel is depicted

[1] x,y,z are selected according to the restrictions of the CUDA architecture of the device. In our case, Fermi device [11]

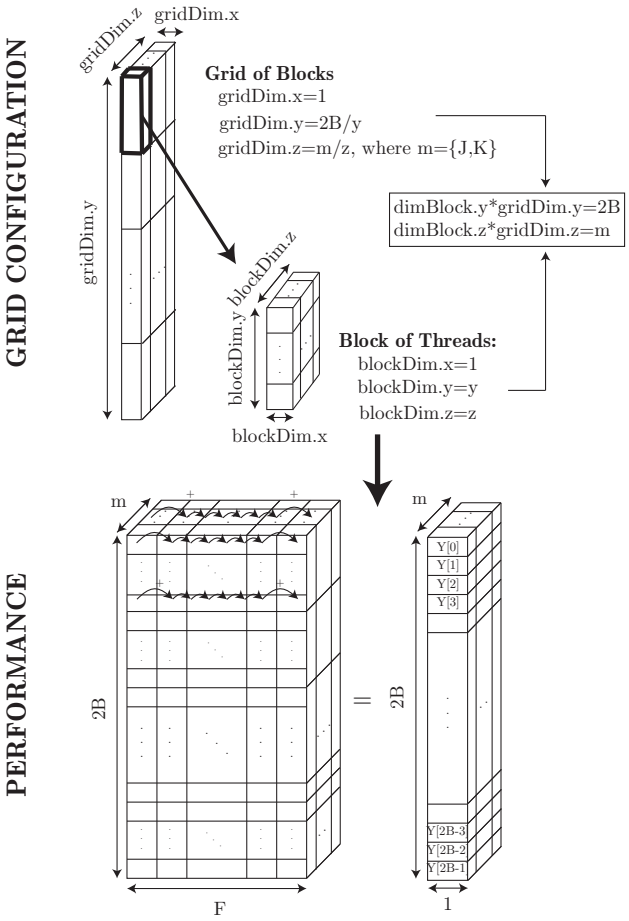


Figure 5.4. GPU kernel of an element-wise multiplication.

in Figure 5.4. The figure considers the case of reducing F columns to a single one. To this end, the kernel uses a three-dimensional grid of three-dimensional blocks, where the dimension of the blocks and grid are (1, y, z) and (1, 2B/y, m/z), respectively. Note that m is the number of planes (channels) of $2B \times F$ elements of the matrix involved. Therefore, the kernel launches $2B \cdot m$ threads in total, and each thread carries out F sums. The result is a $2B \times 1 \times m$ matrix where each element of the $2B$-dimension columns contain the reduction sum of each row.

K3 Kernel 3 is similar to kernel 2, but instead of performing the sum of

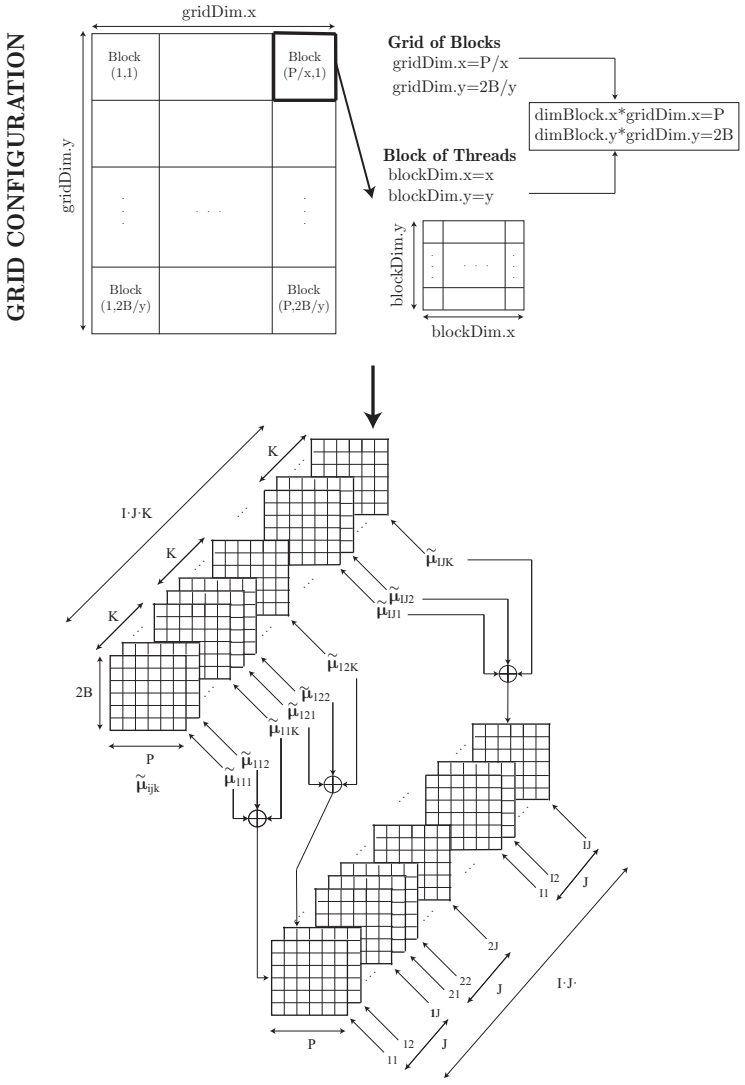


Figure 5.5. GPU kernel of an element-wise multiplication.

columns, it sums planes (channels). It performs a sum of m planes with the same sub-index. Figure 5.5 depicts an example of the performance of the kernel and the grid/block configuration. In this example, the matrix $\widetilde{\boldsymbol{\mu}}$ of IJK planes results in a matrix of IJ planes after performing this kernel. Therefore, a sum of K planes ($m = K$) with the same ij sub-

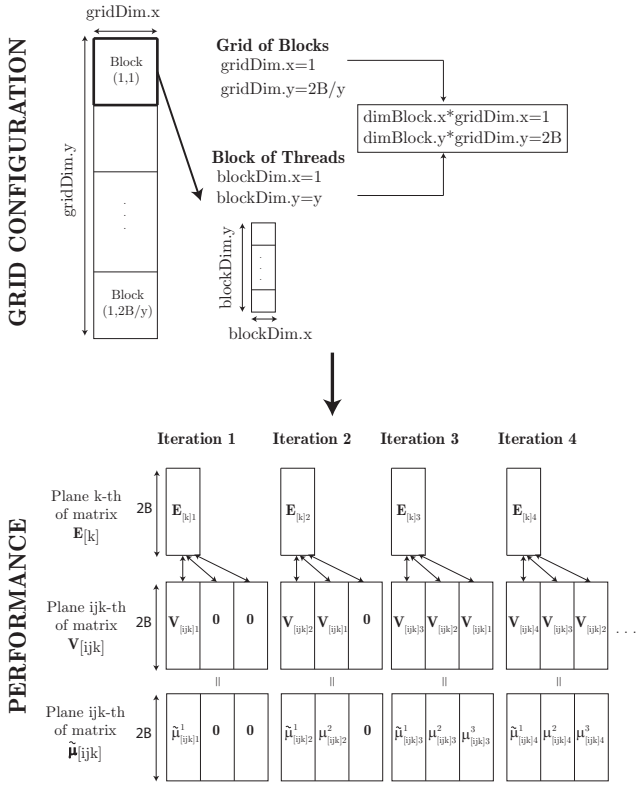


Figure 5.6. GPU kernel of an element-wise multiplication.

indexes is performed.

This Kernel launches $2B \cdot n$ threads, where $n = F, P$ depending on the number of partitions of the matrix involved. The dimensions of the grid are (n/x, 2B/y, 1), while the dimensions of the blocks are (x, y, 1). Each thread performs the sum of m elements.

K4 kernel 4 has the same thread configuration as kernel 1, but each thread performs a sum instead of a multiplication.

K5 This kernel performs an element-wise multiplication, and its operation is similar to the operation of kernel 1. However, the difference is that instead of multiply two matrices, this kernel performs the element-wise multiplication of a column vector ($\widehat{\mathbf{E}}_{[k]}$) and a matrix ($\mathbf{V}_{[ijk]}$). Therefore, as the kth plane of the error matrix ($\widehat{\mathbf{E}}_{[k]}$) used in the dot product is a

column vector instead of a matrix, each column vector of the ijkth plane of matrix $\mathbf{V}_{[ijk]}$ is element-wise multiplied by the same column vector $\widehat{\mathbf{E}}_{[k]}$. The performance of the kernel as well as the grid configuration is depicted in Figure 5.6.

5.4 Results

Some experiments are presented here to study the performance of the GPU-based adaptive equalization prototype. The experiments have been conducted in live using the prototype described in section 2.3. First, the algorithm behavior has been evaluated. Secondly, the computational limits of the GPU implementation have been studied.

5.4.1 Algorithm behavior

Regarding the algorithm behavior, the experiment consists on evaluating the system distance index (D) [99], 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 nth iteration at each sensor k as

$$D_{[k]_n} = 10 \log_{10} \left(\frac{\| \sum_{j=1}^{J} (\mathbf{w}_{[1j]_n} * \mathbf{s}_{[jk]}) - \mathbf{d}_{[k]} \|^2}{\| \mathbf{d}_{[k]} \|^2} \right), \tag{5.8}$$

being $\mathbf{d}_{[k]}$ the kth desired system vector which ideally corresponds to a delayed delta functions, and $\| \cdot \|_2$ the 2-norm.

Figure 5.7 shows the evolution of the parameter D at both microphones using a 1:2:2 configuration, a block size of $B = 512$ and adjusting the μ 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 power of the interferences increase, the SIR ratio decrease and the maximum μ parameter that assures stability also decrease [100]. As a result, the convergence speed is reduced. Results show that 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.

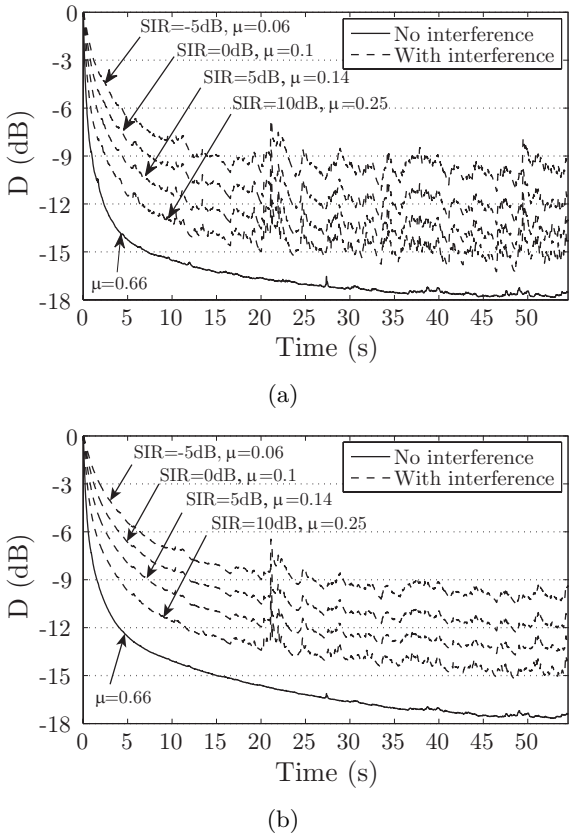


Figure 5.7. Dk_n evolution for the 1:2:2 configuration at microphone 1 (a) and microphone 2 (b).

5.4.2 Computing results

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 [101], 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 5.2 shows the maximum number of loudspeakers and microphones that the GPU implementation of the adaptive equalization system

	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

Table 5.2. Maximum I:J:K system for different sizes of B

can handle in real-time with one input signal (I=1), the same number of loudspeakers and microphones (J=K) and varying the size of B. 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 like in section 4.4.2 of the best distribution of threads in a block and blocks in a grid is necessary for each specific case in order to achieve a good performance. 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.

5.5 Conclusions

As it has been commented along this thesis, the main goal is to analyze the suitability of GPUs for real-time implementation of multichannel adaptive systems. In this sense, the previous chapter has demonstrated the feasibility of GPUs for real-time implementations of adaptive systems. Besides, the work presented in this chapter further confirms this conclusion. Here, an adaptive equalization systems based on the FxLMS algorithm has been implemented using a GPU. To fit the hardware/GPU requirements, the algorithm has been implemented in frequency domain, working with blocks of data and partitioning the adaptive filters. As a result, a prototype of multichannel adaptive equalization application has been successfully implemented on GPU using CUDA language and taking benefit of the parallelization of the multiple channels involved.

Results have shown a good performance of the adaptive equalization prototype even when there are interfering signals with low SIR levels. More-

over, 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 adaptive equalization system has been studied. It has been demonstrated that the GPU is a meaningful and versatile solution for massive multichannel adaptive equalization systems with even more than 1,000 channels processed in real time. Finally, 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.

Part of this work was presented in the International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014) [102] and is based on the algorithm presented in [103]. In [103] the FPBFxLMS algorithm was presented for a multichannel active noise control application, which is the main work of this thesis (see chapters 6, 7 and 8).

Active Noise Control - LMS algorithm 6

Active Noise Control - LMS algorithm 6

Following with the GPU implementation of adaptive applications, this chapter describes the implementation of a multichannel feedforward local Active Noise Control (ANC) system using a GPU as the main processor. As in the previous applications, this implementation is also based on the LMS algorithm. First, the GPU implementation based on the LMS algorithm is presented using two different filtering schemes: one based on the conventional filtered-x scheme, and the other based on the modified filtered-x scheme. Secondly, the details regarding the parallelization of the algorithms are given. Thirdly, the prototype has been tested in a listening room, so, finally, some experimental results are presented to compare the performance of both multichannel ANC GPU implementations.

6.1 Introduction

Active noise control [26, 25, 104] is a field that combines digital signal processing techniques with traditional acoustics. The ANC systems are based on the principle of destructive interference between a disturbance sound field called primary noise and a secondary sound field generated by a controlled secondary source called actuator. The target is to cancel, or at least

minimize, the primary noise signal. To cancel the primary noise, the ANC system commonly uses adaptive algorithms [17] to generate the secondary sound field from a reference signal that is correlated with the primary noise. As commented in section 2.1.4, the adaptive ANC systems can use two different approaches to obtain a reference of the noise signal. Both approaches are known as: feedforward or feedback systems. In this chapter, the feedforward approach will be used, and, therefore, a reference of the noise or undesired signal is obtained directly by a sensor. Moreover, the noise or the undesired signal is monitored by at a specific spatial point by a sensor that is called the error sensor. Therefore, cancellation is only carried out at that specific spatial point and also at close points around the error sensor (approximately $\lambda/10$ [105], where λ is the wavelength of the highest frequency of the undesired noise). The ANC systems can be extended to multichannel ANC systems by overlapping different controlled areas and setting multiple secondary sources [106].On the other hand, these multichannel systems require a high computational capacity. An even greater capacity is required when considering massive control systems with a high number of channels, with one channel being defined as each pair of error sensor - secondary source. Therefore, in practice, the computational cost is one of the main bottlenecks of the multichannel ANC systems.

As it has been commented during this thesis, in recent years, the number of scientific contributions and research projects related to the use of GPUs for signal processing applications has significantly increased. In section 2.2.1, numerous GPU implementations of audio application are cited. All these GPU implementations of audio application demonstrate that GPUs are a suitable platform for implementing multichannel audio applications where the processing of each channel could be done in parallel. Moreover, the results obtained from the previous chapters further demonstrate the viability of using a GPU as the main processor for audio applications based on adaptive filtering. Therefore, in contrast to traditional implementations that are based on conventional hardware devices such as Digital Signal Processors (DSPs) [107] and dedicated hardware, this chapter uses the GPU platform to implement a multichannel ANC prototype.

This chapter presents and compares two different GPU implementations of a multichannel ANC prototype. Both implementations are based on the FPBLMS algorithm, but using different filtering schemes. The choice

of this algorithm is motivated by the same reasons given in the previous chapters. These reasons can be summarized in allowing a fast implementation by efficiently exploiting the parallel resources of a GPU. Therefore, the FPBLMS algorithm used in chapter 4 for a channel identification application, is now implemented for an ANC system using two different filtering schemes: the conventional and the modified filtered-x scheme. The filtered-x structure is used to avoid the instability that causes the introduction of a secondary-path transfer function into a controller using the standard LMS algorithm [108]. To avoid this instability, the conventional filtered-x structure was suggested by Morgan in [109]. Moreover, the FxLMS algorithm was independently derived by Widrow [97] in the context of adaptive control and by Burgess [110] for ANC applications. In 2.1.5, a briefly analysis of the evolution over the years of the FxLMS algorithm, was given. There, it is outlined that the modified filtered-x structure [41] was presented to improve the convergence performance of the conventional filtered-x scheme. Throughout this thesis, the FPBLMS algorithm based on the conventional filtered-x scheme will be referred to as FPBFxLMS, whereas the FPBLMS algorithm based on the modified filtered-x scheme will be referred to as FPBMFxLMS.

As it has been commented above, the modified scheme provides a faster convergence rate. However, it is more demanding from a computational cost point of view. This chapter aims to demonstrate that this computational drawback can be alleviated by making use of the parallelism of the GPU computing. Therefore, the proposed implementation of the modified scheme can deal with the increase of computational burden. The rest of the chapter present and discuss about the advantages and disadvantages of the GPU implementation for both filtering schemes.

6.2 Description of the algorithms

This section focuses on illustrating the FPBFxLMS and FPBMFxLMS algorithms applied to an ANC system. The goal of the algorithms is to minimize the sum of the power of the error sensors. The block diagram of the multichannel ANC systems based on the two algorithms is depicted in Figure 6.1. A generic multichannel ANC system with I reference signals, J secondary sources, and K error sensors (I:J:K) has been considered. The

I	Number of reference signals (reference sensors)
J	Number of secondary sources (actuators)
K	Number of error signals (error 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 estimated secondary paths
P	M/B, number of partitions of the estimated secondary paths
$x_{[i]_n}$	nth sample of the ith reference signal
$\mathbf{x}_{[i]_n}$	$[x_{[i]_{Bn}} \quad x_{[i]_{Bn-1}} \quad \cdots \quad x_{[i]_{Bn-B+1}}]^T$
$y_{[j]_n}$	nth sample of the jth actuator signal
$\mathbf{y}_{[j]_n}$	$[y_{[j]_{Bn}} \quad y_{[j]_{Bn-1}} \quad \cdots \quad y_{[j]_{Bn-B+1}}]^T$
$e_{[k]_n}$	nth sample of the kth microphone signal
$\mathbf{e}_{[k]_n}$	$[e_{[k]_{Bn}} \quad e_{[k]_{Bn-1}} \quad \cdots \quad e_{[k]_{Bn-B+1}}]^T$
$\mathbf{s}_{[jk]}$	M-length estimation of the secondary path that links the jth secondary source with the kth error sensor
$\mathbf{S}^p_{[jk]}$	FFT of size $2B$ of the pth partition of the acoustic path $\mathbf{s}_{[jk]}$
$\mathbf{w}_{[ij]}$	Coefficients of the adaptive filter of length L that link the ith reference signal with the jth secondary source
$\mathbf{W}^f_{[ij]_n}$	FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}_{[ij]}$ at the nth block iteration

Table 6.1. Notation of the description of the algorithms

notation in Table 6.1 will be used to describe the algorithms. In this work, samples are processed by blocks of size B. The length of the adaptive filters is L, and, M is the length of the FIR filters that model the estimated secondary paths. As commented in the previous chapters, if L and M are greater than B, we have to split up both the adaptive filters and the estimated secondary paths into F and P partitions, respectively. Thus, the algorithm works simultaneously with all the partitions of size B. In line with the notation of the thesis, the subscript between brackets is referred to the (I:J:K) channel configuration, and the subscript and superscript of the notation denote block iteration and number of partition respectively.

It was commented in 2.1.5 that there are different approaches with regard to the modeling of the secondary path. One is based on the hypothesis that the secondary path model does not have to be accurate and it can be

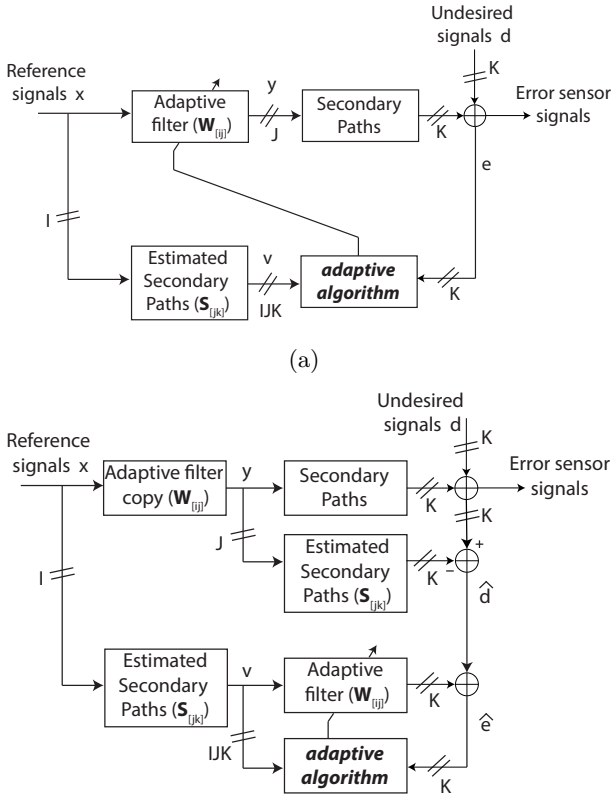


Figure 6.1. Block diagrams of the multichannel ANC system based on the (a) FPBFxLMS and (b) FPBMFxLMS algorithms.

also represented just by a delay (delayed-x LMS) [111]. This technique is used in applications with variable systems in which rapid reaction is also of utmost interest. However, since the application described in this chapter is set in a listening room with an almost invariant response, the secondary paths have been previously modeled with an accurate estimation by FIR filters called $\mathbf{s}_{[jk]}$.

6.2.1 The FPBFxLMS algorithm applied to active noise control

According to the notation of Table 6.1, the adaptive filter output is calculated as follows:

$$\mathbf{Y}_{[j]_n} = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}^f_{[ij]_n} \circ \mathbf{X}_{[i]_{n-f+1}}, \tag{6.1}$$

where $\mathbf{X}_{[i]_n} = \mathrm{FFT}_{2B}\{[\mathbf{x}^T_{[i]_{n-1}} \quad \mathbf{x}^T_{[i]_n}]^T\}$, $\mathbf{W}^f_{[ij]_n}$ is the FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}_{[ij]}$ at the nth block iteration, and $\circ$ denotes the element-wise product of two vectors. The valid samples of the adaptive filter output $\mathbf{y}_{[j]_n}$ are the last B samples of $\mathrm{IFFT}_{2B}\{\mathbf{Y}_{[j]_n}\}$.

The filter coefficients are updated in the frequency domain by calculating the DFT of the correlation between the reference signals that are filtered through the estimated secondary paths, and the error signals. To this end, the following operations are performed in each iteration

$$\mathbf{V}_{[ijk]_n} = \sum_{p=1}^{P} \mathbf{S}^p_{[jk]} \circ \mathbf{X}_{[i]_{n-p+1}}, \tag{6.2}$$

$$\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n} = \mathbf{E}_{[k]_n} \circ \mathbf{V}_{[ijk]_{n-f+1}}{}^*, \tag{6.3}$$

for $f = 1, 2, \ldots, F$. Vector $\mathbf{E}_{[k]_n}$ is defined as

$$\mathbf{E}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{0}^T_B \quad \mathbf{e}^T_{[k]_n}]^T\}. \tag{6.4}$$

The update of the coefficients of each partition of the ijth adaptive filters is calculated as follows

$$\mathbf{W}^f_{[ij]_{n+1}} = \mathbf{W}^f_{[ij]_n} - \mu \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^{f^T}_{[ijk]_n} \quad \mathbf{0}^T_B]^T\}, \tag{6.5}$$
$$f = 1, 2, \ldots, F.$$

where μ is the step-size parameter, and the vector $\boldsymbol{\phi}^f_{[ijk]_n}$ corresponds to the first B samples of the 2B-IFFT of the corresponding partition $\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n}$

$$[\boldsymbol{\phi}^{f^T}_{[ijk]_n} \quad \bar{\boldsymbol{\phi}}^{f^T}_{[ijk]_n}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n}\}, \tag{6.6}$$
$$f = 1, 2, \ldots, F.$$

6.2.2 The FPBMFxLMS algorithm applied to active noise control

This section focuses on describing the FPBMFxLMS algorithm. The adaptive filter output is calculated as in Eq. (6.1).

The output signal vectors in frequency domain $\mathbf{Y}_{[j]}$ are used to estimate the undesired signal vectors in the frequency domain, $\widehat{\mathbf{D}}_{[k]}$. To this end, the following operations are performed at the nth iteration:

$$\mathbf{YF}_{[k]_n} = \sum_{j=1}^{J} \sum_{p=1}^{P} \mathbf{Y}_{[j]_{n-p+1}} \circ \mathbf{S}_{[jk]}^{p}, \tag{6.7}$$

$$\widehat{\mathbf{D}}_{[k]_n} = \mathbf{E}_{[k]_n} - \mathbf{YF}_{[k]_n}, \tag{6.8}$$

where $\mathbf{E}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{e}_{[k]_{n-1}}^{T} \quad \mathbf{e}_{[k]_n}^{T}]^{T}\}$.

Moreover, the K estimated error vectors in the frequency domain at the nth iteration, $\hat{\mathbf{E}}_{[k]_n}$, are obtained as:

$$\mathbf{V}_{[ijk]_n} = \sum_{p=1}^{P} \mathbf{S}_{[jk]}^{p} \circ \mathbf{X}_{[i]_{n-p+1}}, \tag{6.9}$$

$$\boldsymbol{\xi}_{[k]_n} = \widehat{\mathbf{D}}_{[k]_n} + \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{f=1}^{F} \mathbf{V}_{[ijk]_{n-f+1}} \circ \mathbf{W}_{[ij]_n}^{f}, \tag{6.10}$$

and

$$\widehat{\mathbf{E}}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{0}_{B}^{T} \quad \boldsymbol{\varepsilon}_{[k]_n}^{T}]^{T}\}, \tag{6.11}$$

where vector $\boldsymbol{\varepsilon}_{[k]_n}$ is obtained from

$$[\bar{\boldsymbol{\varepsilon}}_{[k]_n}^{T} \quad \boldsymbol{\varepsilon}_{[k]_n}^{T}]^{T} = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\xi}}_{[k]_n}\} \tag{6.12}$$

Finally, the update rule of the frequency-domain filter coefficients is given by Eq. (6.5), where $\boldsymbol{\phi}_{[ijk]_n}^{f}$, and therefore $\widetilde{\boldsymbol{\mu}}_{[ijk]_n}^{f}$ (Eq. (6.3) and Eq. (6.6)) are calculated using the estimated error signal $\widehat{\mathbf{E}}_{[k]_n}$ instead of the error signal $\mathbf{E}_{[k]_n}$.

6.3 GPU implementation of the prototype

This section describes the main issues involved in the GPU implementation of the real-time multichannel ANC prototype. Details of the multichannel ANC prototype were given in section 2.3.

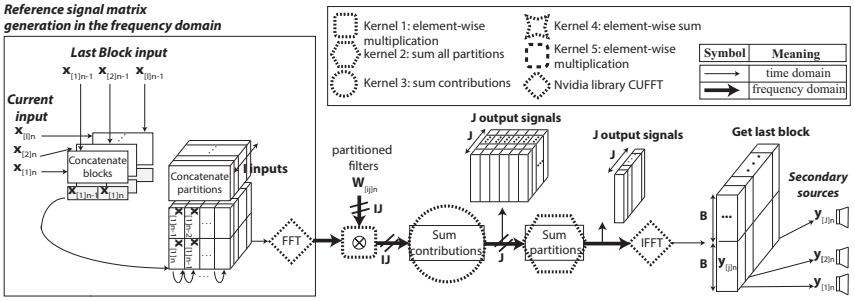
The GPU implementation consists of three steps: the output signal generation, the error signal calculation, and the update of the adaptive filters. Both algorithms generate the ANC outputs by filtering the reference signals through the adaptive filters, and they both update the adaptive filters using the error signals and the reference signals filtered through the estimated secondary paths. The main difference between the two algorithms is in the error signals that are used to update the filters. The FPBFxLMS algorithm uses the signals picked up by the microphones as the error signal, while the FPBMFxLMS algorithm computes an estimate of the error signals. These three steps are implemented as follows:

S1 *ANC output generation.* This step is common to both algorithms and aims to calculate the ANC output signals $y_{[j]}$. The operations of this step correspond to Eq. (6.1). The implementation and the CUDA kernels involved in it are shown in Figure 6.2(a).

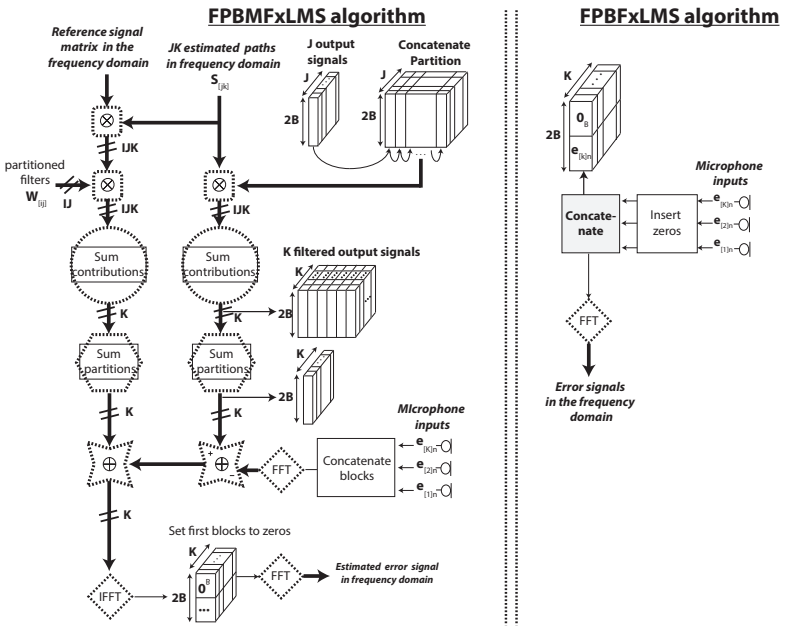
S2 *Error signal calculation.* This step is different for each implementation. While the conventional scheme uses Eq. (6.4) directly, the modified scheme calculates an estimate of the error signal (see Figure 6.2(b)). The corresponding description is shown in Eq. (6.7)-(6.12).

S3 *Filter updates.* The update of the adaptive filter coefficients involve the implementation of Eq. (6.3)-(6.6). The details regarding the different steps and kernels are illustrated by Figure 6.2(c).

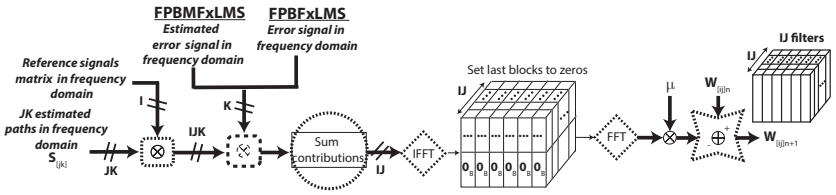
The GPU implementation depicted in Figure 6.2 makes use of five optimized kernels in order to achieve the most efficient performance of the algorithm. These kernels are the same as those presented in chapter 5 for a multichannel AE application. In this chapter, those kernels are reused for an ANC application. Moreover, the cuFFT library of NVIDIA was used to simultaneously carry out multiple one-dimensional FFTs.



(a) Block diagram of Step 1. Common for both algorithms.



(b) Block diagram of Step 2. Different for both algorithms



(c) Block diagram of Step 3. Common for both algorithms

Figure 6.2. Block diagram of the GPU implementation of the FPBFxLMS and the FPBMFxLMS algorithms.

Figure 6.3. Photograph of the ANC prototype using a 1:2:2 configuration.

6.4 Results

Several experiments were performed to validate the ANC systems and to compare both algorithms. The experiments were carried out using the prototype described in section 2.3. Some different configurations of the ANC system (I:J:K) were considered. As an example, Figure 6.3 shows a picture of the multichannel ANC system with the 1:2:2 configuration. Some other arrangements were also used to test the performance of other configurations, like 1:1:1 or 1:4:4. It should be highlighted that the prototype is capable of dealing with massive systems with high numbers of I, J and K, reaching more than 600 processed channels in the highest case.

The performance of the ANC system was evaluated from different points of view. It is worth mentioning that the implementation of the algorithm using the GPU does not affect either to improve or to impair the performance of the algorithm in terms of attenuation levels or speed of convergence. Therefore, section 6.4.1 is devoted to validating the ANC performance. For this purpose, the attenuation of the reference signal achieved by the ANC system at the error sensors was measured. A convergence per-

formance comparison between the FPBMFxLMS and the FPBFxLMS for different types of reference signals is shown in section 6.4.2. The computational complexity of the GPU implementation of both algorithms is detailed in section 6.4.3, and, finally, some computing results of the FPBMFxLMS algorithm are analyzed in section 6.4.4.

As commented in the previous chapters, some kind of study of the thread-block distribution of the kernels at each specific case is necessary in order to analyze the computing results. Similarly to that explained in 4.4.2 for the channel identification application, this study consists in a previous test of the processing delay of the algorithm when changing both the dimensions of the blocks of threads and the grid of blocks in order to find the fastest configuration for each different case.

6.4.1 Residual noise level

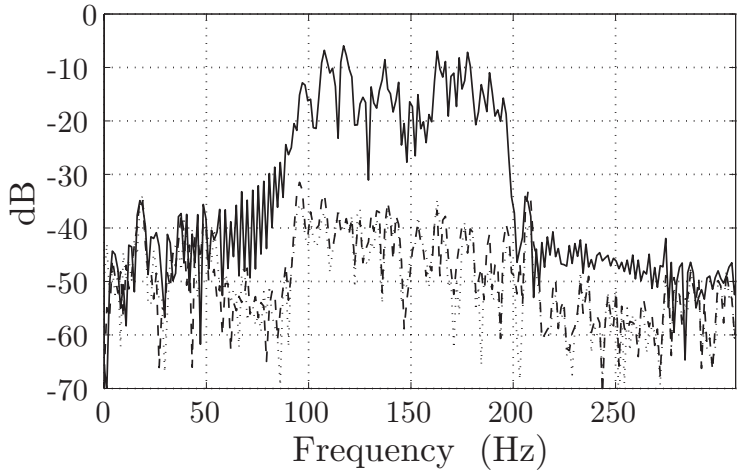
The 1:2:2 configuration of the ANC system was considered in this set of experiments. The following parameters were chosen: $B = 2048$, $L = M = 4096$, and two different types of reference signal. The reference signals were:

1. Band-limited white noise.
2. A periodic noise that emulates an engine signal composed of six harmonics between 100Hz and 200Hz with an effective fundamental frequency of 20 Hz.

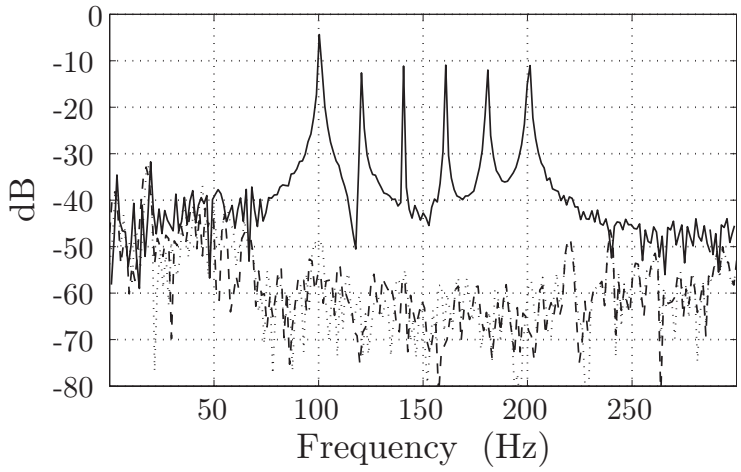
Figure 6.4 shows the power spectral density of the average signals measured at both error sensors by using the described algorithms. A similar performance is observed in both algorithms with noise reductions that depend on the type of the reference signal. If we consider the band pass filtered white noise, an attenuation between 20-30 dB is achieved depending on the frequency (see Figure 6.4(a)). Figure 6.4(b) illustrates the results for periodic noise. It can be easily observed that a reduction up to 45 dB is achieved at each harmonic.

6.4.2 Convergence performance

This section compares the convergence performance of the FPBMFxLMS and the FPBFxLMS algorithms. The same set of parameters used in the



(a)



(b)

Figure 6.4. Power spectral density of the average of the signals measured at the error sensors before (solid line) and after the ANC system operation by using the FPBFxLMS algorithm (dotted line) or the FPBMFxLMS algorithm (dashed line). The disturbance noise used in (a) is a band-limited white noise, and in (b) is a periodic tonal noise.

previous section was chosen. The learning curves of both algorithms were obtained using the following equation:

$$A_n = 10\log_{10}\left(\frac{Pe_n}{Pd_n}\right), \tag{6.13}$$

with

$$\begin{aligned} Pd_n &= \alpha Pd_{n-1} + (1-\alpha)pd_n, \\ Pe_n &= \alpha Pe_{n-1} + (1-\alpha)pe_n, \end{aligned} \tag{6.14}$$

and

$$\begin{aligned} pd_n &= \sum_{k=1}^{K} d^2_{[k]_n}, \\ pe_n &= \sum_{k=1}^{K} e^2_{[k]_n}. \end{aligned} \tag{6.15}$$

where K is the number of error sensors.

Figure 6.5 illustrates the performance of the algorithms. The highest step size μ that ensures the stability for each case was chosen. Using the filtered white noise as the disturbance signal, the step-size parameter was set to $\mu_c = 1.5 \cdot 10^{-5}$ and $\mu_m = 2.8 \cdot 10^{-5}$ for the conventional filtered-x scheme and the modified filtered-x scheme, respectively. For the periodic noise, the step-size parameter was set to $\mu_c = 3.5 \cdot 10^{-7}$ and $\mu_m = 7.3 \cdot 10^{-7}$.

As Figure 6.5 shows, the FPBMFxLMS algorithm provides faster convergence speed than the FPBFxLMS algorithm but similar steady-state behavior. Specifically, for the case of the filtered white noise, the modified scheme converges 1.2 seconds before the conventional scheme, which corresponds to a 15% reduction in the convergence time. A larger difference is observed when periodic noise is considered. The FPBMFxLMS algorithm converges almost 2.25 seconds before the FPBMFxLMS algorithm, which corresponds to a 47% reduction in the convergence time. Therefore, it can be concluded that the FPBMFxLMS algorithm significantly outperforms the FPBFxLMS algorithm in terms of convergence speed. Also, note that both algorithms converge faster for the periodic noise signal than for the filtered random noise signal.

Another important property of the adaptive algorithms is the stability limit. In the literature, there are some contributions made to study the

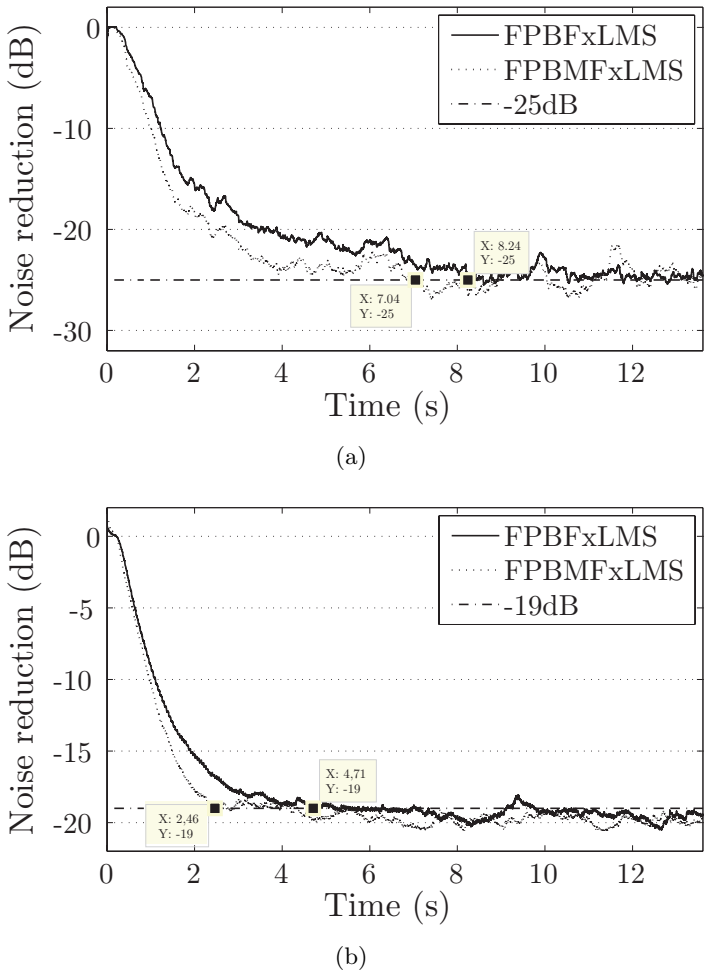


Figure 6.5. Learning curves of the FPBMFxLMS and the FPBFxLMS algorithms for the 1:2:2 ANC system and different reference signals. The disturbance noise used in (a) is a band-limited white noise, and in (b) is a periodic tonal noise.

convergence behavior of the Block Filtered-x LMS algorithm (BFxLMS). The maximum μ parameter that leads to the fastest convergence rate was derived in [112] and is given by

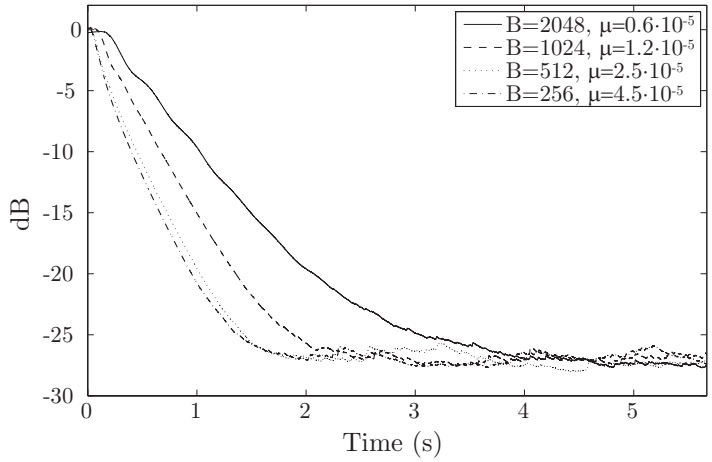
$$0 \leq \mu < \frac{1}{B\lambda_{max}} \tag{6.16}$$

where λ_{max} is the maximum eigenvalue of the filtered input signal autocorrelation matrix $\mathbf{R}_{vv}$ defined as $\mathbf{R}_{vv} = E[\mathbf{V}\mathbf{V}^T]$. Therefore, the convergence performance of the algorithm depends on the statistics of the input signal, the acoustic paths, and the block length B. For the same reference signal, the step-size parameter μ depends on B, so the maximum μ value increases by reducing the size of B, and, consequently, the convergence speed is improved by reducing B. However, the size of B is also limited by the real-time condition $t_{proc} < B/f_s$. Therefore, there is a minimum value of B for each configuration that assures the real-time condition and maximum convergence speed.

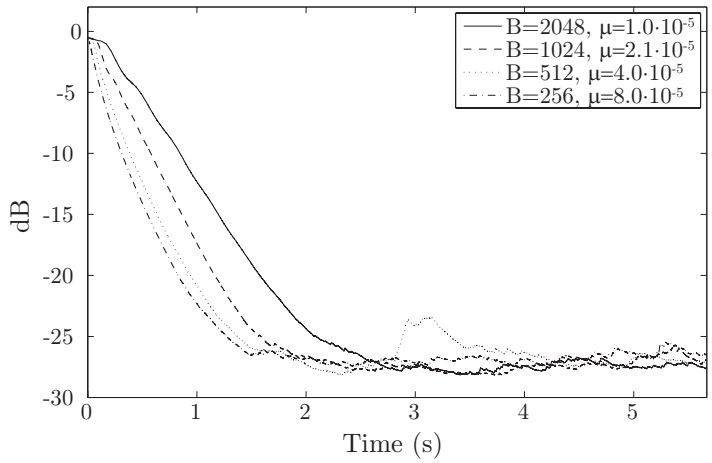
Figure 6.6 illustrates the convergence behavior of both algorithms using a when varying the size of B between 256 and 2048. Although these results are obtained when the disturbance signal is a single tone of 200 Hz, similar results can be obtained with other type of noise signals. As expected, it shows that the algorithms converge faster with a smaller block size, B. As these results show, the maximum μ is more or less doubled when B is halved. This fact can be explained from Eq. (6.16), where, for the same reference signal, the maximum μ is doubled by reducing the size of B by half. The difference in convergence time between $B = 2048$ and $B = 256$ is around 2.5 seconds in the conventional scheme and 1 second in the modified scheme (using $f_s = 44.1$ kHz). Although these results are obtained when considering a tone as the noise signal, the same dependence behavior between the speed of convergence and the block size could be expected for different kind of noise signals. Finally, Figure 6.6 shows that in order to achieve a certain convergence speed, the modified scheme can use a larger block size than the conventional scheme. This means the adaptive controller has more time for processing without violating the real-time condition, and, therefore, more channels can be handled while maintaining a given convergence speed.

6.4.3 Computational complexity

Table 6.2 compares the computing time and the computational complexity in terms of multiplications, additions, and FFTs per iteration of the GPU implementation of the two algorithms (FPBMFxLMS and FPBFxLMS) for



(a)



(b)

Figure 6.6. Learning curves of the FPBMFxLMS and FPBFxLMS algorithms for different B size and a single tone reference.

different configurations. Since we use a value of $M = L$, and $B = L/2$ (two partitions), the computational complexity only depends on L.

First, table 6.2 shows that the computational complexity of both algorithms increases significantly with the number of channels. In the modified

		I:J:K configuration		
		1:1:1	1:2:2	1:4:4
(1)	Multiplications	$8L$	$24L$	$80L$
	Additions	$4L$	$14L$	$52L$
	FFTs	8	14	26
	Time (ms)	0.55	0.78	1.58
(2)	Multiplications	$12L$	$40L$	$144L$
	Additions	$9L$	$32L$	$120L$
	FFTs	5	9	17
	Time (ms)	0.68	0.96	2.05
	M_m/M	1.5	1.6	1.8
	A_m/A	2.25	2.29	2.31
	F_m/F	1.25	1.29	1.31
	t_m/t	1.24	1.23	1.30

Table 6.2. Processing time and total number of multiplications, additions, and FFTs per iteration of the GPU implementation of (1) the FPBFxLMS algorithm and (2) the FPBMFxLMS algorithm for different ANC configurations.

scheme, when the ANC configuration changes from 1:1:1 to 1:4:4 (16 secondary paths) the number of multiplications increases by a factor of 12, the additions by a factor of 13.3, and the FFTs by a factor of 3.4 while the time delay increases only by a factor of 3. Taking into account that the complexity increases with the number of channels, we can conclude that the computational complexity is a bottleneck of massive multichannel ANC systems. Therefore, if the operations of each channel are properly parallelized, an implementation over GPU could be a viable and meaningful solution.

Table 6.2 also shows that the modified scheme exhibits higher computational complexity due to the estimation of the error signals (see Figure 6.2). Furthermore, we define the ratio M_m/M as the number of multiplications

of the FPBMFxLMS algorithm divided by the number of multiplications of the FPBFxLMS algorithm. The same ratio is defined for additions (A_m/A), processing time (t_m/t) and number of FFTs (F_m/F). It is shown that the ratios of multiplications, additions and number of FFTs are greater than the ratio of processing time. It means that the increase of computational complexity of the modified scheme can be overcome by the implementation of the algorithm on a GPU. This result further confirms that the GPU implementation is a good solution for multichannel ANC systems.

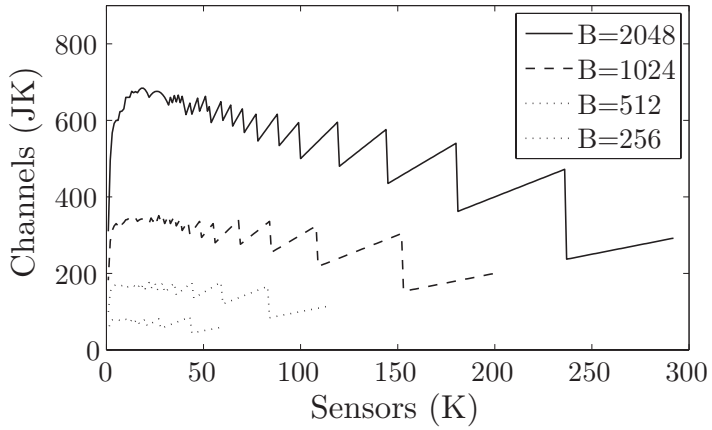
6.4.4 Prototype computing performance

It is well known that the zone of high attenuation achieved by an ANC system can be extended by adding more sensors and loudspeakers. However, as stated in the previous section, the computational cost can become extremely large.

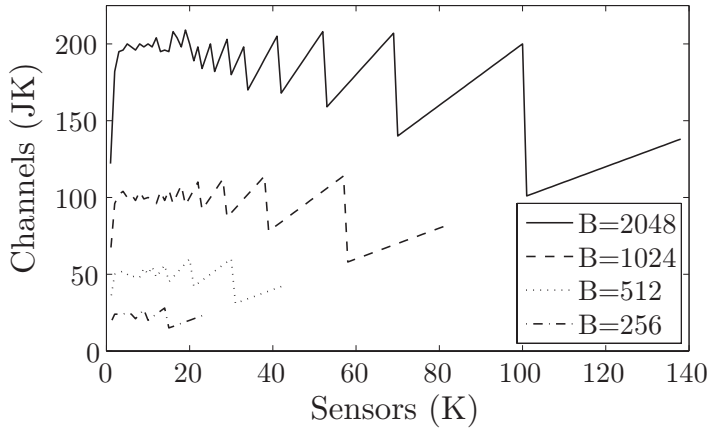
In this subsection, we will study the computational constraints of the multichannel ANC prototype using the FPBMFxLMS algorithm. For this purpose, Figure 6.7 shows the maximum number of channels that the ANC system can handle without violating the real-time condition for L=M=4096, different B values, and in two cases (Figure 6.7(a) one reference signal, and Figure 6.7(b) four reference signals). This maximum number of channels is calculated by fixing the number of error sensors and finding the maximum number of actuators (J) that the system can handle without violating the real-time condition. When the maximum J value for each value of K is found, the maximum number of physical channels that are processed for each value of K is $J \cdot K$. For example, in Figure 6.7(a), when $K = 180$, the maximum number of actuators that can be used without violating the real-time condition is $J = 3$; therefore, the maximum number of processed channels is 540. However, for $K = 181$, the maximum number of actuators is $J = 2$ because the real-time condition is violated with $J = 3$. Thus, the maximum number of processed channels is 362 ($J = 2$) when $K = 181$. For this reason, the shape of the curves jumps with the increase of error sensors.

The following considerations are highlighted in the simulation results depicted in Figure 6.7:

- The maximum number of actuators for each value of K is calculated by taking into account that it has to satisfy the real-time condition:



(a) I=1



(b) I=4

Figure 6.7. Maximum number of channels (JK) for each K value performing in real-time when (a) I=1 and (b) I=4 for different sizes of B and L=M=4096.

$t_{proc} < B/f_s$. Therefore, if B increases, there is more time for processing and thus more channels can be handled.

- Two systems with different I:J:K configurations could have the same number of physical channels but different computational costs. For example, both the 1:1:2 and 1:2:1 configurations have 2 physical chan-

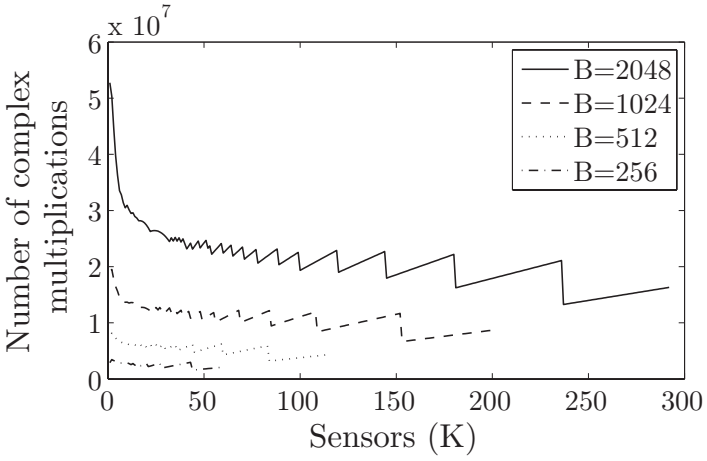
nels, but the second configuration has a higher computational cost because it has two adaptive filters instead of one. An increase in the number of adaptive filters involves many more operations than an increase in the number of error sensor signals, with IJ being the number of adaptive filters. Therefore, when K is low and J is high, the maximum number of channels is limited by the delay of processing the adaptive filters (see Figure 6.7 when K is low).

- When K increases, J has to decrease in order to satisfy the real-time condition, and, consequently, the number of adaptive filters decreases and the curves of the number of processed channels grow quickly reaching the maximums. The maximum of the curves is reached when neither K nor J is much greater than the other.

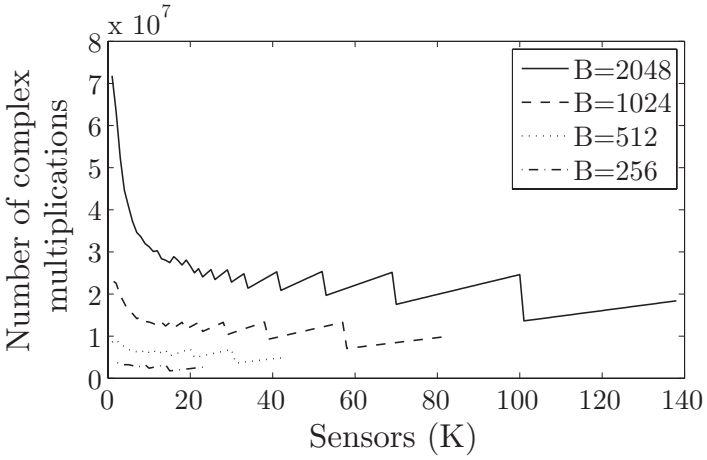
- On the other hand, when J is small and K is large, which means that the number of adaptive filters is low, the system is limited by the error signal handling (see Figure 6.2). Moreover, for low values of B or configurations with more than one reference signal, the decrease in processed channels with the increase of K is not so significant. This is because the system is able to process fewer sensors in real time than when $I = 1$ and $B = 2048$; therefore, since K is moderate, the decrease in the number of processed channels is also moderate.

- Two systems with the same JK configuration but a different number of reference signals have the same number of processed physical channels but different computational costs. For instance, if four reference signals instead of one are handled, the maximum number of processed channels are reduced because each channel is used four times instead of one (one time for each reference signal). On the other hand, due to the parallelization of the operations of each reference signal, even though the channels are used four times, the number of processed channels is not decreased by four.

Once the maximum number of processed channels has been analyzed for each value of K, the number of complex multiplications (CM) involved in both the products and the FFTs for the JK configurations derived in Figure 6.7 is depicted in Figure 6.8. It can be observed that the number of CM performed is not constant for the different configurations. This is because the GPU implementation is affected by the JK configuration.



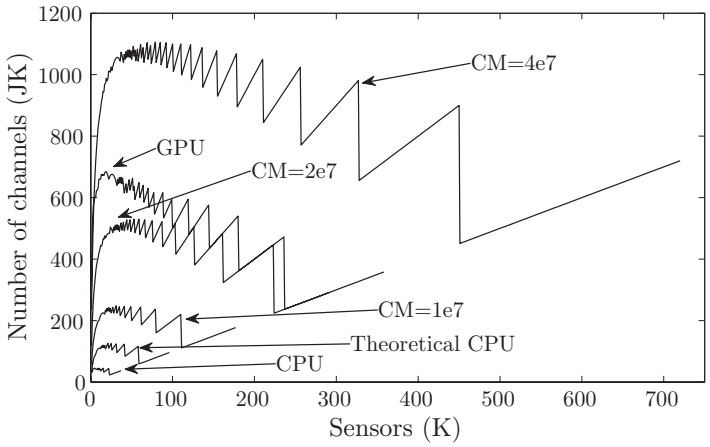
(a) I=1



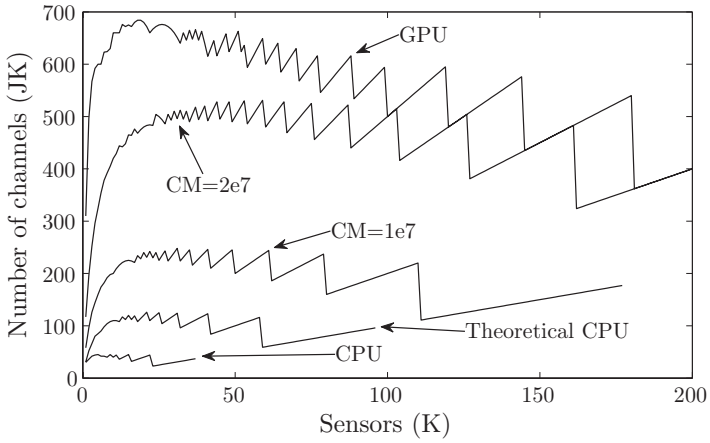
(b) I=4

Figure 6.8. Number of complex multiplications (CM) performed for the maximum JK configuration when (a) $I = 1$ and (b) $I = 4$ for different sizes of B and L=M=4096.

The most remarkable aspect of Figure 6.8 is that the JK configurations with more CM are those with low values of K and high values of J. As explained above, the number of adaptive filters depends on both the I and J variables; therefore, if J grows, more CM are performed because there



(a) I=1, B=2048



(b) Detail for the first 200 values of K

Figure 6.9. A comparison of the maximum number of processed channels for the GPU implementation, the CPU implementation, and a theoretical processing machine limited to perform $1 \cdot 10^7$, $2 \cdot 10^7$, or $4 \cdot 10^7$ complex multiplications (CM) per block of samples for $B = 2048$ and $I = 1$.

are more adaptive filters to cope with. Consequently, this is accentuated when $I = 4$.

The maximum number of channels that can be handled in real time by the GPU is shown in Figure 6.7. In order to compare the computational

capabilities of the GPU with other hardware platforms, an ANC system based on a CPU i7 was also implemented using one core and a sequential execution. The maximum number of channels allowed by the CPU implementation was theoretically and practically studied. Moreover, the GPU and CPU evaluation results were also compared with a theoretical processing machine that was limited to perform $1 \cdot 10^7$, $2 \cdot 10^7$, or $4 \cdot 10^7$ CM per buffering time. The results of this comparison are illustrated in Figure 7.7 for the case when $I = 1$ and $B = 2048$. In this case, the buffering time is $B/f_s = 2,048/44,100 = 0.0464$ seconds. For example, the curve labeled as 'CM=$1e7$' in Figure 6.9 represents the maximum number of channels for each value of K that a given machine would process if this machine is able to carry out $1 \cdot 10^7$ CM every 46.4 milliseconds. Finally, the theoretical maximum performance of our CPU was found by calculating the maximum number of CM that this CPU could handle in a real-time execution. A CM involves 4 floating point multiplications and 2 floating point additions. The floating point multiplications are performed by our CPU in 5 clock cycles, whereas the floating point additions are performed in 3 clock cycles (see annex 3 of [113]). Therefore, a CM involves 26 clock cycles. The CPU operates at 3.07GHz, which means that our CPU could make $3.07 \cdot 10^9/26$ CM per second and $(3.07 \cdot 10^9/26) \cdot 0.0464$ CM per buffering time. This is represented in the figure with the curve labeled as 'theoretical CPU'. Since our CPU also performs memory transactions and flow control instructions, the practical CPU implementation handles fewer channels than the 'theoretical CPU'. Furthermore, the theoretical processing machines that process $1 \cdot 10^7$, $2 \cdot 10^7$, or $4 \cdot 10^7$ CM per buffering time would also have to perform memory transactions and flow control instructions in addition to the complex multiplications. Therefore, in practice, these three curves would be lower.

Figure 6.9 illustrates that the GPU implementation outperforms the CPU implementation and shows the number of CM that a processing machine would have to carry out each buffering time to outperform the GPU implementation. Moreover, the maximum benefit of the GPU is obtained when low values of K and high values of J are used. Therefore, the GPU reaches $4 \cdot 10^7$ CM per buffering time. This can be explained by the fact that an increase in the value of K involves an increase in the time required to handle the error signals in the frequency domain.

6.5 Conclusions

In consonance with the previous chapters, and continuing with the analysis of the suitability of the GPUs for the real-time implementation of multichannel adaptive systems, this chapter has dealt with a GPU implementation of an ANC system based on the FxLMS algorithm. In this regard, two different schemes of the FxLMS algorithm has been compared: one that is based on the conventional filtered-x scheme (FPBFxLMS) and another that is based on the modified filtered-x scheme (FPBMFxLMS). In line with all the work presented in this thesis, the algorithms have been implemented in the frequency domain, working with blocks of data and partitioning the adaptive filters in order to exploit the parallel resources of the GPU.

The results of this chapter has shown that the FPBMFxLMS algorithm converges faster than the FPBFxLMS algorithm. However, the computational complexity of the FPBFxLMS increases significantly, especially for massive multichannel systems. Nevertheless, by taking advantage of the parallelization capabilities of the GPU, this increase in computational cost did not lead to a great increase in the processing delay. Therefore, the use of a GPU platform can help to overcome the disadvantage of the modified scheme in a real-time ANC system. As a conclusion, the same convergence behavior of the conventional scheme can be obtained with the modified scheme by using a larger block size. This provides more time for processing and, therefore, also provides the possibility to extend the zone of high attenuation by adding more microphones and loudspeakers.

On the other hand, the computing results have analyzed the computational limits of the ANC system. This study demonstrates that the GPU can provide slightly more than 600 processed channels in real time. Therefore, it is a meaningful and versatile solution for massive multichannel ANC systems that could provide a large area of high attenuation by using many sensors. Moreover, the computing results are related with the GPU used for the implementation. Therefore, as commented in the previous chapter, more channels could be processed by using a different audio card with a lower frequency sampling, by decimating, or by using newer GPUs with more computational capacity.

The main contributions of this chapter were published in the Transactions of Audio, Speech and Language Processing of IEEE [114]. Moreover, this work was based on the work presented in the following conferences:

International Conference of Sound and Vibration (ICSV) [115], Inter-noise [103] and the International Conference of the Audio Engineering Society (AES) [116].

Active Noise Control - NLMS-OCF algorithm

7

Active Noise Control - NLMS-OCF algorithm

7

The previous chapter introduces the use of the GPUs in multichannel active noise control systems. This chapter continues this work, and goes a step forward by presenting a GPU implementation of a multichannel feedforward local ANC system based on the NLMS-OCF algorithm. This algorithm increases the convergence performance of those presented in the previous chapter. However, the NLMS-OCF algorithm also increases the computational burden. Therefore, in order to minimize this computational drawback, the algorithm is implemented making use of the parallel resources of a GPU.

The results of the ANC prototype based on the NLMS-OCF algorithm are presented in this chapter, and then, they are compared with the results of the ANC systems presented in the previous chapter. In line with the results of this thesis, this chapter shows the usefulness of GPUS for developing versatile, scalable, and low-cost multichannel ANC systems.

7.1 Introduction

The previous chapter presented a GPU implementation of an ANC system based on the FxLMS algorithm. The normalized version of the FxLMS (FxNLMS) algorithm [4] and its multichannel version (see for example [106]), are among the most widely used adaptive filtering techniques applied to single or multiple channel adaptive noise cancelers for ANC applications. As it has been commented along this thesis, this is due to its computational simplicity and ease of implementation. On the other hand, one of the major drawbacks of the FxNLMS algorithm is the speed of convergence. In this sense, the Affine Projection Algorithm (APA)[43] was developed to address this problem. To this end, it uses affine subspace projections to speed the convergence of the LMS-type algorithms. From [43], some variants of the affine projection algorithm have been proposed for different authors, such as the Regularized APA (R-APA)[48], the Partial Rank Algorithm (PRA) [49], the Decorrelating Algorithm (DA) [44] or the NLMS with orthogonal Correction Factors (NLMS-OCF)[50] [45].

As opposed to the FxNLMS algorithm, which uses a single input signal vector to update the weights of the coefficients, the basic idea of the above mentioned algorithms is to update the weights of the adaptive filters on the basis of multiple input signal vectors. These algorithms are commonly referred to in literature as the APA family [42]. In the existing literature, the algorithms of the APA family have traditionally been implemented in the time domain for single-channel system identification or echo cancellation problems. However, in this chapter, we use one of them for a multichannel ANC system.

Among the APA family algorithms, we have focused on the NLMS-OCF algorithm because we find its practical implementation easier than the APA implementation and, as considered in [45], the NLMS-OCF algorithm, which is a generalization of the APA, allows other than unit delay between input vectors. With regards to its implementation in an ANC system, the unavailability of the undesired signal in ANC systems, justifies the use of the modified filtered-x scheme. Therefore, the modified filtered-x NLMS-OCF (M-OCF) [1] algorithm is derived from the OCF algorithm and is used

[1]For the sake of simplicity, we will refer throughout this chapter to the filtered-x NLMS-OCF algorithm as the OCF algorithm and to the modified filtered-x NLMS-OCF algorithm as the M-OCF algorithm.

in a multichannel ANC system.

Since the M-OCF uses multiple input vector signals to update the weights, the algorithm is highly demanding from a computational point of view, particularly in multichannel scenarios. This computational drawback limits its implementation in multichannel real-time systems. In line with the theme of the thesis, the GPU implementation of the M-OCF algorithm has been used in order to ameliorate this computational drawback. For this purpose, the same way that it was done in the others implementations, the M-OCF algorithm has been adjusted to successfully meet the hardware requirements, and, therefore, a frequency-domain partitioned block-based M-OCF algorithm has been proposed. Throughout the chapter, it will be called the Frequency Partitioned Block Modified Filtered-X Normalized Least Mean Square algorithm with Orthogonal Correction Factors (FPM-OCF). As we mentioned above, the use of the NLMS-OCF algorithms instead of the APA was motivated in part by the ease of implementation since it can be easily changed to the frequency domain while the implementation of the APA in the frequency domain (which is required in most of the practical implementations using non dedicated hardware) is not straightforward.

The main motivation of the work presented in this chapter is twofold: on the one hand, to find an algorithm that improves the convergence rate of the NLMS-type algorithms, and, on the other, to obtain an efficient version of that algorithm to reduce the computational requirements. For these reasons, we have considered the NLMS-OCF algorithm, in which both the convergence rate and the computational complexity depend on the number of correction factors (R) that are used. Therefore, this leads to an adjustable and configurable algorithm with regard to the complexity and the convergence rate. Accordingly, the main contributions of this work can be summarized in the following points: first, we present the NLMS-OCF algorithm with an embedded modified filtered-x structure (M-OCF) in order to make it suitable for multichannel ANC systems. Second, the M-OCF has been adjusted to its frequency partitioned block-based version (FPM-OCF) in order to meet the hardware requirements of a real-time implementation. Finally, a parallel implementation using a GPU hardware platform is presented to increase the applicability and versatility of the multichannel system.

7.2 Description of the algorithms

This section focuses on describing the M-OCF algorithm for a multichannel ANC system that is derived from the NLMS-OCF algorithm introduced in [50] for a system identification application. In [50], the NLMS-OCF algorithm is derived from the NLMS algorithm. The NLMS algorithm adapts the filter weights so that the error signal e is minimized in the mean-square sense. When both the input signal x and desired output signal d are stationary, the algorithm converges to a filter weights which, on average, are equal to the Wiener-Hopf solution. In other words, the NLMS is an iterative algorithm that implements Wiener filters without explicitly solve the Wiener-Hopf equation. Finally, the NLMS algorithm is a stochastic implementation of the steepest-descent algorithm, and the adaptation of the filters is done replacing the cost function $\xi = \mathrm{E}[e_n^2]$ by its instantaneous error [16].

On the other hand, the NLMS-OCF algorithm solves a constrained minimization problem. As the NLMS-OCF algorithm uses R input vectors for the filter weights update, it iteratively solves the constrained minimization problem until each of the R a posteriori error is forced to zero. In summary, it finds the filter weights $\mathbf{w}_n^r$ such that $\|\mathbf{w}_n^r - \mathbf{w}_n\|$ is minimized subject to $d_{n-r} - \mathbf{x}_{n-r}^T \mathbf{w}_n^r = 0$ for $r = 0, 1, ..., R$ (see appendix of [45]).

In a system identification application, where the NLMS-OCF algorithm was introduced, the error signal is the error in estimating the so-called desired signal that is picked up by the microphones. In contrast, in an ANC system, the microphones pick up a mixture of the disturbance or undesired signals and the signals that are used to cancel the noises. Therefore, since the desired signals are not available and they are needed to calculate the error signals [117], we have to use a modified filtering scheme to estimate them. Therefore, this section begins with a reminder of the modified filtered-x NLMS algorithm, and, then, introduces the modified filtered-x structure in the NLMS-OCF algorithm (M-OCF). Finally, the FPM-OCF algorithm is derived from the M-OCF algorithm in order to adjust the algorithm to meet the hardware requirements and to seek the best parallel implementation.

For this purpose, a generic multichannel ANC system with I reference signals, J secondary sources, and K error sensors (I:J:K) has been considered. Moreover, we define L as the length of the adaptive filters and M

I	Number of reference signals (reference sensors)
J	Number of secondary sources (actuators)
K	Number of error signals (error 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 estimated secondary paths
P	M/B, number of partitions of the estimated secondary paths
R	Number of successive input vectors used in the weight update of the NLMS-OCF algorithm
D	Number of iterations between the successive input vectors used in the weight update of the NLMS-OCF algorithm
$x_{[i]_n}$	ith reference signal at sample n
$\mathbf{x}_{[i]_n}$	$[x_{[i]_{Bn}} \quad x_{[i]_{Bn-1}} \quad \cdots \quad x_{[i]_{Bn-B+1}}]^T$
$y_{[j]_n}$	jth actuator signal at sample q
$\mathbf{y}_{[j]_n}$	$[y_{[j]_{Bn}} \quad y_{[j]_{Bn-1}} \quad \cdots \quad y_{[j]_{Bn-B+1}}]^T$
$e_{[k]_q}$	kth microphone signal at sample q
$\mathbf{e}_{[k]_n}$	$[e_{[k]_{Bn}} \quad e_{[k]_{Bn-1}} \quad \cdots \quad e_{[k]_{Bn-B+1}}]^T$
$\mathbf{s}_{[jk]}$	M-length estimation of the secondary path that links the jth secondary source with the kth error sensor
$\mathbf{S}^p_{[jk]}$	FFT of size $2B$ of the pth partition of the acoustic path $\mathbf{s}_{[jk]}$
$\mathbf{w}_{[ij]_n}$	Coefficients of the adaptive filter of length L that links the ith reference signal with the jth secondary source at at the nth block iteration
$\mathbf{W}^f_{[ij]_n}$	FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter that links the ith reference signal with the jth secondary source at the nth block iteration

Table 7.1. Notation for the multichannel ANC algorithms considered

as the length of the FIR filters that model the estimated secondary paths. The notation in Table 7.1 will be used to describe the algorithms. As happened in the previous chapter, the secondary paths have been previously modeled by FIR filters and considering an accurate estimation.

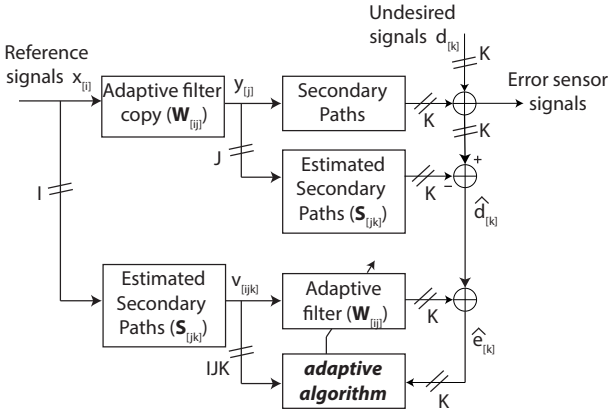


Figure 7.1. Scheme of the multichannel ANC system.

7.2.1 The Modified Filtered-x NLMS algorithm (MFxNLMS)

As it has been mentioned above, this subsection outlines the multichannel version of the MFxNLMS [41] algorithm in time domain and working with sample-by-sample processing. The block diagram of a multichannel ANC system based on the MFxNLMS algorithm is depicted in Figure 7.1. The MFxNLMS algorithm is constituted by the following steps:

1. Calculation of the output signals.

   The nth sample of the jth output signal is calculated as

   $$y_{[j]n} = \sum_{i=1}^{I} \mathbf{w}_{[ij]_n}^T \mathbf{x}_{[i]_n}, \tag{7.1}$$

   where $\mathbf{x}_{[i]_n} = [x_{[i]_n} \quad x_{[i]_{n-1}} \quad x_{[i]_{n-2}} \quad \dots \quad x_{[i]_{n-L+1}}]^T$ is the ith input signal vector with the last L samples from sample n.

2. Calculation of the estimated undesired signals:

   $$\hat{d}_{[k]_n} = e_{[k]_n} - yf_{[k]_n}, \tag{7.2}$$

   where $e_{[k]_n}$ is the nth sample of the kth microphone signal, and sample $yf_{[k]_n}$ is defined as

$$yf_{[k]n} = \sum_{j=1}^{J} \mathbf{s}_{[jk]}^T \mathbf{y}_{[j]_n}, \tag{7.3}$$

where $\mathbf{y}_{[j]_n} = [y_{[j]_n} \quad y_{[j]_{n-1}} \quad y_{[j]_{n-2}} \quad \cdots \quad y_{[j]_{n-M+1}}]^T$ is the jth output signal vector with the last M samples from sample n.

3. Filter updates:

$$\mathbf{w}_{[ij]_{n+1}} = \mathbf{w}_{[ij]_n} - \sum_{k=1}^{K} \hat{\mu}_{[ijk]_n} \mathbf{v}_{[ijk]_n}, \tag{7.4}$$

where the step-size $\hat{\mu}_{[ijk]_n}$ is calculated as

$$\hat{\mu}_{[ijk]_n} = \mu \frac{\hat{e}_{[k]_n}}{\|\mathbf{v}_{[ijk]_n}\|^2}, \tag{7.5}$$

with μ being the step size parameter and $\hat{e}_{[k]_n}$ being the error signal when estimating the nth sample of the kth undesired signal, $\hat{d}_{[k]_n}$, given by

$$\hat{e}_{[k]_n} = \hat{d}_{[k]_n} + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}_{[ij]_n}^T \mathbf{v}_{[ijk]_n}, \tag{7.6}$$

where $\mathbf{v}_{[ijk]_n} = [v_{[ijk]_n} \quad v_{[ijk]_{n-1}} \quad v_{[ijk]_{n-2}} \quad \cdots \quad v_{[ijk]_{n-L+1}}]^T$ is a vector with the last L samples of the ith input signal filtered through the jkth secondary path, which is calculated as

$$v_{[ijk]_n} = \mathbf{s}_{[jk]}^T \mathbf{x}_{[i]_n}. \tag{7.7}$$

It is important to note the difference between vector $\mathbf{x}_{[i]_n}$ in Eq. (7.1) and Eq. (7.7). In Eq. (7.7), the vector $\mathbf{x}_{[i]_n}$ is filtered through an M-length filter ($\mathbf{s}_{[jk]}$), instead of an L-length filter ($\mathbf{w}_{[ij]_n}$). Therefore, in this case, vector $\mathbf{x}_{[i]_n}$ is defined as the ith input signal vector with the last M samples from sample n.

7.2.2 The Modified Filtered-x NLMS algorithm with Orthogonal Correction Factors (M-OCF)

This section introduces the modified filtered-x structure in the NLMS-OCF algorithm, resulting in what we have called the M-OCF algorithm. Since the M-OCF algorithm updates the weights of the adaptive filters on the basis of multiple input vectors, we define R as such number of input vectors, which are also called "correction factors". Moreover, the parameter D is defined as the number of iterations between the successive input vectors used in the weights update. The best improvement in convergence rate occurs if the successive input vectors are orthogonal. When the successive input vectors are not orthogonal, the authors in [50] proposed generating orthogonal directions and moving along those directions.

Here we introduce the M-OCF algorithm for an ANC system derived from the MFxNLMS algorithm. It is done similarly to the way the NLMS-OCF was derived from the NLMS in [50] in a system identification application. The equations for calculating the output signals and the estimated undesired signals are the same as in the MFxNLMS algorithm. However, the weight update equation in Eq. (7.4) is modified as [45]

$$\begin{aligned}\mathbf{w}_{[ij]_{n+1}} &= \mathbf{w}_{[ij]_n} - \\ &- \sum_{k=1}^{K} \hat{\mu}_{0_{[ijk]_n}} \mathbf{v}_{0_{[ijk]_n}} - \sum_{k=1}^{K} \hat{\mu}_{1_{[ijk]_n}} \mathbf{v}_{1_{[ijk]_n}} - \ldots - \sum_{k=1}^{K} \hat{\mu}_{R_{[ijk]_n}} \mathbf{v}_{R_{[ijk]_n}} \\ &= \mathbf{w}_{[ij]_n} - \sum_{r=0}^{R} \sum_{k=1}^{K} \hat{\mu}_{r_{[ijk]_n}} \mathbf{v}_{r_{[ijk]_n}},\end{aligned} \tag{7.8}$$

where R is the number of input vectors that are used in the tap weight adaptation, and $\mathbf{v}_{0_{[ijk]_n}}$, $\mathbf{v}_{1_{[ijk]_n}}$,..., $\mathbf{v}_{R_{[ijk]_n}}$ are orthogonal to each other. As before, $\mathbf{v}_{0_{[ijk]_n}}$ is the filtered input vector at the nth instant, and $\hat{\mu}_{0_{[ijk]_n}}$ is chosen as in the MFxNLMS

$$\hat{\mu}_{0_{[ijk]_n}} = \mu \frac{\hat{e}_{[k]_n}}{\|\mathbf{v}_{0_{[ijk]_n}}\|^2}. \tag{7.9}$$

Let us define the vector $\mathbf{v}_{r_{[ijk]_n}}$ obtained from $\mathbf{v}_{[ijk]_{n-rD}}$ but constrained to be orthogonal to $\mathbf{v}_{[ijk]_n}$, $\mathbf{v}_{[ijk]_{n-D}}$, $\mathbf{v}_{[ijk]_{n-2D}}$, ..., $\mathbf{v}_{[ijk]_{n-(r-1)D}}$, where D

is the distance between the input vectors. For example, considering $D = 1$, $\mathbf{v}3_{[ijk]_n}$ is the vector that is obtained from $\mathbf{v}_{[ijk]_{n-3}}$ but constrained to be orthogonal to $\mathbf{v}_{[ijk]_n}$, $\mathbf{v}_{[ijk]_{n-1}}$ and $\mathbf{v}_{[ijk]_{n-2}}$. The orthogonalization can be calculated using the Gram-Schmidt procedure [78]. The corresponding step-size $\hat{\mu}_{r_{[ijk]_n}}$ is calculated as

$$\hat{\mu}_{r_{[ijk]_n}} = \begin{cases} \mu \dfrac{\hat{e}_{r_{[k]_n}}}{\|\mathbf{v}_{r_{[ijk]_n}}\|^2}, & \text{if } \|\mathbf{v}_{r_{[ijk]_n}}\|^2 \neq 0 \\ 0, & otherwise \end{cases} \tag{7.10}$$

where

$$\hat{e}_{r_{[k]_n}} = \hat{d}_{[k]_{n-rD}} + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}^T_{r_{[ij]_{n+1}}} \mathbf{v}_{[ijk]_{n-rD}}, \tag{7.11}$$

and

$$\begin{aligned} \mathbf{w}_{r_{[ij]_{n+1}}} = \mathbf{w}_{[ij]_n} - \sum_{k=1}^{K} \hat{\mu}_{0_{[ijk]_n}} \mathbf{v}_{0_{[ijk]_n}} - \sum_{k=1}^{K} \hat{\mu}_{1_{[ijk]_n}} \mathbf{v}_{1_{[ijk]_n}} - \dots \\ \dots - \sum_{k=1}^{K} \hat{\mu}_{r-1_{[ijk]_n}} \mathbf{v}_{r-1_{[ijk]_n}}. \end{aligned} \tag{7.12}$$

When R increases, the convergence is faster, but R must be chosen depending on the available computing resources. However, there exists a value of R where the convergence rate saturates. With regard to D, for low values of R, larger values of D provide better performance. This might be explained by the fact that when D increases, the successive input signals used in the weights update are less correlated. Table 7.2 summarizes the M-OCF algorithm.

7.2.3 The Frequency-domain Partitioned Block Modified Filtered-x NLMS with Orthogonal Correction Factors algorithm (FPM-OCF)

Finally, the FPM-OCF algorithm has been derived from the M-OCF algorithm. Following the nomenclature of this thesis, B is the block size, L is the length of the adaptive filters, and M is the length of the FIR filters that model the estimated secondary paths. F and P are the partitions of the adaptive filter length and the estimated secondary path length, respectively. Furthermore, the subscript n and the superscripts f and p of the

following notation denote block iteration and the number of each partition, respectively.

According to the notation, the FPM-OCF algorithm is described by the following steps:

1. Calculation of the output signals,

$$\mathbf{Y}_{[j]_n} = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}^f_{[ij]_n} \circ \mathbf{X}_{[i]_{n-f+1}}, \tag{7.13}$$

where $\mathbf{X}_{[i]_n} = \mathrm{FFT}_{2B}\{[\mathbf{x}^T_{[i]_{n-1}} \quad \mathbf{x}^T_{[i]_n}]^T\}$, $\mathbf{W}^f_{[ij]_n}$ is the FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}_{[ij]}$ at the nth block iteration. The output signals $\mathbf{y}_{[j]_n}$ are the last B samples of $\mathrm{IFFT}_{2B}\{\mathbf{Y}_{[j]_n}\}$.

2. Calculation of the estimated undesired signals,

$$\hat{\mathbf{D}}_{[k]_n} = \mathbf{E}_{[k]_n} - \mathbf{YF}_{[k]_n}, \tag{7.14}$$

where vector $\mathbf{E}_{[k]_n}$ is the kth microphone signal vector in the frequency domain at the nth iteration, and is obtained as $\mathbf{E}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{e}^T_{[k]_{n-1}} \quad \mathbf{e}^T_{[k]_n}]^T\}$. Vector $\mathbf{YF}_{[k]_n}$ is defined as

$$\mathbf{YF}_{[k]_n} = \sum_{j=1}^{J} \sum_{p=1}^{P} \mathbf{Y}_{[j]_{n-p+1}} \circ \mathbf{S}^p_{[jk]}. \tag{7.15}$$

3. Filter updates.

The update of the coefficients of each partition of the ijth adaptive filter is calculated in the frequency domain taking into account the R input vectors and is given by

$$\begin{aligned}\mathbf{W}^f_{[ij]_{n+1}} &= \mathbf{W}^f_{[ij]_n} - \\ &- \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^{f^T}_{0_{[ijk]_n}} \ \mathbf{0}_B^T]^T\} - \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^{f^T}_{1_{[ijk]_n}} \ \mathbf{0}_B^T]^T\} - \\ &- \ldots - \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^{f^T}_{R_{[ijk]_n}} \ \mathbf{0}_B^T]^T\},\end{aligned} \tag{7.16}$$

The input vectors are filtered through the corresponding secondary path to obtain the signal vectors $\mathbf{V}_{[ijk]}$. Vector $\mathbf{V}_{[ijk]}$ is defined at the nth block iteration as

$$\mathbf{V}_{[ijk]_n} = \sum_{p=1}^{P} \mathbf{S}^p_{[jk]} \circ \mathbf{X}_{[i]_{n-p+1}}. \tag{7.17}$$

Generalizing for an input vector r, the vector $\boldsymbol{\phi}^f_{r_{[ijk]_n}}$ corresponds to the first B samples of the 2B-IFFT of the corresponding partition $\widetilde{\boldsymbol{\mu}}^f_{r_{[ijk]_n}}$

$$\begin{gathered}[\boldsymbol{\phi}^{f^T}_{r_{[ijk]_n}} \quad \widetilde{\boldsymbol{\phi}}^{f^T}_{r_{[ijk]_n}}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\mu}}^f_{r_{[ijk]_n}}\}, \\ f = 1, 2, \ldots, F, and \ \ r = 1, 2, \ldots, R.\end{gathered} \tag{7.18}$$

We define $\mathbf{V}_{r_{[ijk]_{n-f+1}}}$ as the vector that is obtained from vector $\mathbf{V}_{[ijk]_{n-f+1-(rF)D}}$ with the constraint to be orthogonal to $\mathbf{V}_{[ijk]_{n-f+1}}$, $\mathbf{V}_{[ijk]_{n-f+1-(F)D}}$, $\mathbf{V}_{[ijk]_{n-f+1-(2F)D}}$, $\ldots$, $\mathbf{V}_{[ijk]_{n-f+1-((r-1)F)D}}$, where D is the distance between the input vectors. The corresponding step-size vector $\widetilde{\boldsymbol{\mu}}^f_{r_{[ijk]_n}}$ is calculated as

$$\widetilde{\boldsymbol{\mu}}^f_{r_{[ijk]_n}} = \begin{cases} \mu \dfrac{\hat{\mathbf{E}}_{r_{[k]_n}} \circ \mathbf{V}^*_{r_{[ijk]_{n-f+1}}}}{\|\mathbf{V}_{r_{[ijk]_{n-f+1}}}\|^2}, & \text{if } \|\mathbf{V}_{r_{[ijk]_{n-f+1}}}\|^2 \neq 0 \\ \mathbf{0}_{2B}, & otherwise \end{cases} \tag{7.19}$$

for $f = 1, 2, \ldots, F$. The K estimated error vectors in the frequency domain at the nth iteration, $\hat{\mathbf{E}}_{r_{[k]_n}}$, are obtained by performing the following equations

$$\boldsymbol{\xi}_{r_{[k]_n}} = \widehat{\mathbf{D}}_{[k]_{n-rD}} + \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{f=1}^{F} \mathbf{V}_{[ijk]_{n-f+1-(rF)D}} \circ \mathbf{W}^{f}_{r_{[ij]_n}}, \quad (7.20)$$

$$[\bar{\boldsymbol{\varepsilon}}^{T}_{r_{[k]_n}} \quad \boldsymbol{\varepsilon}^{T}_{r_{[k]_n}}]^{T} = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\xi}}_{r_{[k]_n}}\}, \quad (7.21)$$

and, finally

$$\widehat{\mathbf{E}}_{r_{[k]_n}} = \mathrm{FFT}_{2B}\{[\mathbf{0}^{T}_{B} \quad \boldsymbol{\varepsilon}^{T}_{r_{[k]_n}}]^{T}\}. \quad (7.22)$$

With regard to the R and D parameters, since this algorithm works by blocks of samples, even when $D = 1$, the successive input signals used in the weights update are separated by at least B samples. Moreover, when the reference signal $x_{[i]}$ is a white random noise, each block of samples $\mathbf{x}_{[i]_n}$ is orthogonal to the previous ones. This means that $\mathbf{V}_{[ijk]_{n-f+1-rF}}$ is sufficiently uncorrelated with $\mathbf{V}_{[ijk]_{n-f+1}}$, $\mathbf{V}_{[ijk]_{n-f+1-F}}$, $\mathbf{V}_{[ijk]_{n-f+1-2F}}$ …, $\mathbf{V}_{[ijk]_{n-f+1-(r-1)F}}$, so that we could obviate the orthogonalization procedure. The blocks of the signals are already uncorrelated due to the nature of the input signal, so we can take $\mathbf{V}_{r_{[ijk]_{n-f+1}}} = \mathbf{V}_{[ijk]_{n-f+1-rF}}$. This fact makes the NLMS-OCF algorithm especially suited to cancel broadband white noise signals, and for this reason, it is the case studied in section 7.4.

Note that the FPM-OCF algorithm in the particular case when $R = 0$ and $D = 1$ is similar to the one introduced in the previous chapter named FPBMFxLMS, but adding a power normalization in the weights update. However, for the sake of simplicity, in this chapter we will refer to the FPM-OCF algorithm with no orthogonal correction factors ($R = 0$) as the FPM algorithm. A summary of the algorithm instructions executed at each iteration is given in Table 7.2 for the M-OCF algorithm and in Table 7.3 for the FPM-OCF algorithm.

7.3 GPU implementation of the prototype

This section describes the main issues involved in the GPU implementation of the real-time multichannel ANC prototype based on the FPM-OCF algorithm. Details of the multichannel ANC prototype were given in section 2.3.

M-OCF algorithm

Choose an arbitrary $\mathbf{w}_{[ij]_0}$ and the number of Orthogonal Correction Factors $R < L$. Repeat the following steps at each new iteration.

1. $y_{[j]_n} = \sum_{i=1}^{I} \mathbf{w}_{[ij]_n}^T \mathbf{x}_{[i]_n}$,
   $\mathbf{y}_{[j]_n} = [y_{[j]_n} \quad y_{[j]_{n-1}} \quad \cdots \quad y_{[j]_{n-M+1}}]^T$
2. $\hat{d}_{[k]n} = e_{[k]_n} - \sum_{j=1}^{J} \mathbf{s}_{[jk]}^T \mathbf{y}_{[j]_n}$
3. $v_{[ijk]_n} = \mathbf{s}_{[jk]}^T \mathbf{x}_{[i]_n}$,
   $\mathbf{v}_{[ijk]_n} = [v_{[ijk]_n} \quad v_{[ijk]_n} \quad \cdots \quad v_{[ijk]_{n-L+1}}]^T$
4. $\hat{e}_{[k]_n} = \hat{d}_{[k]_n} + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}_{[ij]_n}^T \mathbf{v}_{[ijk]_n}$
5. $\hat{\mu}_{[ijk]_n} = \mu \frac{\hat{e}_{[k]_n}}{\|\mathbf{v}_{[ijk]_n}\|^2}$
6. $\mathbf{w}_{1_{[ij]_{n+1}}} = \mathbf{w}_{[ij]_n} - \sum_{k=1}^{K} \hat{\mu}_{[ijk]_n} \mathbf{v}_{[ijk]_n}$

For all $1 \leq r \leq R$, do

7. Compute $\mathbf{v}_{r_{[ijk]_n}}$ as the vector obtained from $\mathbf{v}_{[ijk]_{n-rD}}$ that is orthogonal to $\mathbf{v}_{[ijk]_n}, \mathbf{v}_{[ijk]_{n-D}}, \mathbf{v}_{[ijk]_{n-2D}}, \ldots, \mathbf{v}_{[ijk]_{n-(r-1)D}}$ using the Gram-Schmidt procedure [78].
8. $\hat{e}_{r_{[k]_n}} = \hat{d}_{[k]_{n-rD}} + \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbf{w}_{r_{[ij]_{n+1}}}^T \mathbf{v}_{[ijk]_{n-rD}}$
9. $\hat{\mu}_{r_{[ijk]_n}} = \begin{cases} \mu \frac{\hat{e}_{r_{[k]_n}}}{\|\mathbf{v}_{r_{[ijk]_n}}\|^2}, & \text{if } \|\mathbf{v}_{r_{[ijk]_n}}\|^2 \neq 0 \\ 0, & otherwise \end{cases}$
10. $\mathbf{w}_{r+1_{[ij]_{n+1}}} = \mathbf{w}_{r_{[ij]_{n+1}}} - \sum_{k=1}^{K} \hat{\mu}_{[ijk]n}^r \mathbf{v}_{r_{[ijk]_n}}$

end for

11. $\mathbf{w}_{[ij]_{n+1}} = \mathbf{w}_{R_{[ij]_{n+1}}}$

Table 7.2. Summary of the M-OCF algorithm.

FPM-OCF algorithm

Choose an arbitrary $\mathbf{W}_{[ij]_0}$ and the number of Orthogonal Correction Factors $R < L$. Repeat the following steps at each new iteration. All the equations with references to partitions are performed for $f = 1, 2, \ldots, F$.

1. $\mathbf{Y}_{[j]_n} = \sum_{i=1}^{I} \sum_{f=1}^{F} \mathbf{W}^f_{[ij]_n} \circ \mathbf{X}_{[i]_{n-f+1}}$
2. $\hat{\mathbf{D}}_{[k]_n} = \mathbf{E}_{[k]_n} - \mathbf{YF}_{[k]_n}$, where $\mathbf{E}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{e}^T_{[k]_{n-1}} \quad \mathbf{e}^T_{[k]_n}]^T\}$.
3. $\mathbf{YF}_{[k]_n} = \sum_{j=1}^{J} \sum_{p=1}^{P} \mathbf{Y}_{[j]_{n-p+1}} \circ \mathbf{S}^p_{[jk]}$
4. $\mathbf{V}_{[ijk]_n} = \sum_{p=1}^{P} \mathbf{S}^p_{[jk]} \circ \mathbf{X}_{[i]_{n-p+1}}$
5. $\boldsymbol{\xi}_{[k]_n} = \widehat{\mathbf{D}}_{[k]_n} + \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{f=1}^{F} \mathbf{V}_{[ijk]_{n-f+1}} \circ \mathbf{W}^f_{[ij]_n}$
6. $[\bar{\boldsymbol{\varepsilon}}^T_{[k]_n} \quad \boldsymbol{\varepsilon}^T_{[k]_n}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\xi}}_{[k]_n}\}$
7. $\widehat{\mathbf{E}}_{[k]_n} = \mathrm{FFT}_{2B}\{[\mathbf{0}^T_B \quad \boldsymbol{\varepsilon}^T_{[k]_n}]^T\}$
8. $\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n} = \mu \frac{\hat{\mathbf{E}}_{[k]_n} \circ (\mathbf{V}_{[ijk]_{n-f+1}})^*}{\|\mathbf{V}_{[ijk]_{n-f+1}}\|^2}$
9. $[\boldsymbol{\phi}^{f^T}_{[ijk]_n} \quad \bar{\boldsymbol{\phi}}^{f^T}_{[ijk]_n}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\mu}}^f_{[ijk]_n}\}$
10. $\mathbf{W}^f_{1_{[ij]_{n+1}}} = \mathbf{W}^f_{[ij]_n} - \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^{f^T}_{[ijk]_n} \quad \mathbf{0}^T_B]^T\}$

For all $1 \leq r \leq R$, do

11. Compute $\mathbf{V}_{r_{[ijk]_{n-f+1}}}$ as the vector obtained from $\mathbf{V}_{[ijk]_{n-f+1-(rF)D}}$ that is orthogonal to $\mathbf{V}_{[ijk]_{n-f+1}}$, $\mathbf{V}_{[ijk]_{n-f+1-(F)D}}$, $\mathbf{V}_{[ijk]_{n-f+1-(2F)D}}$, ..., $\mathbf{V}_{[ijk]_{n-f+1-((r-1)F)D}}$.
12. $\boldsymbol{\xi}_{r_{[k]_n}} = \widehat{\mathbf{D}}_{[k]_{n-rD}} + \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{f=1}^{F} \mathbf{V}_{[ijk]_{n-f+1-(rF)D}} \circ \mathbf{W}^f_{r_{[ij]_n}}$
13. $[\bar{\boldsymbol{\varepsilon}}^T_{r_{[k]_n}} \quad \boldsymbol{\varepsilon}^T_{r_{[k]_n}}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\xi}}_{r_{[k]_n}}\}$
14. $\widehat{\mathbf{E}}_{r_{[k]_n}} = \mathrm{FFT}_{2B}\{[\mathbf{0}^T_B \quad \boldsymbol{\varepsilon}^T_{r_{[k]_n}}]^T\}$

15. $\widetilde{\boldsymbol{\mu}}^{f}_{r_{[ijk]_n}} = \begin{cases} \mu \dfrac{\hat{\mathbf{E}}_{r_{[k]_n}} \circ \mathbf{V}^{*}_{r_{[ijk]_{n-f+1}}}}{\|\mathbf{V}_{r_{[ijk]_{n-f+1}}}\|^2}, & \text{if } \|\mathbf{V}_{r_{[ijk]_{n-f+1}}}\|^2 \neq 0 \\ \mathbf{0}_{2B}, & otherwise. \end{cases}$

16. $[\boldsymbol{\phi}^{f^T}_{r_{[ijk]_n}} \quad \widetilde{\boldsymbol{\phi}}^{f^T}_{r_{[ijk]_n}}]^T = \mathrm{IFFT}_{2B}\{\widetilde{\boldsymbol{\mu}}^{f}_{r_{[ijk]_n}}\}$,

17. $\mathbf{W}^{f}_{r+1_{[ij]_{n+1}}} = \mathbf{W}^{f}_{r_{[ij]_{n+1}}} - \sum_{k=1}^{K} \mathrm{FFT}_{2B}\{[\boldsymbol{\phi}^{f^T}_{r_{[ijk]_n}} \quad \mathbf{0}^T_B]^T\}$

end for

17. $\mathbf{W}^{f}_{[ij]_{n+1}} = \mathbf{W}^{f}_{R_{[ij]_{n+1}}}$

Table 7.3. Summary of the FPM-OCF algorithm.

The GPU implementation consists of three steps that are depicted in Figure 7.2: the output signal generation, the estimation of the "undesired" signals, and the update of the adaptive filters. These three steps are implemented as follows:

S1 *Calculation of the output signals.* This step aims to calculate the ANC output signals $y_{[j]_n}$. The operations of this step correspond to Eq. (7.13). The implementation and the CUDA kernels involved in it are shown in Figure 7.2(a).

S2 *Calculation of the estimated "undesired" signals.* This step calculates an estimate of the undesired signals. The corresponding description is shown in Eq. (7.14) and Eq. (7.15). The implementation and the CUDA kernels involved in it are shown in Figure 7.2(b).

S3 *Filter updates.* The update of the adaptive filter coefficients involve the implementation of Eq. (7.16)-(7.22). The details regarding the different steps and kernels are illustrated in Figure 7.2(c).

The GPU implementation depicted in Figure 7.2 makes use of five optimized kernels in order to achieve the most efficient performance of the algorithm. As commented in the previous chapters, the kernels are those presented in chapter 5 for a multichannel AE application and reused in

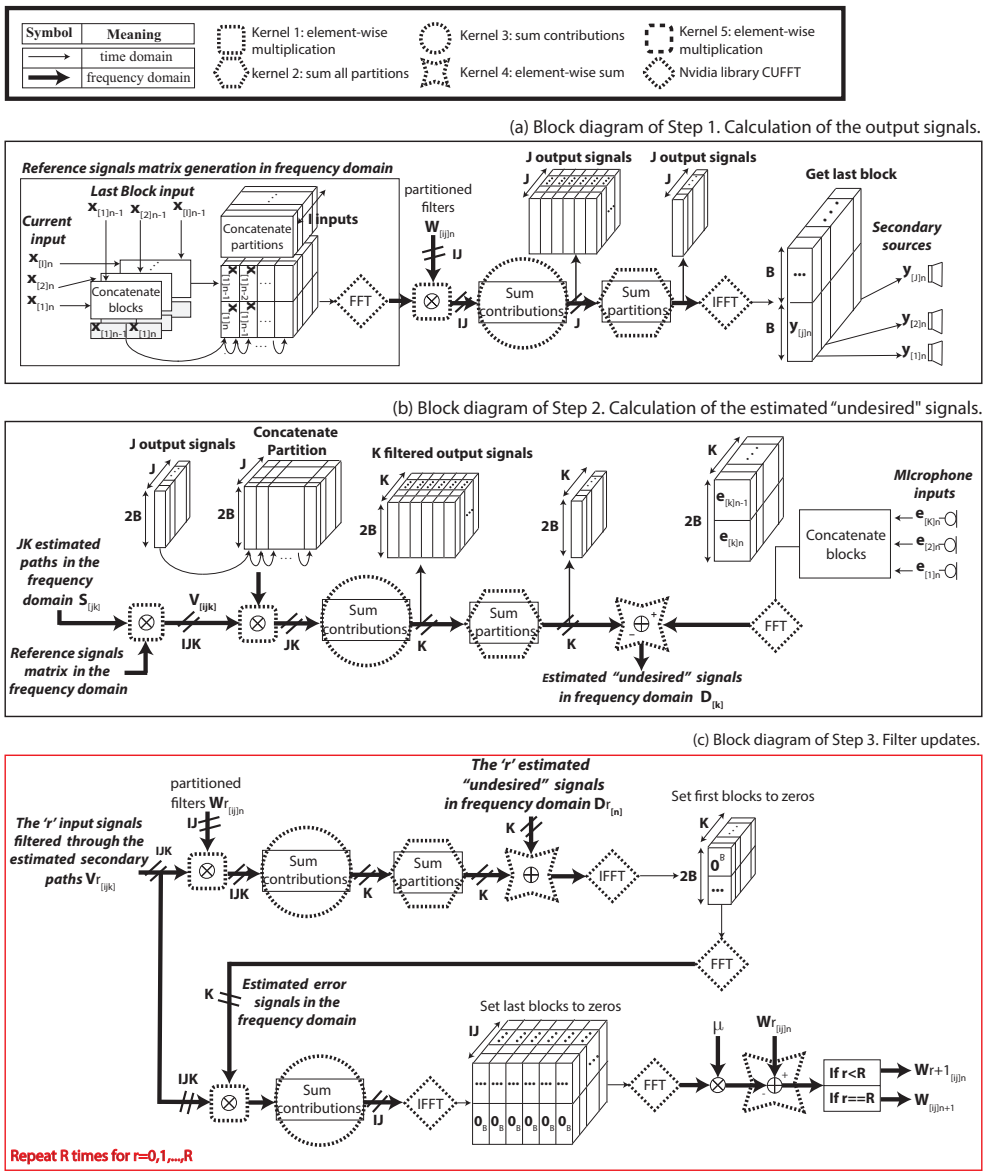


Figure 7.2. Block diagram of the GPU implementation of the FPM-OCF algorithm.

chapter 6 for an ANC application. Therefore, details of the implementation of the Kernels depicted in Figure 7.2 can be found in section 5.3.

7.4 Results

Several experiments were performed to validate the ANC system based on the FPM-OCF algorithm. The experiments were carried out using the prototype described in section 2.3. The experiments consider different I:J:K configurations of the ANC system varying the value of J and K, but considering only a reference signal ($I = 1$).

In consonance with the other chapters, the ANC system has been evaluated from two points of view: the algorithm and the computing behavior. Similarly to what has been done in the previous chapter, the first two sections are devoted to analyze de algorithm behavior in terms of attenuation levels and speed of convergence. Moreover, the computational complexity of the GPU implementation of the algorithm is also analyzed in section 7.4.3 for different values of R. Finally, section 7.4.4 presents the computing results of the GPU implementation of the FPM-OCF algorithm. Since both the previous and the current chapter present GPU implementations of ANC systems, the computing analysis of both chapters follows the same pattern.

7.4.1 Residual noise levels

A multichannel ANC system with dimensions $I = 1$, $J = 2$ and $K = 2$ (1:2:2 configuration) has been considered in this experiment. The following parameters were chosen: $B = 2048$, $L = M = 4096$, $R = 0$, and $R = 4$, with broadband white random noise as reference signal.

Figure 7.3 shows the power spectral density of the average signals measured at both error sensors by using the proposed algorithm when $R = 0$ (FPM algorithm) and $R = 4$. The noise reductions showed in Figure 4 are obtained at the specific points where the error sensors are placed. A similar noise reduction performance is achieved in both cases. It can be easily observed that a reduction of around 25-30 dB can be achieved up to about 20 kHz. We can conclude that the noise reduction in the steady state does not depend on the R value.

7.4.2 Convergence performance

The convergence performance of the NLMS-OCF has been theoretically analyzed for system identification configuration in [45]. The theoretical

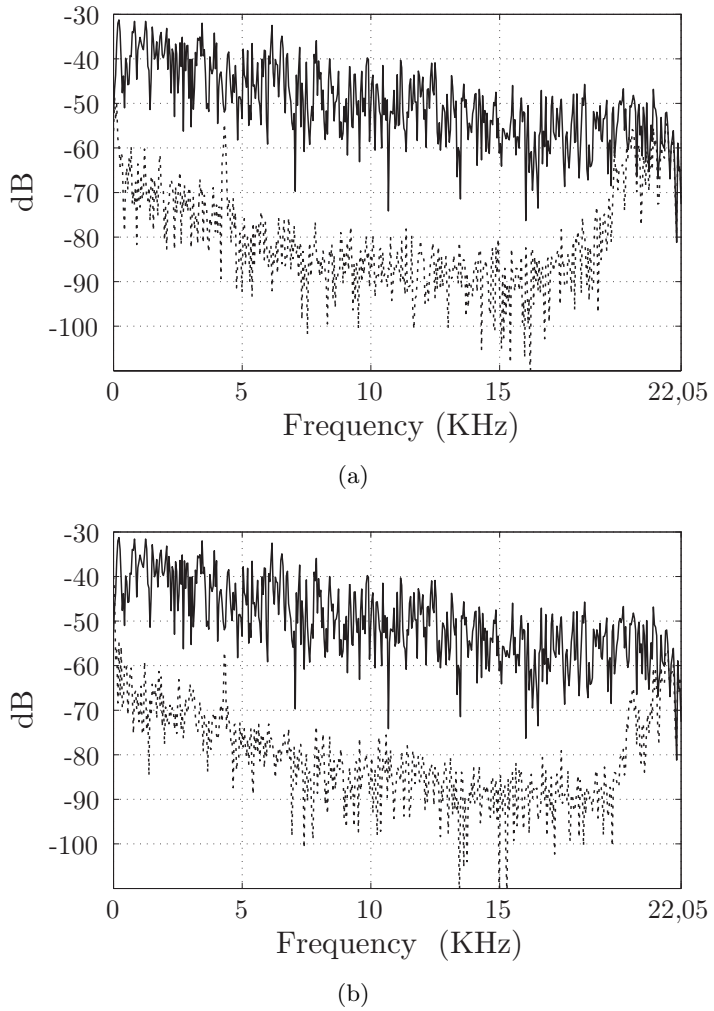


Figure 7.3. Power spectral density of the average of the signals measured at both error sensors before (solid line) and after (dashed line) the ANC system operation by using the FPM-OCF algorithm in two cases: (a) $R = 0$ (FPM algorithm), and (b) $R = 4$.

analysis is also validated with simulation results. Specifically, in [45], it has been shown how the NLMS-OCF algorithm outperforms the NLMS algorithm in terms of convergence speed. Moreover, the benefits in terms

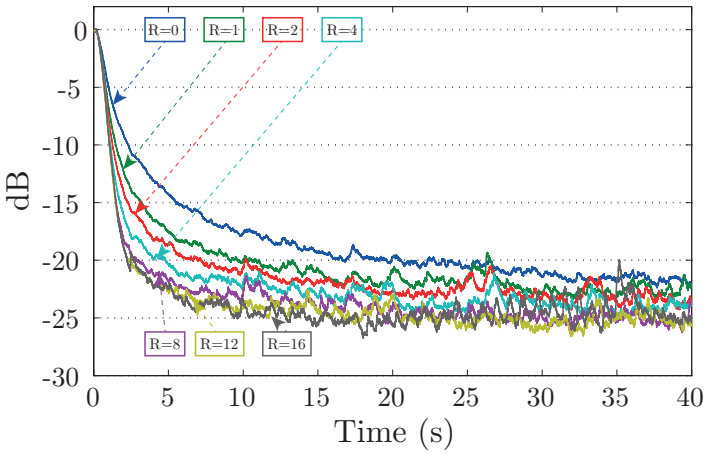


Figure 7.4. The learning curves of the FPM-OCF algorithm for different values of R and using white noise as the reference signal.

of convergence speed of the NLMS-OCF algorithm have also been shown when compared with the APA family algorithms in [118]. In this chapter, the convergence performance of the FPM-OCF algorithm is evaluated for an ANC application when varying the value of R. As it has been said, when $R = 0$, the FPM-OCF algorithm becomes the FPM algorithm. The FPM solves a mean square problem using a gradient descent algorithm while the FPM-OCF solves a constrained minimization problem. To reach the solution, both algorithms use an iterative procedure. However, the FPM-OCF algorithm uses R input vectors instead of only one. For this reason, the FPM-OCF algorithm converges faster than the FPM, but exhibits a higher computational cost. In order to evaluate the algorithm performance, the learning curves were obtained for the experiments as:

$$A_n = 10 \log_{10} \left(\frac{Pe_n}{Pd_n} \right), \tag{7.23}$$

with

$$\begin{aligned} Pd_n &= \alpha Pd_{n-1} + (1-\alpha) pd_n, \\ Pe_n &= \alpha Pe_{n-1} + (1-\alpha) pe_n, \end{aligned} \tag{7.24}$$

and

$$pd_n = \sum_{k=1}^{K} d^2_{[k]_n},$$
$$pe_n = \sum_{k=1}^{K} e^2_{[k]_n}. \tag{7.25}$$

with K being the number of error sensors.

Figure 7.4 illustrates the simulations results with the configuration used in subsection 7.4.1. In this figure it is shown that when the value of R grows, the FPM-OCF provides a faster convergence rate. However, there is a value of R where the increase in the speed of convergence saturates. Specifically, the fastest convergence rate is obtained when 12 input vectors are used to update the weights (R=12), and, consequently, a higher value of R does not result in a faster convergence rate. On the other hand, the learning curves show a similar steady-state behavior, which confirms the results of the subsection 7.4.1. Focusing on the transient, the algorithm reaches 20 dB of attenuation in approximately 2 seconds when $R = 12$, while the algorithm takes around 17 seconds when $R = 0$, which means that the convergence speed is accelerated 8.5 times. Another important aspect is that a good convergence performance is obtained even with low R values. For example, when the value of R is increased from $R = 0$ to $R = 1$, the convergence time is more or less halved.

7.4.3 Computational complexity

The results of the subsection 7.4.2 demonstrate that the FPM-OCF algorithm outperforms the convergence rate of the FPM when $R > 0$. However, the computational complexity increases with the increase of R. Therefore, this section is devoted to analyzing its computational complexity, which is the major drawback of the algorithm. To this end, Table 8.2 compares the computing time and the computational complexity in terms of multiplications, additions, and FFTs per iteration of the GPU implementation of the FPM-OCF algorithm. It is done for different I:J:K configurations and different R values. As before, note that the FPM-OCF algorithm becomes the FPM algorithm when $R = 0$. Since we use a value of $M = L$ and B=$L/2$

	I:J:K configuration								
	1:1:1			1:2:2			1:4:4		
	R=0	R=4	R=16	R=0	R=4	R=16	R=0	R=4	R=16
Multiplications	$13L$	$33L$	$93L$	$44L$	$112L$	$316L$	$160L$	$420L$	$1200L$
Additions	$10L$	$34L$	$106L$	$36L$	$124L$	$388L$	$138L$	$474L$	$1482L$
FFTs	10	34	96	18	66	210	34	130	418
Time (ms)	0.73	1.91	5.46	1.16	3.22	9.29	2.25	6.64	19.46
M_R/M_o	1	2.5	7.2	1	2.5	7.2	1	2.5	7.5
A_R/A_0	1	3.4	10.6	1	3.4	10.8	1	3.4	10.7
$\mathrm{FFT_R/FFT_0}$	1	3.4	9.6	1	3.7	11.7	1	3.8	12.3
t_R/t_0	1	2.6	7.5	1	2.8	8.0	1	2.9	8.6

Table 7.4. Processing time and total number of multiplications, additions and FFTs per iteration of the GPU implementation of the FPBMFxLMS-OCF algorithm for different ANC configurations and varying the value of R when $L = M$.

(filters are split up into 2 partitions), the computational complexity only depends on L. It is important to note that the computational complexity can be analyzed by changing the I:J:K configuration (and therefore the number of channels) and also by changing the number of input vectors (R) used in the filter coefficients update.

First, Table 8.2 shows that, for a given value of R, the computational complexity of the algorithm increases significantly with the increase in the number of channels. As an example, when $R = 0$ (FPM algorithm), if the ANC configuration changes from 1:1:1 to 1:4:4 (16 secondary paths), the number of multiplications increases by a factor of 12, the additions by a factor of 13.8, and the FFTs by a factor of 3.4 while the time delay increases only by a factor of 3. Approximately the same occurs if $R > 0$. For example, for a value of $R = 16$ and changing the arrangement configuration from 1:1:1 to 1:4:4, the number of multiplications increases by a factor of 13, the additions by a factor of 14, and the FFTs by a factor of 4.3 while the time delay increases only by a factor of 3.3. This behavior with respect

to the number of channels is similar to that offered by the algorithms of the previous chapter. Therefore, since the increase in the number of channels greatly increases the complexity, we can conclude that the computational complexity is a bottleneck of massive multichannel ANC systems regardless of the value of R. However, if the operations of each channel are properly parallelized, an implementation over GPU can reduce this computational drawback and could be a viable and meaningful solution.

On the other hand, Table 8.2 also shows that when $R > 0$, the FPM algorithm exhibits higher computational complexity than without using correction factors ($R = 0$). This is due to the use of the R input signal vectors in the tap weight update (see Figure 7.2, box in red). As Table 8.2 shows, the number of multiplications, additions and FFTs significantly increases. Let us define the ratio M_R/M_0 as the number of multiplications of the FPM-OCF algorithm when $R > 0$, divided by the number of multiplications of the FPBM algorithm ($R = 0$). We define the same ratio for additions (A_R/A_0), FFTs ($\mathrm{FFT}_R/\mathrm{FFT}_0$), and processing delay (t_R/t_0). It is shown that although the ratio of processing delay is similar to the ratio of multiplications and is less than the ratios of additions and FFTs, the difference among them is not significant. This can be explained by the fact that the processing of step 3 in Figure 7.2 must be executed in a sequential mode because the coefficients of the adaptive filters $\mathbf{W}_{r_{[ij]_{n+1}}}$ are needed for the calculation of the filters $\mathbf{W}_{r+1_{[ij]_{n+1}}}$, and, therefore, it can not be parallelized. However, even though the calculation of $\mathbf{W}_{r_{[ij]_{n+1}}}$, $\mathbf{W}_{r+1_{[ij]_{n+1}}}$, ..., $\mathbf{W}_{R_{[ij]_{n+1}}}$ can not be implemented in parallel, the calculation of each one of the R times that the coefficients are updated at each iteration could be accelerated by being implemented on a parallel computing device such as a GPU.

Finally, it is important to emphasize that, as showed in sections 7.4.2 and 7.4.3, both the improvement in convergence rate and the increase in complexity are linked to the number of correction factors. Therefore, this leads to an adjustable and configurable implementation of the FPM-OCF algorithm with regards to the complexity and convergence rate.

7.4.4 Prototype computing performance

As is well known, the ANC prototype can extend the zone of cancellation by properly adding more sensors and transducers. However, as we have

seen in the subsection 7.4.3, the computational cost can become extremely large, especially when $R > 0$.

This section is devoted to studying the computational constraints of the multichannel ANC prototype based on the FPM -OCF algorithm. For this purpose, Figure 7.5 shows the maximum number of channels that the GPU-based ANC system can handle without violating the real-time condition for L=M=4096, $B = 2048$, for one reference signal ($I = 1$) and different values of R. This maximum number of channels is calculated by fixing the number of error sensors and finding the maximum number of actuators (J) that the system can feed without violating the real-time condition. When the maximum J value for each value of K is found, the maximum number of physical channels that are processed for each value of K is $J \cdot K$. For example, for the curve labeled as $R = 0$ in Figure 7.5, when $K = 170$, the maximum number of actuators that can be used without violating the real-time condition is $J = 3$; therefore, the maximum number of processed channels is 510. However, for $K = 171$, the maximum number of actuators is $J = 2$ because the real-time condition is violated with $J = 3$. Thus, the maximum number of processed channels is 342 ($J = 2$) when $K = 171$. For this reason, the shapes of the curves jump with the increase in error sensors.

The following considerations are highlighted in the simulation results depicted in Figure 7.5:

- The maximum number of actuators for each value of K is calculated by taking into account that it has to satisfy the real-time condition: $t_{proc} < B/f_s$. Therefore, if R increases, there are more operations to perform in the same time period and thus less channels can be handled.

- Two systems with different I:J:K configurations could have the same number of physical channels but different computational costs. For example, both the 1:1:2 and 1:2:1 configurations have 2 physical channels, but the second configuration exhibits a higher computational cost because it has to update two adaptive filters instead of one. Therefore, an increase in the number of adaptive filters involves many more operations than an increase in the number of error sensor signals, with IJ being the number of adaptive filters. Furthermore, when K is low and J is high, the maximum number of channels is

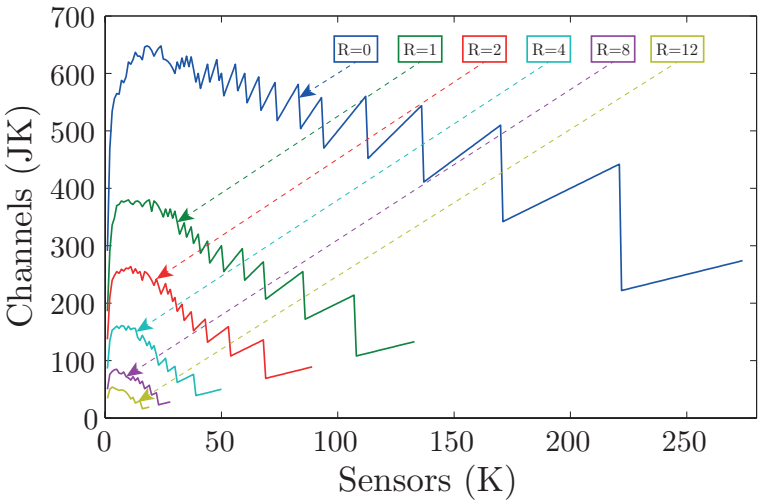


Figure 7.5. Maximum number of channels (JK) for each K value performing in real-time when I=1 for different values of R, $B = 2048$, and L=M=4096.

limited by the delay of processing the adaptive filters (see Figure 7.5 when K is low).

- When K increases, J has to decrease in order to satisfy the real-time condition, and, consequently, the number of adaptive filters decreases and the curves of the number of processed channels grow quickly reaching the maximums. The maximum of the curves is reached when neither K nor J is much bigger than the other.

- On the other hand, when J is small and K is large, even though the number of adaptive filters is low and therefore less operations have to be carried out, the system operation is limited by the handling of the estimated undesired signals and the estimated error signals, which involve memory transfers and flow control instructions (see Figure 7.2). Therefore, the maximum number of channels decreases with the increase of K. Moreover, since the calculation of the estimated error signal is repeated R times (one for each of the R input vectors used in the tap weight update), when the value of R increases, the decrease in the number of channels with the increase of K is bigger than when $R = 0$.

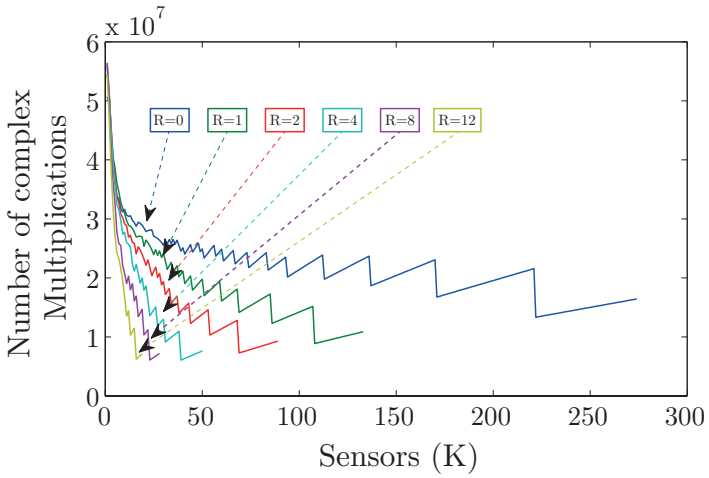
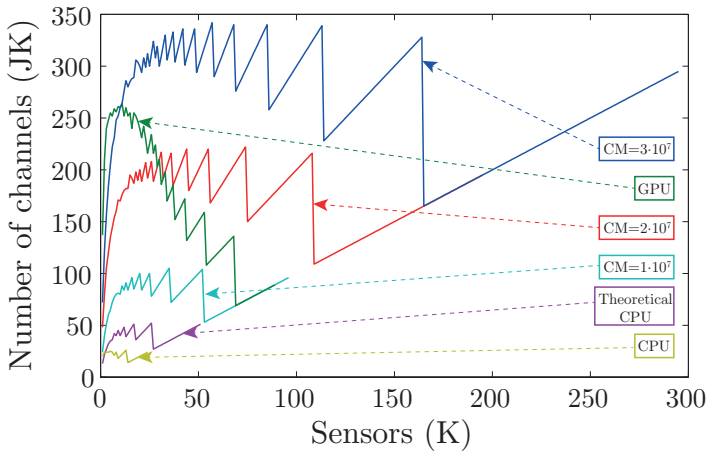
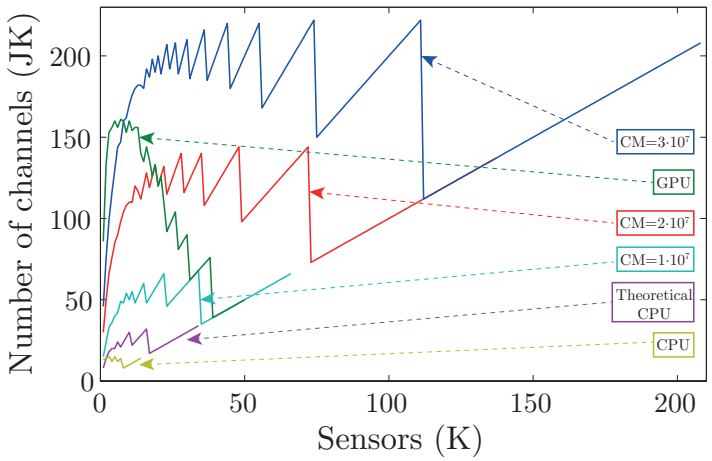


Figure 7.6. Number of complex multiplications (CM) performed for the maximum JK configuration when I=1 for different values of R, $B = 2048$, and L=M=4096.

Once the maximum number of processed channels has been analyzed for each value of K, the number of complex multiplications (CM) involved in both the products and the FFTs for the JK configurations derived in Figure 7.5 is depicted in Figure 7.6. A CM involves 4 floating point multiplications and 2 floating point additions. It can be observed that for a given value of R, the curve of the number of CM performed is not constant for the different JK configurations. As commented above, this is because the GPU implementation is affected by the JK configuration. The most remarkable aspect of Figure 7.6 is that the JK configurations with more CM are those with low values of K and high values of J. As explained above, the number of adaptive filters depends on both the I and J variables; therefore, when J grows, more CM are performed because there are more adaptive filters to cope with. Moreover, as the value of K grows, the handling of the estimated "undesired" signals and the estimated error signals limits the processing and therefore less CM are performed. Finally, if we compare the curves with different values of R, it is shown that for low values of K, all of the curves perform a similar number of complex multiplications, but as K increases, the decrease in the number of performed complex multiplications is accentuated with the value of R. This is because the algorithm performs the handling of the K error signals R times.



(a)



(b)

Figure 7.7. A comparison of the maximum number of processed channels for the GPU implementation, the CPU implementation, and a theoretical processing machine limited to perform $1 \cdot 10^7$, $2 \cdot 10^7$, or $3 \cdot 10^7$ complex multiplications (CM) per block of samples for $B = 2048$ and $I = 1$.

The maximum number of channels that can be handled in real time by the GPU is shown in Figure 7.5 for different values of R. Now, the same way it was done in the previous chapter, we compare the computational

capabilities of the GPU with other hardware platforms. To this end, the ANC system based on the FPM-OCF algorithm has been implemented on a CPU i7 but using one core and a sequential execution. In this sense, the maximum number of channels allowed by the CPU implementation was theoretically and practically studied. Moreover, in line with the computing analysis of the previous chapter, the GPU and CPU evaluation results were also compared with a theoretical processing machine that was limited to perform $1 \cdot 10^7$, $2 \cdot 10^7$, or $3 \cdot 10^7$ CM per buffering time. The results of this comparison are illustrated in Figure 7.7 for the case when $I = 1$, $M = L = 4096$, $B = 2048$, and $R = 2$ (Figure 7.7.a) or $R = 4$ (Figure 7.7.b). In this case, the buffering time is $B/f_s = 2048/44100 = 0.0464$ seconds. For example, the curve labeled as 'CM=$1 \cdot 10^7$' in Figure 7.7 represents the maximum number of channels for each value of K that a given machine would process if this machine was able to carry out $1 \cdot 10^7$ CM every 46.4 milliseconds. Finally, the theoretical maximum performance of our CPU was found the same way it was done in the previous chapter: since a floating point multiplication and a floating point addition are performed by our CPU in 5 and 3 clock cycles, respectively (see annex 3 of [113]), a CM involves 26 clock cycles. The CPU operates at 3.07GHz, which means that our CPU could make $3.07 \cdot 10^9/26$ CM per second and $(3.07 \cdot 10^9/26) \cdot 0.0464 \simeq 5.5 \cdot 10^6$ CM per buffering time. This is represented in the figure with the curve labeled as 'theoretical CPU'. Since our CPU also performs memory transactions and flow control instructions, the practical CPU implementation handles fewer channels than the 'theoretical CPU'. Furthermore, the theoretical processing machine that processes $1 \cdot 10^7$, $2 \cdot 10^7$, or $3 \cdot 10^7$ CM per buffering time would has also to perform memory transactions and flow control instructions in addition to the complex multiplications. Therefore, in practice, these three curves would be lower.

Figure 7.7 shows that this algorithm can not be used with $R > 0$ for multichannel ANC systems using the i7 CPU as the processor. Therefore, expensive CPUs would be needed. On the other hand, the figure illustrates that the GPU implementation outperforms the CPU implementation for both values of R and also shows the number of CM that a processing machine would have to carry out each buffering time to outperform the GPU implementation. Moreover, the maximum benefit of the GPU is obtained when low values of K and high values of J are used, where the GPU carries out $3 \cdot 10^7$ CM per buffering time. As in previous sections, this can

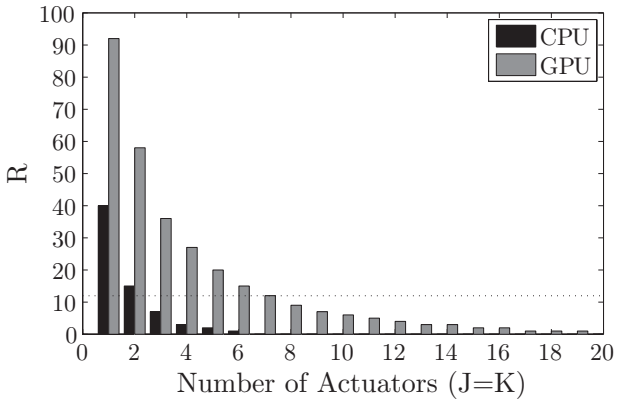


Figure 7.8. Maximum number of input vectors (R) that can be used in the tap weight update by the CPU and GPU configurations for square I:J:K configurations when $I = 1$, $B = 2048$, and L=M=4096.

be explained by the fact that an increase in the value of K involves an increase in the time required to handle the estimated undesired signals and the estimated error signals in the frequency domain. This fact is accentuated with the increase in the value of R. As a conclusion, Figure 7.7 shows that even though the increase of the value of R results in a high increase in the computational complexity, the algorithm can be implemented in a real-time ANC prototype using a GPU and a proper parallelization of the operations of the multiple channels.

Finally, in order to better clarify the computational limitations of the CPU implementation, Figure 7.8 depicts the maximum value of R that can be used by both the CPU implementation and the GPU implementation for different I:J:K configurations. It has been calculated for square I:J:K configurations because they are the most representative. This means configurations with one reference signal (I=1) and the same number of loudspeakers as microphones (J=K). The figure also shows (with a dotted line) the saturation value of R where it saturates; in this case, $R = 12$. It is shown that the CPU implementation can reach the saturation value of R only for the single channel configuration and the 1:2:2 configuration, whereas the GPU configuration can reach up to the 1:7:7 configuration. Moreover, the maximum channel configuration that can be implemented on a CPU with a value of $R = 1$ is the 1:6:6 configuration, which involves

36 channels processed in real time. In contrast, the GPU implementation reaches the 1:19:19 configuration, which means 361 channels processed in real time.

7.5 Conclusions

The work of this chapter continues with the study of the use of GPUs for implementing ANC systems. To this end, the M-OCF algorithm has been introduced for a multichannel active noise control system. This algorithm has been derived from the NLMS-OCF algorithm that had been previously introduced for system identification. To meet the hardware requirements, the M-OCF algorithm has been developed in the frequency domain, working with blocks of data and partitioning the adaptive filters, resulting in the FPM-OCF algorithm. Moreover, it has been shown that the FPBFxLMS algorithm presented in the previous chapter is a particular case of the FPM-OCF algorithm when $R = 0$, but without the power normalization in the weights update.

The results presented in this chapter has proved that the convergence rate of the FPM-OCF algorithm improves with the increase in the value of R. Consequently, when $R > 0$ this algorithm converges faster than the FPBFxNLMS algorithm ($R = 0$). Even though the increase in the value of R results in a high increase in the computational complexity (specially for massive multichannel systems), it has been demonstrated that by taking advantage of the parallelization capabilities of the GPU, the increase in computational complexity due to the increase of channels produces a reduced increase in the processing delay. Therefore, the use of a GPU platform can help to overcome the disadvantage of the FPM-OCF in real-time for multichannel ANC systems. Moreover, both the improvement in convergence rate and the increase in computational complexity can be controlled and adjusted with the number of correction factors used in the execution of the algorithm, and, therefore, the value of R can be set depending on the performance requirements. As a conclusion, we have demonstrated that although using a current cheap GPU (GeForce GTX 580), the GHPU implementation of the FPM-OCF algorithm in a real-time multichannel ANC prototype is feasible, and, capable of processing hundreds of channels in real time, depending on the value of R.

On the other hand, a computing analysis of the GPU implementation has been carried out. This analysis has shown that a commitment relation between the convergence needing (value of R) and the available computing resources (CPU/GPU) is necessary depending on the size of the multichannel structure (I:J:K configuration). For example, in a single channel structure or a small multichannel structure (e.g 1:2:2), the FPM-OCF algorithm can be implemented sequentially in a CPU using the maximum number of correction factors without saturating. However, in bigger multichannel structures, a parallelization of the operations is needed due to the big computational cost, and therefore, the GPU can be used. It is also important to note that with a considerable number of R, for example $R = 8$ in the case of section 7.4.2, the gain in speed of convergence from $R = 8$ to $R = 12$ is not significantly. Therefore, $R = 8$ could be considered as the saturation value. In this case, larger multichannel structures could be obtained with $R = 8$ instead of $R = 12$. As a conclusion, the optimum relation between R and I:J:K depends on the application needing and the available computing resources. Finally, as it has been pointed out in all the chapters, the results of this chapter in terms of number of processed channels could be grater by using a different audio card with lower frequency sampling, decimating, or even using newer GPUs with better computational capacity.

The main results of this chapter were published in the Digital Signal Processing Journal of Elsevier [119].

8 Distributed Active Noise Control

8 Distributed Active Noise Control

The previous chapters have demonstrates that the GPU is a meaningful device for the implementation of multichannel Active Noise Control (ANC) systems, allowing the processing of hundreds of channels in real-time. However, sometimes, big multichannel systems can't be considered due to hardware or physical constraints. For this reason, this chapter goes a step forward in the development of multichannel ANC systems by proposing a multichannel ANC system over a network of distributed acoustic nodes. In the distributed case, the processing of a centralized ANC system is divided into several nodes. Therefore, this chapter presents a new formulation to introduce a distributed algorithm together with an incremental collaborative strategy in the network. Results of this chapter demonstrate that the scalable and versatile distributed algorithm can exhibit the same performance as the centralized version. Finally, the computational complexity and some implementation aspects have been analyzed.

8.1 Introduction

A wireless acoustic sensor network (WASN) [120] is a type of wireless sensor network (WSN) [121, 122] whose sensor devices are microphones. It is a

cheap, flexible and efficient solution that is generally used for monitoring acoustic fields. The acoustic nodes are commonly composed of one or more microphones that are used to collect signals and a processor with some kind of communication and computation capability. The way the signals are processed in each node depends on the network topology [123].

Some applications that make use of a WASN are presented in [124] and the references therein. There, the acoustics nodes are usually used to record signals through microphones, process them and even share the signals or some local and network parameters. However, in some applications like active noise control, the nodes have to act on their own environment. To this end, the nodes have to own loudspeakers.

The goal of this chapter is to study the usefulness of the acoustic sensor networks for ANC systems. To this end, we propose an ANC system working over a distributed network with an incremental approach in a ring topology [125]. In literature, one can find other works where the distributed networks have been applied to ANC system. As a representative example, in [126], a distributed ANC system based on the Me-FxLMS algorithm was presented in the time domain with a sample-by-sample acquisition. However, as commonly happens in practical systems, we propose to work with block processing and in the frequency domain. Therefore, in line with the centralized ANC system introduced in chapter 6, the distributed ANC system presented in this chapter is based on the FPBFxLMS algorithm, where the goal is to minimize the sum of the power of the error sensors. Hence, the approach presented in this chapter is the applicability of the FPBFxLMS algorithm to a distributed ANC system.

The centralized ANC systems have been discussed along this thesis. As it has been commented, the ANC systems can be extended to multichannel ANC systems by overlapping different controlled areas and setting multiple secondary sources. These multichannel systems require a high computational capacity. However, the multichannel ANC systems can be divided into a network with smaller multichannel nodes or single-channel nodes, and therefore, the computational cost is divided into the different nodes. It is another way of forming big multichannel ANC systems. Figure 8.1 illustrates it. Finally, this chapter presents some simulation results that demonstrate that the performance of both the centralized and the distributed ANC systems are exactly the same.

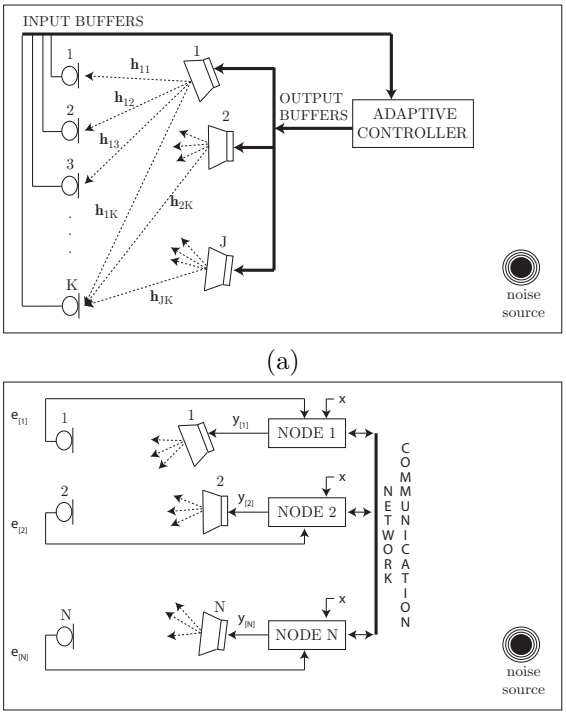


Figure 8.1. Schemes of (a) a centralized ANC system, (b) a distributed ANC system with single-channel nodes.

8.2 Description of the algorithm

In chapter 6, the FPBFxLMS algorithm was presented for a generic centralized ANC system with I reference signals, J secondary sources, and K error sensors (I:J:K configuration). The block diagram of the centralized ANC system based on the FPBFxLMS algorithm is shown in Figure 6.1.(a).

Here, we derive the centralized version of the so-called FPBFxLMS algorithm to a distributed ANC system with an incremental topology. The incremental topology means that the nodes collaborate by transmitting information to an adjacent node in a consecutive order. For the sake of simplicity, we consider one disturbance noise (I=1) and single-channel nodes (J=K=1). Therefore, each node is composed of a processor, a microphone and a loudspeaker. For a better understanding, the FPBFxLMS is first presented to a single channel node, and then, it is extended to a network

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
$x_{[i]_n}$	ith reference signal at sample n
$\mathbf{x}_{[i]_n}$	$[x_{[i]_{Bn}} \quad x_{[i]_{Bn-1}} \quad \cdots \quad x_{[i]_{Bn-B+1}}]^T$
$y_{[j]_n}$	jth actuator signal at sample q
$\mathbf{y}_{[j]_n}$	$[y_{[j]_{Bn}} \quad y_{[j]_{Bn-1}} \quad \cdots \quad y_{[j]_{Bn-B+1}}]^T$
$e_{[k]_q}$	kth microphone signal at sample q
$\mathbf{e}_{[k]_n}$	$[e_{[k]_{Bn}} \quad e_{[k]_{Bn-1}} \quad \cdots \quad e_{[k]_{Bn-B+1}}]^T$
$\mathbf{s}_{[jk]}$	M-length estimation of the secondary path that links the jth secondary source with the kth error sensor
$\mathbf{S}^p_{[jk]}$	FFT of size $2B$ of the pth partition of the acoustic path $\mathbf{s}_{[jk]}$
$\mathbf{w}_n$	Coefficients of the adaptive filter of length L during the nth block iteration
$\mathbf{W}^f_n$	FFT of size $2B$ of the fth partition of the coefficients of the adaptive filter $\mathbf{w}$ during the nth block iteration

Table 8.1. Notation of the description of the algorithms

with N nodes.

8.2.1 The FPBFxLMS for a single-channel node

Following the nomenclature of this thesis, in this section, samples are processed by blocks of size B. L is the length of the adaptive filter $\mathbf{w}$, and M is the length of the FIR filters that model the estimated secondary paths $\mathbf{s}$. If L and M are larger than B, both $\mathbf{w}$ and $\mathbf{s}$ have to be partitioned into F and P partitions, respectively. Thus, the super-index of the following notation denotes the number of partition, and the index between brackets denotes the block iteration. The notation in Table 8.1 is used to describe the algorithm. According to the notation, the adaptive filter output is calculated as follows

$$\mathbf{Y}_n = \sum_{f=1}^{F} \mathbf{W}_n^f \circ \mathbf{X}_{n-f+1}, \tag{8.1}$$

where $\mathbf{X}_n = \text{FFT}\{[\mathbf{x}_{n-1}^T \quad \mathbf{x}_n^T]^T\}$. Vector $\mathbf{W}_n^f$ is the FFT of size $2B$ of the fth partition of $\mathbf{w}$ at the nth block iteration, and $\circ$ denotes the element-wise product of two vectors. The valid samples of the adaptive filter output $\mathbf{y}_n$ are the last B samples of $\text{IFFT}\{\mathbf{Y}_n\}$.

The filter coefficients are updated in the frequency domain by calculating the DFT of the correlation between the reference signal filtered through the estimated secondary path, and the error signal. To this end, the following operations are performed

$$\mathbf{V}_n = \sum_{p=1}^{P} \mathbf{S}^p \circ \mathbf{X}_{n-p+1}, \tag{8.2}$$

$$\widetilde{\boldsymbol{\mu}}_n^f = \mathbf{E}_n \circ \mathbf{V}_{n-f+1}^*, \tag{8.3}$$

where

$$\mathbf{E}_n = \text{FFT}[\mathbf{0}_B^T \quad \mathbf{e}_n^T]^T. \tag{8.4}$$

The update of the coefficients of each partition of the adaptive filter at the nth block iteration is calculated as follows

$$\mathbf{W}_{n+1}^f = \mathbf{W}_n^f - \mu\text{FFT}\{[\boldsymbol{\phi}_n^{f^T} \quad \mathbf{0}_B^T]^T\}, \tag{8.5}$$

where μ is the step-size parameter, and the vector $\boldsymbol{\phi}_n^f$ corresponds to the first B samples of the 2B-IFFT of the partition $\widetilde{\boldsymbol{\mu}}^f[n]$

$$[\boldsymbol{\phi}_n^{f^T} \quad \bar{\boldsymbol{\phi}}_n^{f^T}]^T = \text{IFFT}\{\widetilde{\boldsymbol{\mu}}_n^f\}. \tag{8.6}$$

Equations (8.3)-(8.6) are performed for each partition (f=1,...,F).

8.2.2 The FPBFxLMS for a distributed ANC system

The proposed distributed ANC system is applied to a ring network of N single-channel nodes. Therefore, N error sensors and N secondary sources

has to be considered. In this context, there exists a global state of the network, which is defined by N adaptive filters, one of each node. The global network adaptive filter matrix, $\underline{\mathbf{W}}_n$, can be defined as

$$\underline{\mathbf{W}}_n = [\mathbf{W}_{[1]_n}, \mathbf{W}_{[2]_n}, \ldots, \mathbf{W}_{[N]_n}], \tag{8.7}$$

$$\mathbf{W}_{[k]_n} = [\mathbf{W}^1_{[k]_n}, \mathbf{W}^2_{[k]_n}, \ldots, \mathbf{W}^F_{[k]_n}], \tag{8.8}$$

where $\underline{\mathbf{W}}_n$ is a $[2B \times FN]$ matrix composed of the concatenation of the adaptive filter of each node at the nth block iteration. Matrix $\mathbf{W}_{[k]_n}$ of size $[2B \times F]$ is the adaptive filter of the kth node at the nth block iteration, and vector $\mathbf{W}^f_{[k]_n}$ of size $2B$, is the fth partition of the adaptive filter in frequency domain of the kth node at the nth block iteration.

The N nodes collaborate with each other by updating their part of $\underline{\mathbf{W}}$ and transferring $\underline{\mathbf{W}}$ to the next node. Therefore, each node contains its own global state of the network, that we will refer as $\underline{\mathbf{W}}_{[k]}$. As commented before, the collaboration between nodes is done in an incremental way. This means that the kth node update its $\underline{\mathbf{W}}_{[k]}$ and sends it to the $k+1$th node. Once all the nodes have finished the actualization of the filters, the global updated vector $\underline{\mathbf{W}}_{[N]}$ is disseminated to the rest of the nodes before the next iteration begins. This means $2(N-1)$ transfers of the global state of the network. Figure 8.2 illustrate this collaboration between nodes.

For calculating the output signal, each node takes its filters from the global filters when they are completely adapted. Hence, the kth node at the nth block iteration, uses $\underline{\mathbf{W}}_{[N]_{n(:,1+(k-1)F:kF)}}$ to calculate its output signal like in Eq. (8.1). Moreover, we define

$$\underline{\mathbf{V}}_{[k]_n} = [\mathbf{V}_{[1k]_n}, \mathbf{V}_{[2k]_n}, \ldots, \mathbf{V}_{[Nk]_n}], \tag{8.9}$$

where $\underline{\mathbf{V}}_{[k]_n}$ is a matrix of size $[2B \times FN]$ of the kth node at the nth block iteration. It is composed of the concatenation of the reference signal filtered through all the secondary paths that links the jth loudspeaker with the kth microphone ($\mathbf{S}_{[jk]}$ for $j = 1, \ldots, N$). Each node knows the estimation of the secondary paths that links all the loudspeakers of the other nodes with its sensor. This means that, for example, the second node knows vectors $\mathbf{S}_{[j2]}$ for $j = 1, \ldots, N$. Matrix $\mathbf{V}_{[jk]_n}$ of size $[2B \times F]$ is calculated as stated in Eq. (8.2) for each secondary path and each partition

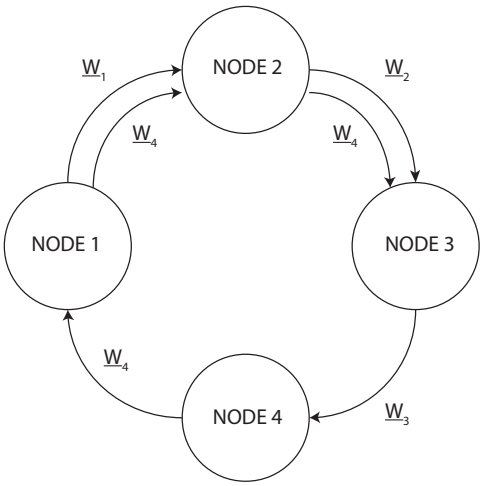


Figure 8.2. Example of an incremental collaboration in a ring network of 4 nodes.

$$\mathbf{V}_{[jk]_n} = [\mathbf{V}^1_{[jk]_n}, \mathbf{V}^2_{[jk]_n}, \ldots, \mathbf{V}^F_{[jk]_n}]. \tag{8.10}$$

Furthermore, each node takes its error signal from its sensor to form its error vector as in Eq. (8.4). Then, each node replicates its error vector FN times forming the matrix $\underline{\mathbf{E}}_{[k]}$ of size $[2B \times FN]$. Equation (8.3) is redefined for the kth node as

$$\underline{\widetilde{\boldsymbol{\mu}}}_{[k]_n} = \underline{\mathbf{E}}_{[k]_n} \circ \underline{\mathbf{V}}^*_{[k]_n}. \tag{8.11}$$

Matrix $\underline{\widetilde{\boldsymbol{\mu}}}_{[k]_n}$ of size $[2B \times FN]$ is used at the nth block iteration by each node to calculate the adaptation matrix of the node, $\underline{\boldsymbol{\Psi}}_{[k]_n}$, as

$$[\underline{\boldsymbol{\phi}}^T_{[k]_n} \quad \underline{\bar{\boldsymbol{\phi}}}^T_{[k]_n}]^T = \mathrm{IFFT}\{\underline{\widetilde{\boldsymbol{\mu}}}_{[k]_n}\}, \tag{8.12}$$

$$\underline{\boldsymbol{\Psi}}_{[k]_n} = \mathrm{FFT}\{[\underline{\boldsymbol{\phi}}^T_{[k]_n} \quad \mathbf{0}^T_{[B\times FN]}]^T\} \tag{8.13}$$

where $\underline{\boldsymbol{\phi}}_{[k]}$ and $\mathbf{0}_{[B\times FN]}$ are matrices of size $[B \times FN]$. Moreover, the operators FFT and IFFT perform direct and inverse fast fourier transforms of size 2B of each column of the matrices involved. Finally, each node calculates its own estimate of the global adaptive filters using the global state

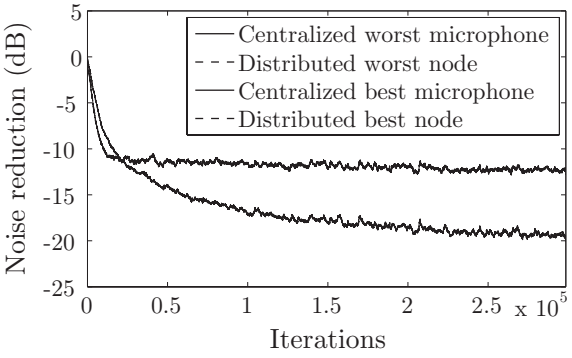


Figure 8.3. Noise reduction of the distributed system with 4 single-channel nodes and the centralized system with a 1:4:4 configuration represented for the best and worst microphone.

of the previous node, and its own adaptation matrix $\underline{\mathbf{\Psi}}_{[k]}$. The adaptation of the global adaptive filters at the kth node is performed at the nth block iteration as

$$\underline{\mathbf{W}}_{[k]_{n+1}} = \underline{\mathbf{W}}_{[k-1]_n} - \mu \underline{\mathbf{\Psi}}_{[k]_n}. \tag{8.14}$$

As commented before, once all the nodes have finished the actualization of the filters at the nth block iteration, the global updated vector $\underline{\mathbf{W}}_{[N]_n}$ is disseminated to the rest of the nodes for the *(n+1)*th iteration. Moreover, note that $\underline{\mathbf{W}}_{[0]_n} = \underline{\mathbf{W}}_{[N]_{n-1}}$.

8.3 Results

Some experiments were performed to validate the distributed ANC system. In a first stage, the noise reduction and the convergence performance of the distributed ANC system are evaluated and compared with the centralized ANC system. In a second stage, we evaluate and compare the computational complexity of both ANC systems.

8.3.1 Simulation results

In this section, some simulation results are presented to validate the performance of the FPBFxLMS algorithm in a distributed network with an incremental topology. The simulations have been carried out using real acoustic channels between microphones and loudspeakers sampled at 2 kHz. This channels have been measured inside the listening room described in section 2.3. We have considered a zero-mean Gaussian random noise with unit variance as the disturbance noise. Furthermore, we have considered a block-size of $B = 512$ samples, and, the same length of $L = M = 1024$ for both the adaptive filters and the estimated secondary paths. This means that two partitions are carried out. In order to evaluate the performance of the algorithm, we define the instantaneous Noise Reduction ratio at the kth node as

$$NR_{[k]_n} = 10 \log_{10} \left(\frac{e^2_{[k]_n}}{d^2_{[k]_n}} \right), \tag{8.15}$$

where $e_{[k]_n}$ and $d_{[k]_n}$ are the signals measured at the kth microphone with and without the ANC operation, respectively. Moreover, the power of these signals have been estimated using an exponential windowing.

First, we compare the noise reduction of a square centralized ANC system with a 1:4:4 configuration and a distributed ANC system with 4 single channel nodes. Figure 8.3 shows the noise reduction of both the centralized and the distributed implementations of the FPBFxLMS algorithm. Figure 8.3 illustrates the results for the microphone with best and worst performance in the centralized implementation, and the node with the best and worst performance in the distributed implementation. As expected, the distributed implementation has exactly the same results as the centralized implementation in terms of convergence speed and final residual noise.

8.3.2 Computational complexity

Table 8.2 compares the computational complexity in terms of multiplications, additions, and FFTs per iteration of the FPBFxLMS algorithm implemented for a centralized and a distributed ANC system. For the centralized implementation, we consider a multichannel ANC system with one disturbance noise and the same number (N) of microphones and loudspeakers (1:N:N configuration). For the distributed implementation, we

		Generic	N=1	N=4	N=8
(1)	MUX	$4LN + 4LN^2$	$8L$	$80L$	$288L$
	ADD	$LN + 3LN^2$	$4L$	$52L$	$200L$
	FFTs	$2 + 6N$	8	26	50
(2)	MUX	$2L + 6LN$	$8L$	$26L$	$50L$
	ADD	$L + 3LN$	$4L$	$13L$	$25L$
	FFTs	$4 + 4N$	8	20	36

Table 8.2. Total number of multiplications (MUX), additions (ADD), and FFTs per iteration of the implementation of the FPBFxLMS algorithm in (1) a centralized ANC system and (2) a distributed ANC system.

consider a network of N single-channel nodes. It is important to note that the complexity of the network as a whole is at least as high as the centralized algorithm. However, each node of the network could perform all the operations independently, except the last addition of the global adaptive filters calculated by the previous node (see Eq.(8.14)). Therefore, in Table 8.2, we have only computed the operations of one single-channel node. Moreover, since we use a value of $M = L$, and $B = L/2$ (two partitions) the computational complexity only depends on L and N.

First, the third column of table 8.2 shows the computational complexity of both algorithms related to values of L and N. Then, this computational complexity is particularized for $N = 1$, $N = 4$ and $N = 8$. As expected, when $N = 1$, both implementations make the same operations. This is because both the centralized and the distributed ANC system become a single-channel system. Moreover, for $N = 4$, we compare the operations of a centralized ANC system with a 1:4:4 configuration (16 channels) with the operations of a single-channel node of a network of 4 nodes. Finally, the same is done for $N = 8$. Results show that in a centralized ANC system, the computational complexity increases significantly with the number of channels. This fact constitute a bottleneck in massive multichannel ANC systems. Otherwise, the increase of computational complexity in a distributed ANC system is not so significant.

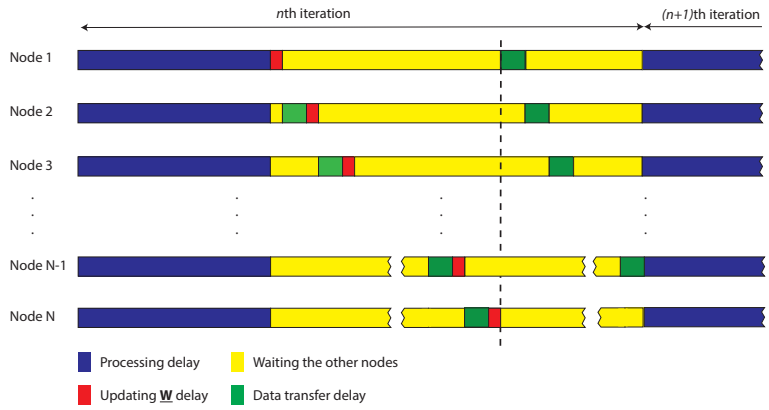


Figure 8.4. Timing diagram of the processes carried out by each node of the network at each iteration.

8.4 Considerations

This chapter has introduced the FPBFxLMS algorithm for a distributed ANC system. The simulations have demonstrated that the performance of the proposed distributed version of the algorithm can be the same as the centralized version. However, in order to successfully implement the distributed algorithm in a prototype with several nodes, some considerations must be taken into account.

Let us consider a network composed of N single-channel nodes. It has been commented that each of the N nodes can perform the algorithm independently at the same time, except Eq. (8.14). Equation (8.14) corresponds to the update of the global state of the network. Therefore, apparently, the distributed ANC system could perform the algorithm in the same time as a centralized single-channel ANC system. However, it is not true. In a distributed ANC system, the update of the global state of the network (Eq. (8.14)) has to be done sequentially. This means that the nth node needs $\underline{\mathbf{W}}$ of the *(n-1)*th node. Therefore, we also have to consider the processing time that spend each node in updating its own global state of the network ($\underline{\mathbf{W}}$), and the delay in transmitting $\underline{\mathbf{W}}$ to the following node. This fact, involves the transmission of $2L \times N$ coefficients between N nodes. As the transmission of data is done in an incremental mode, there are $(N-1)$ transmissions in each direction. Therefore, $2L \times N$ coefficients have to be transmitted $2(N-1)$ times in each block iteration.

Figure 8.4 depicts a timing diagram of the processes carried out by each node of the network at each iteration. Let us refer the processing time depicted in blue as t_{proc}, the time spent in updating $\underline{\mathbf{W}}$ as t_w, and, finally, the time spent in transferring $\underline{\mathbf{W}}$ as t_{trans}. Note that t_{proc} is the time spent by a node for processing all the algorithm except Eq. (8.14). Therefore, the time spent from the whole network in each iteration is calculated as $t_{dis} = t_{proc} + Nt_w + 2(N-1)t_{trans}$. On the other hand, as commented in the other chapters, the system has to satisfy the real-time condition, and, therefore, the processing time of the whole network, t_{dis}, has to be less than the buffering time, t_{buff}. In this sense, as t_{dis} depends on the time spent in the transfer of $\underline{\mathbf{W}}$, the data-transfer speed of the network has to be taken into account. Therefore, when the synchronization between nodes is feasible and offers enough bandwidth to transfer the $2L \times N$ coefficients $2(N-1)$ times, the proposed algorithm can perform as the centralized version.

As a future line of research, this kind of distributed ANC systems could be implemented over a prototype owning several GPUs, where, each GPU could handle the processing of each node of the network. The implementation of this prototype would be based on what it has been described in section 2.2.4. There, the data transfers over the network would be modeled by writing and reading data in the CPU memory, which would be common to all the GPU, and therefore, to all the nodes. A more realistic or practical implementation would consider each node composed of a microphone, a loudspeaker and a small portable device with an embedded processor such as the Jetson card [127] of NVIDIA. In this case, each Jetson card, which owns a GPU, would be the processor of each node. Moreover, the nodes could communicate through a wireless Local Area Network (LAN).

8.5 Conclusions

In contrast to the centralized ANC systems presented in the previous chapters, this chapter has presented a multichannel ANC system over a network of distributed acoustic nodes, which allows the processing of a centralized ANC system to be divided into several acoustic nodes. Therefore, this chapter has introduced a new formulation of the FPBFxLMS algorithm for its distributed implementation applied to an ANC system and using a

collaborative incremental strategy between the nodes in the network.

The results have demonstrated that the distributed implementation of the algorithm has the same performance as its centralized version. This results has been taken from a simulation without communication constraints in the network. Moreover, the computational complexity of the distributed algorithm has been studied and compared with the centralized version. Since in the distributed algorithm, each node can perform almost all the operations independently, the computational complexity is significantly reduced. However, when we consider the implementation of the proposed algorithm in a prototype, we have to analyze some aspects with regards to the communication capabilities between nodes in the network. In this sense, a brief overview has been given that conclude that the distributed ANC system will work in real time if the network has enough bandwidth to transfer the network information without violating the real-time condition.

The main contributions of this work were presented in the European Signal Processing Conference (EUSIPCO 2015).

9 Conclusion

9 Conclusion

The overall aim of this research has been to deepen into the audio signal processing algorithms that are based on the adaptive filtering, and to evaluate their potential when they are implemented on GPUs. The motivation of this research comes from the necessity of developing and accelerating audio applications that require high computational resources.

This chapter summarizes the findings and the main contributions of this research. First, Section 9.1 reviews the contents of this study, outlining the main conclusions that were extracted from each chapter. Recommendations for future research are discussed in Section 9.2. Additionally, the final sections contain a list of works published during the course of candidature for the P.h.D. degree, and the projects and stipends that have funded the presented work.

9.1 Main contributions

The overall contribution of this thesis has been the development and implementation of adaptive signal processing algorithms for multichannel spatial sound on GPUs. The use of GPUs in adaptive applications in which real-time interaction between microphones and loudspeakers is required, was

questioned. It was questioned due to the iterative transactions of data between the CPU and the GPU and vice versa. For this reason, first of all, the viability of the use of GPUs for real-time adaptive audio applications has been studied. To this end, a single channel identification system has been implemented using the FPBLMS algorithm. Among the wide variety of adaptive filtering applications, the channel identification application was chosen due to its simplicity. The LMS algorithm has been implemented in frequency domain and with a block-based processing in order to exploit the SIMD GPU architecture. For this purpose, each CUDA kernel has been designed seeking the most efficient implementation and avoiding GPU memory transfers. Moreover, some CUDA aspects as the number of threads per block, the distribution of these threads within the blocks, the number of blocks in the grid and the distribution of the blocks in the grid have been also analyzed. As a result of the good performance offered by the GPU implementation, it was stated that GPUs are suitable for audio adaptive systems working as the main processors. Moreover, it has demonstrated that the GPUs are capable of managing multiple channels without overloading the CPU.

In a second stage, the FPBLMS algorithm has been used with a filtered-x structure (FPBFxLMS) to implement a multichannel AE system on the GPU. The use of a filtered-x structure is necessary for the use of the LMS algorithm in adaptive equalization. Results have shown good performance of the adaptive equalization prototype even when there are interfering signals with low SIR levels. Moreover, 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 adaptive equalization system have been studied. It has been demonstrated that the GPU is a meaningful and versatile solution for massive multichannel adaptive equalization systems with even more than 1,000 channels processed in real time.

The adaptive application that has been mainly studied in this thesis is the active noise control. Three prototypes of multichannel feedforward local ANC systems have been developed using a GPU as the main processor. These implementations have been based on two different algorithms: the LMS algorithm with both the conventional and the modified filtered-x structure, and the NLMS-OCF algorithm. On the one hand, regarding the implementations based on the LMS algorithm, it has been shown that the

FPBMFxLMS algorithm converges faster than the FPBFxLMS algorithm, but exhibits a higher computational complexity, especially for massive multichannel systems. Nevertheless, it has been demonstrated that the use of a GPU platform can help to overcome the disadvantage of the modified scheme in a real-time ANC system. Moreover, the computational limits of both ANC systems have been analyzed in order to obtain massive multichannel systems that provide a large area of high attenuation by using more sensors. This work has demonstrated that the GPU is a meaningful and versatile solution for massive multichannel ANC systems, which can provide, slightly more than 600 processed channels in real time.

On the other hand, the study of the ANC application goes a step forward by developing a GPU implementation of a multichannel feedforward local ANC system based on the NLMS-OCF algorithm. To meet the hardware requirements, the algorithm has been developed in the frequency domain, working with blocks of data and partitioning the adaptive filters, resulting in the FPM-OCF algorithm. This algorithm increases the convergence performance of the FPBMFxLMS algorithm. However, it also increases the computational burden. It has been proved that both the improvement in convergence rate and the increase in computational complexity can be controlled and adjusted with the number of correction factors used in the execution of the algorithm, and, therefore, this value can be set depending on the performance requirements. As a conclusion, the GPU implementation of the FPM-OCF algorithm in a real-time multichannel ANC prototype is feasible and capable of processing hundreds of channels in real time, depending on the value of R.

Finally, it has been studied the implementation of an active noise control system over a network of distributed acoustic nodes. It has been demonstrated that the proposed algorithm has the same performance than the centralized version when there are no communication constraints in the network. Moreover, the computational complexity of the distributed algorithm has been studied and compared with the centralized version. Since in the distributed algorithm, each node can perform almost all the operations independently, the processing delay is significantly reduced. However, when considering the implementation of the proposed distributed algorithm in a prototype, it has been stated that some aspects regarding the communication capabilities between nodes in the network, has to be considered. In this sense, a brief overview has been given that conclude that the distributed

ANC system will work in real time if the network has enough bandwidth to transfer the network information without violating the real-time condition.

The important conclusion to point out in this dissertation is the usefulness of GPUs for developing versatile, scalable, and low-cost multichannel applications based on adaptive filtering. All the proposed GPU implementations offer excellent performances regarding the audio resources they can manage. Moreover, the fact of using GPUs for audio processing allows the CPU resources can be used for other tasks. Thus, this dissertation demonstrates that the use of the GPUs provides a good solution to build applications that require massive audio processing.

9.2 Further work

This thesis has laid within the convergence of the fields of signal processing and high performance computing. Concretely, in taking advantage of the parallel resources that offers the GPU for general processing. The use of GPU for general processing, instead of its traditional use in the processing of graphics, is a relatively new field of research. Moreover, it is a scientific and technological challenge that has been recently being addressed and should allow tackling potential applications that so far seemed unattainable for the consumer market. Therefore, the possibilities that it offers for future work are really vast.

Focusing on this thesis, the future research lines are grouped in three main blocks: development of new algorithms suitable for GPU implementations, the development of distributed prototypes based on acoustic sensor networks, and the development of GPU libraries.

In the first case, there is still a vast variety of algorithms that can be suitable for its GPU implementation. For example, the NLMS-OCF algorithm was implemented for an ANC system in chapter 7. As commented, the NLMS-OCF belongs to the APA family of algorithms. The choice of this algorithm rather than the others was, in part, because its implementation in frequency domain was easier than the affine projection algorithm. Therefore, the development of the AP algorithm in frequency domain for an ANC system, and its parallel implementation in a many-core device can be a useful future challenge in the active noise control field.

In the second case, chapter 8 has shown some simulation results of a distributed ANC system. In this area, the future work is straightforward, and would consist in the implementation of what was presented in chapter 8 in a real prototype. To this end, it is necessary to decide the composition of the nodes and to study how to communicate nodes in the network. In this sense, section 8.4 introduced the idea of implementing the distributed acoustic network with small portable nodes. The use of NVIDIA Jetson cards as the processors of the nodes, could be a feasible option to achieve small portable nodes with high computing capabilities.

Finally, as a consequence of the developed work throughout this dissertation, we have now a large number of computational kernels for different adaptive applications. As a future work, all these kernels could be packed in specific libraries for audio applications using GPUs. Libraries are valuable tools for specialists of a particular field, since it facilitates the development of scientific codes without knowing the GPU characteristics. These future libraries will also consider the new advances in the GPU architectures. Thus, it is expected that the performances of these libraries improve meaningfully the performances that are collected in this manuscript.

9.3 List of publications

First author publications during the candidature for the doctorate degree:

Journal Papers indexed in JCR

- Jorge Lorente, Miguel Ferrer, Maria de Diego and Alberto Gonzalez. "The Frequency Partitioned Block Modified Filtered-X NLMS with Orthogonal Correlation Factors for Multichannel Active Noise Control". Digital Signal Processing. Vol. 43, Pag. 47-58, 2015
- Jorge Lorente, Miguel Ferrer, Maria de Diego and Alberto Gonzalez. "GPU Implementation of Multichannel Adaptive Algorithms for Local Active Noise Control". IEEE Transactions on Audio, Speech, and Language Processing. Vol. 22, Num. 11, Pag. 1624-1635, 2014.

International Conference Papers

- Jorge Lorente, Miguel Ferrer, Maria de Diego and Alberto Gonzalez.

"Block-based Distributed Adaptive Filter for Active Noise Control in a Collaborative Network". European Signal Processing Conference (EUSIPCO), September 2015.

- Jorge Lorente, Miguel Ferrer, Maria de Diego and Alberto Gonzalez. "GPU Based Implementation of Multichannel Adaptive Room Equalization". International Conference on Acoustics, Speech and Signal Processing (ICASSP). Florence, Italy. May, 2014.

- Jorge Lorente, Miguel Ferrer, Maria de Diego, Jose Antonio Belloch and Alberto Gonzalez. "GPU Implementation of a Frequency-Domain Modified Filtered-X LMS Algorithm for Multichannel Local Active Noise Control". Audio Engineering Society Conference: 52nd International Conference: Sound Field Control-Engineering and Perception (52nd AES). Guildford, UK. September, 2013.

- Jorge Lorente, Jose Antonio Belloch, Miguel Ferrer, Alberto Gonzlez, Maria de Diego, Gema Piñero and Antonio Vidal. "Multichannel active noise control system using a GPU accelerator". Internoise. New York, USA. August, 2012.

- Jorge Lorente, Jose Antonio Belloch, Miguel Ferrer, Alberto Gonzalez, Maria de Diego and Antonio Vidal. "Active Noise Control Using Graphics Processing Units". International Congress on Sound and Vibration. Vilnius, Lithuania. July, 2012.

- Jorge Lorente, Jose Antonio Belloch, Miguel Ferrer, Alberto Gonzalez, Maria de Diego, Gema Piñero and Antonio Vidal. "Parallel Implementations of Beamforming Design and Filtering for Microphone Array Applications". 19th European Signal Processing Conference (EUSIPCO). Barcelona, Spain. August - September, 2011.

Peer-reviewed non-ISI Journal Papers

- J. Lorente, M. Ferrer, J. A. Belloch, G. Piñero, M. de Diego, A. Gonzalez, A. M. Vidal, "Real-time adaptive algorithms using GPUs", Waves, vol. 4, pp. 59-68, September 2012.

Spanish Conference Papers

- Jorge Lorente, Miguel Ferrer, Maria de Diego and Alberto Gonzalez. "Algoritmo NLMS-OCF en el dominio de la frecuencia aplicado a Control Activo de Ruido". 45 Congreso Español de Acústica - TECNIACSTICA 2014. Murcia, Spain. October, 2014.

On the other hand, there are two coauthored publications related with this thesis:

International Conference Papers

- J. A. Belloch, M. Ferrer, A. Gonzalez, J. Lorente, A. M. Vidal, "GPUbased WFS systems with mobile Virtual Sound Sources and Room Compensation", Proceedings of the 52nd Conference on Sound Field Control - Audio Engineering Society, Guildford, England, September 2013.

Peer-reviewed non-ISI Journal Papers

- 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-Ort, A. Remon and A. M. Vidal, "Application of Multi-core and GPU Architectures on Signal Processing: Case Studies", Waves, vol. 2, pp. 86-96, September 2010.

9.4 Institutional acknowledgements

This work has received financial support of the following projects and stipends:

- Project TEC2009-13741: Spatial audio systems based on massive parallel processing of multichannel acoustic signals with general purpose-graphics processing units (GP-GPU) and multicores. (Spanish Ministry of Science and Innovation)
- Project TEC2012-38142-C04-01: Distributed and Collaborative Sound Signal Processing: algorithms, tools and applications. (Spanish Ministry of Science and Innovation)

- Project PROMETEO 2009/2013: Computación de altas prestaciones sobre arquitecturas actuales en problemas de procesado de múltiples señales. (Generalitat Valenciana).

Bibliography

[1] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decision support systems*, vol. 15, no. 4, pp. 251–266, 1995.

[2] L. R. Rabiner and B. Gold, *Theory and application of digital signal processing.* Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975.

[3] K. Dowd, *High performance computing.* O'Reilly, 1993.

[4] B. Widrow and S. D. Stearns, *Adaptive signal processing.* Prentice-Hall, Englewood Cliffs, New York, 1985.

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

[6] B. Widrow, J. R. Glover, J. McCool, J. Kaunitz, C. Williams, R. Hearn, J. Zeidler, J. E. Dong, and R. Goodli, "Adaptive noise cancelling: Principles and applications," *Proceedings of the IEEE*, vol. 63, no. 12, pp. 1692–1716, December 1975.

[7] R. W. Lucky, "Techniques for adaptive equalization of digital communication systems," *ell System Technical Journal*, vol. 45, no. 2, pp. 255–286, 1966.

[8] T. Soderstrom and P. Stoica, *System identification*. London: Prentice hall, 1989.

[9] "Jacket, a GPU engine for MATLAB," *online at: http://www.accelereyes.com*.

[10] "Nvidia," *online at: http://www.nvidia.es/page/home.html*.

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

[12] "cuFFT, FFT library for CUDA," *online at: https://developer.nvidia.com/cuFFT*.

[13] "CULA, lapack library for CUDA," *online at: http://www.culatools.com*.

[14] "cuFFT, BLAS library for CUDA," *online at: https://developer.nvidia.com/cuBLAS*.

[15] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[16] B. Farhang-Boroujeny, *Adaptive Filters Theory and Applications*. John Wiley & Sons, Inc., New York, 1998.

[17] S. Haykin, *Adaptive filter theory*. Prentice Hall, 1986.

[18] C. F. N. Cowan and P. M. Grant, *Adaptive Filters*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[19] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *Proceedings of WESCON Convention*, 1960, pp. 96–140.

[20] K. J. Âström and P. Eykhoff, "System identificationa survey," *Automatica 7.2*, pp. 123–162, 1971.

[21] S. U. Qureshi, "Adaptive equalization," *Proceedings of the IEEE*, vol. 73, no. 9, pp. 1349–1387, 1985.

[22] P. Leug, *Process of silencing sound oscillations*. U.S Patent 2043416, 1936.

[23] H. F. Olsen and G. E. May, "Electronic sound absorber," *Journal of the Acoustical Society of America*, vol. 25, pp. 1130–1136, 1953.

[24] L. J. Eriksson, "Recursive algorithms for active noise control," *International Symposium of Active Control of Sound and Vibration*, pp. 137–146, 1991.

[25] S. M. Kuo and D. R. Morgan, "Active noise control: a tutorial review," *Proceedings of the IEEE*, vol. 87, no. 6, pp. 943–973, 1999.

[26] P. A. Nelson and S. J. Elliot, *Active control of sound.* Imperial College Press, New York, 1992.

[27] J. I. Nagumo and A. Noda, "Active noise control: a tutorial review," *Automatic Control, IEEE Transactions on*, vol. 12, no. 3, pp. 282–287, 1967.

[28] A. E. Albert and L. A. Gardner, *Stochastic approximation and nonlinear regression.* MIT Press, 1967.

[29] E. R. Ferrara, "Fast implementations of LMS adaptive filters," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 474–475, 1980.

[30] J. Lee and C. K. Un, "Performance analysis of frequency-domain block LMS adaptive digital filters," *Circuits and Systems, IEEE Transactions on*, vol. 36, no. 2, pp. 173–189, 1989.

[31] B. Farhang-Boroujeny and K. S. Chan, "Analysis of the frequency-domain block LMS algorithm," *Signal Processing, IEEE Transactions on*, vol. 48, no. 8, pp. 2332–2342, 2000.

[32] D. R. Morgan, "An analysis of multiple correlation cancellation loops with a filter in the auxiliary path," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 454–467, 1980.

[33] W. G. Poole, L. and R. Cutter, "The implementation of digital filters using a modified widrow-hoff algorithm for the adaptive cancellation of acoustic noise." *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, pp. 215–218, 1984.

[34] L. Sujbert, "A new filtered LMS algorithm for active noise control," *Proc. of the Active'99-The International EAA Symposium on Active Control of Sound and Vibration*, pp. 1101–1110, 1999.

[35] S. J. Elliott and J. G. Cook, "A preconditioned LMS algorithm for rapid adaptation of feedforward controllers," *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, pp. 845–848, 2000.

[36] M. Rupp and R. Frenzel, "Analysis of LMS and NLMS algorithms with delayed coefficient update under the presence of spherically invariant processes," *Signal Processing, IEEE Transactions on*, vol. 42, no. 3, pp. 668–672, 1994.

[37] A. T. Laichi, F. and W. Steenaart, "Effect of delay on the performance of the leaky LMS adaptive algorithm," *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, pp. 515–518, 1993.

[38] L. J. Eriksson, "Development of the filtered-u algorithm for active noise control," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 257–265, 1991.

[39] A. M. C. Eriksson, L. J. and R. Greiner, "The selection and application of an IIR adaptive filter for use in active sound attenuation," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 35, no. 4, pp. 433–437, 1987.

[40] D. R. Morgan and J. C. Thi, "A delayless subband adaptive filter architecture," *Signal Processing, IEEE Transactions on*, vol. 43, no. 8, pp. 1819–1830, 1995.

[41] E. Bjarnason, "Active noise cancellation using a modified form of the filtered-x LMS algorithm," in *Proceedings of the 6th European Signal Processing Conference*, vol. 2, 1992, pp. 1053–1056.

[42] H. C. Shin and A. H. Sayed, "Mean-square performance of a family of affine projection algorithms," *Signal Processing, IEEE Transactions on*, vol. 52, no. 1, pp. 90–102, 2004.

[43] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," *Electronics Communications*, vol. 67-A, pp. 19–27, 1984.

[44] M. Rupp, "A family of adaptive filter algorithms with decorrelating properties," *Signal Processing, IEEE Transactions on*, vol. 46, no. 3, pp. 771–775, 1998.

[45] S. G. Sankaran and A. Beex, "Convergence behavior of affine projection algorithms," *Signal Processing, IEEE Transactions on*, vol. 48, no. 4, pp. 1086–1096, 2000.

[46] S. A. H. Shin, H. C. and W. J. Song, "Variable step-size NLMS and affine projection algorithms," *IEEE signal processing letters*, vol. 11, no. 2, pp. 132–135, 2004.

[47] S. L. Gay, *The fast affine projection algorithm.* Springer, 2000.

[48] S. L. Gay and J. Benesty, *Acoustic signal processing for telecommunication.* Springer, 2000.

[49] S. G. Kratzer and D. R. Morgan, "The partial-rank algorithm for adaptive beamforming," in *29th Annual Technical Symposium.* International Society for Optics and Photonics, 1986, pp. 9–14.

[50] S. G. Sankaran and A. Beex, "Normalized LMS algorithm with orthogonal correction factors," in *Signals, Systems & Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on*, vol. 2. IEEE, 1997, pp. 1670–1673.

[51] M. D. McCool, "Signal processing and general-purpose computing and GPUs," *IEEE Signal Processing Magazine*, vol. 24, pp. 109–114, 2007.

[52] "Cg language," *online at: https://developer.nvidia.com/cg-toolkit.*

[53] S. Whalen, "Audio and the graphics processing unit," *uthor report, University of California Davis*, 2005.

[54] K. Matsuyama, T. Fujimoto, and N. Chiba, "Real-time sound generation of spark discharge," *IEEE 15th Pacific Conference on Computer Graphics and Applications*, pp. 423–426, 2007.

[55] E. Gallo and N. Tsingos, "Efficient 3d audio processing on the GPU," *ACM Workshop on General Purpose Computing on Graphics Processors*, 2004.

[56] N. Rober, U. Kaminski, and M. Masuch, "Waveguide-based room acoustics through graphics hardware," in *International Computer Music Conference*, New Orleans, USA, 2006, pp. 6–11.

[57] "openGL," *online at: http://www.opengl.org*.

[58] Y. K. Chen, C. Chakrabarti, S. Bhattacharyya, and B. Bougard, "Signal processing on platforms with multiples cores: Part 1 - overview and metodogies," *IEEE Signal Processing Magazine*, vol. 26, no. 6, 2009.

[59] ——, "Signal processing on platforms with multiples cores: Part 2 - applications and design," *IEEE Signal Processing Magazine*, vol. 27, no. 2, 2010.

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

[61] F. Trebien and M. Oliveira, "Realistic real-time sound re-synthesis and processing for interactive virtual worlds," *The Visual Computer*, pp. 469–477, 2009.

[62] B. Cowan and B. Kapralos, "Spatial sound for video games and virtual environments utilizing real-time GPU-based convolution," in *In Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, Ontario, Canada, 2008, pp. 166–172.

[63] ——, "GPU-based one-dimensional convolution for real-time spatial sound generation," *The Journal of the Canadian Game Studies Association*, vol. 14, no. 3, pp. 183–196, 2010.

[64] N. Raghuvanshi, N. Galoppo, and M. C. Lin, "Accelerated wave-based acoustics simulation," in *In Proceedings of the 2008 ACM symposium on Solid and physical modeling*, 2008, pp. 91–102.

[65] C. J. Webb and S. Bilbao, "Computing room acoustics with CUDA-3D FDTD schemes with boundary losses and viscosity," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, 2011, pp. 317–320.

[66] ——, "Virtual room acoustics: A comparison of techniques for computing 3D-FDTD schemes using CUDA," in *Proceedings of the 130th AES Convention*, London, UK, 2011.

[67] A. J. Berkhout, D. de Vries, and P. Vogel, "Acoustic control by wave field synthesis," *The Journal of the Acoustical Society of America*, vol. 93, no. 5, pp. 2764–2778, 1993.

[68] B. D. Van Veen and K. M. Buckley, "Beamforming: A versatile approach to spatial filtering," *IEEE assp magazine*, vol. 5, no. 2, pp. 4–24, 1988.

[69] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, "Multi-core platforms for beamforming and wave field synthesis," *IEEE Transactions on Multimedia*, vol. 99, pp. 235–245, December 2010.

[70] N. Tsingos and W. Jiang, "Using programmable graphics hardware for acoustics and audio rendering," *Journal Audio Eng. Soc*, vol. 59, pp. 628–648, 2011.

[71] 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 *Proceedings of the TG Conference on Speech Communication*, 2012, pp. 1–4.

[72] M. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. 21, pp. 948–960, 1972.

[73] "Features of the Nvidia CUDA capabilities," *online at: https://developer.nvidia.com/cuda-gpus.*

[74] "openMP," *online at: http://www.openmp.org.*

[75] S. Cook, *A Developer's Guide to Parallel Computing with GPUs.* Morgan Kaufmann, 2013.

[76] "Audio and Communications Signal Processing Group (GTAC)," *online at: http://www.gtac.upv.es.*

[77] R. Burdisso, J. Vipperman, and C. Fuller, "Causality analysis of feedforward-controlled systems with broadband inputs," *The Journal of the Acoustical Society of America*, vol. 94, no. 1, pp. 234–242, 1993.

[78] G. H. Golub and C. F. Van Loan, *Matrix computations*. Johns Hopkins University Press, 1996, vol. 1.

[79] M. Gevers, "A personal view of the development of system identification: A 30-year journey through an exciting field," *Control Systems, IEEE*, vol. 26, no. 6, pp. 93–105, 2006.

[80] K.-J. Åström and T. Bohlin, "Numerical identification of linear dynamic systems from normal operating records," in *Theory of self-adaptive control systems*, 1966, pp. 96–111.

[81] G. E. Box and G. M. Jenkins, *Time series analysis: forecasting and control, revised ed.* Holden-Day, 1976.

[82] J. 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, pp. 301–315, 1992.

[83] P. Sommen, "Partitioned frequency domain adaptive filters," in *Proceedings of the 23rd Asilomar Conference on Signals, Systems and Computers*, vol. 2, 1989, pp. 677–681.

[84] J. Vanderkooy, "Aspects of mls measuring systems," *Journal of the Audio Engineering Society*, vol. 42, no. 4, pp. 219–231, April 1994.

[85] D. Rife and J. Vanderkooy, "Transfer-function measurement with maximum-length sequences," *Journal of the Audio Engineering Society*, vol. 37, no. 6, pp. 419–444, June 1989.

[86] A. Farina, "Simultaneous measurement of impulse response and distortion with a swept-sine technique," in *Audio Engineering Society Convention 108*, February 2000.

[87] S. Müller and P. Massarani, "Transfer-function measurement with sweeps," *Journal of the Audio Engineering Society*, vol. 49, no. 6, pp. 443–471, 2001.

[88] A. Potchinkov, "Low-crest-factor multitone test signals for audio testing," *Journal of the Audio Engineering Society*, vol. 50, no. 9, pp. 681–694, 2002.

[89] A. González, P. Zuccarello, M. de Diego, and G. Pifero, "Simultaneous characterization of multichannel acoustic systems," *Journal of the Audio Engineering Society, To appear in*, pp. 26–42, 2003.

[90] J. Lorente, M. Ferrer, J. Belloch, G. P. nero, M. de Diego, A. Gonzalez, and A. Vidal, "Real-time adaptive algorithms using GPUs," *Waves*, vol. 4, pp. 59–68, 2012.

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

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] J. J. López, A. González, and L. Fuster, "Room compensation in wave field synthesis by means of multichannel inversion," in *Applications of Signal Processing to Audio and Acoustics, 2005. IEEE Workshop on*, 2005, pp. 146–149.

[97] B. Widrow, D. Shur, and S. Shaffer, "On adaptive inverse control," in *Proc. 15th ASILOMAR Conf. on Circuits, systems and computers*, 1981, pp. 185–189.

[98] B. Widrow and M. Bilello, "Adaptive inverse control," in *Intelligent Control, 1993., Proceedings of the 1993 IEEE International Symposium on*, 1993, pp. 1–6.

[99] 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.

[100] 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.

[101] 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.

[102] J. Lorente, M. Ferrer, M. de Diego, and A. Gonzalez, "GPU based implementation of multichannel adaptive room equalization," in *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Process. (ICASSP)*, Florence, 2014, pp. 7535–7539.

[103] J. Lorente, J. A. Belloch, M. Ferrer, A. Gonzalez, M. de Diego, G. Piñero, and A. Vidal, "Multichannel active noise control system using a GPU accelerator," in *Proceedings of the Inter-Noise conference*, New York, 2012, pp. 7768–7780.

[104] S. J. Elliot and P. A. Nelson, "Active noise control," *IEEE Signal Processing Magazine*, vol. 10, no. 4, pp. 12–35, October 1994.

[105] S. Elliott, P. Joseph, A. Bullmore, and P. Nelson, "Active cancellation at a point in a pure tone diffuse sound field," *Journal of sound and vibration*, vol. 120, no. 1, pp. 183–189, 1988.

[106] S. J. Elliott, I. M. Stothers, and P. A. Nelson, "A multiple error LMS algorithm and its application to the active control of sound and vibration," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 10, pp. 1423–1434, October 1987.

[107] S. M. Kuo and D. R. Morgan, *Active noise control, algorithms and DSP implementations*. John Wiley & Sons, Inc., New York, 1996.

[108] S. J. Elliott and P. A. Nelson, "The application of adaptive filtering to the active control of sound and vibration," *NASA STI/Recon Technical Report N*, vol. 86, p. 32628, 1985.

[109] D. R. Morgan, "An analysis of multiple correlation cancellation loops with a filter in the auxiliary path," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, no. 4, pp. 454–467, 1980.

[110] J. C. Burgess, "Active adaptive sound control in a duct: A computer simulation," *The Journal of the Acoustical Society of America*, vol. 70, no. 3, pp. 715–726, 1981.

[111] M. Pawełczyk, "Feedforward algorithms with simplified plant model for active noise control," *Journal of sound and vibration*, vol. 225, pp. 77–95, 2003.

[112] D. P. Das, G. Panda, and S. M. Kuo, "New block filtered-x LMS algorithms for active noise control systems," *IET Signal Processing*, vol. 1, no. 2, pp. 73–81, 2007.

[113] "Intel 64 and ia-32 architectures optimization reference manual," *online at: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf*.

[114] J. Lorente, M. Ferrer, M. de Diego, and A. Gonzalez, "GPU implementation of multichannel adaptive algorithms for local active noise control," *IEEE Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 22, no. 11, pp. 1624–1635, 2014.

[115] J. Lorente, A. Gonzalez, M. Ferrer, J. A. Belloch, M. de Diego, G. Piñero, and A. Vidal, "Active noise control using graphics processing units," in *Proceedings of the 19th International Congress on Sound and Vibration*, 2012, pp. 138–146.

[116] J. Lorente, M. Ferrer, M. de Diego, J. A. Belloch, and A. Gonzalez, "GPU implementation of a frequency-domain modified filtered-x LMS algorithm for multichannel local active noise control," in *Proceedings of the 52nd Audio Engineering Society International Conference*, Guilford, UK, Sep 2013.

[117] M. Ferrer, A. Gonzalez, M. de Diego, and G. Piñero, "Fast affine projection algorithms for filtered-x multichannel active noise control," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 8, pp. 1396–1408, 2008.

[118] S. G. Sankaran, "On ways to improve adaptive filter performance," Ph.D. Thesis, Virginia Polytechnic Institute and State University, 1999.

[119] J. Lorente, M. Ferrer, M. de Diego, and A. Gonzalez, "The frequency partitioned block modified filtered-x NLMS with orthogonal correction factors for multichannel active noise control," *Digital Signal Processing*, vol. 43, pp. 47–58, 2015.

[120] W. P. Chen, J. C. Hou, and L. Sha, "Dynamic clustering for acoustic target tracking in wireless sensor networks," *Mobile Computing, IEEE Trans. on*, vol. 3, no. 3, pp. 258–271, 2004.

[121] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications magazine, IEEE*, vol. 40, pp. 102–114, 2002.

[122] D. Puccinelli and M. Haenggi, "Wireless sensor networks: applications and challenges of ubiquitous sensing," *Circuits and Systems Magazine, IEEE*, vol. 5, no. 3, pp. 19–31, 2005.

[123] A. Tanenbaum and M. Van Steen, *Distributed systems*. Pearson Prentice Hall, 2007.

[124] A. Bertrand, "Applications and trends in wireless acoustic sensor networks: a signal processing perspective," *Communications and Vehicular Technology (SCVT), 18th IEEE Symposium on*, pp. 1–6, 2011.

[125] C. G. Lopes and A. H. Sayed, "Incremental adaptive strategies over distributed networks," *Signal Processing, IEEE Trans. on*, vol. 55, pp. 4064–4077, 2007.

[126] M. Ferrer, M. de Diego, G. Piñero, and A. Gonzalez, "Active noise control over adaptive distributed networks," *Signal Processing*, vol. 107, pp. 82–95, 2015.

[127] "Mobile GPU: Jetson." *online at: http://developer.download.nvidia.com/embedded/jetson/TK1/docs.*