MULTI-CHANNEL IIR FILTERING OF AUDIO SIGNALS USING A GPU

Jose A. Belloch[⋆†] *Balázs Bank*[‡] *Lauri Savioja*[⋆] *Alberto Gonzalez*[⋆†] *Vesa Välimäki*[†]

[⋆†] iTeAm, Universitat Politècnica de València, Valencia, Spain
[‡] Budapest University of Technology and Economics, Budapest, Hungary
[⋆] Dept. of Media Technology, Aalto University, Espoo, Finland
[†] Dept. of Signal Processing and Acoustics, Aalto University, Espoo, Finland

ABSTRACT

In the audio signal processing field, multiple IIR filters are required in many applications. As an example, equalizing a Wave Field Synthesis system requires massive filter processing in real time. Graphics Processing Units (GPUs) are well known for their potential in highly parallel data processing. Up to now, the use of the GPUs for implementing IIR filters has not been clearly tackled in audio processing because of its feedback loop that prevents its total parallelization. However, using the Parallel form of IIR filters, this feedback is reduced, since every single sample is computed in a parallel way. This paper analyzes the performance of multiple IIR filters using GPUs and compares it with a powerful multi-core computer. The proposed GPU implementation can run up to 1256 concurrent IIR filters of order 256th in real time, which means 321,536 total filter order, with a latency time of 0.72 ms (sampling frequency of 44.1 kHz). This demonstrates that GPUs are well suited for computing massive IIR filtering.

Index Terms— Audio systems, IIR filters, parallel architectures, parallel processing

1. INTRODUCTION

Modeling or equalizing an acoustic or electro-acoustic transfer function by digital filters is a typical task in audio signal processing. By taking into account the properties of the human hearing, significant savings can be achieved in the required computational power at a given sound quality. As an example, the frequency resolution of the human auditory system has led to the development of special filter design methodologies with a logarithmic frequency resolution, as opposed to the linear frequency resolution of traditional FIR and IIR filters. These techniques include frequency warping [1], Kautz filters [2], or fixed-pole parallel filters [3]. Typically, the required filter order is reduced by a factor of 5 compared to traditional IIR filters (e.g., designed by the Steiglitz–McBride method [4]) with all the above techniques. This advantage is slightly reduced in the case of warped and Kautz filters, because they are implemented by special filter structures, but not for fixed-pole parallel filters, since they are simply implemented as a set of second-order sections. [1]

This work was conducted in fall 2013 when J. A. Belloch was visiting the Aalto University Department of Signal Processing and Acoustics. Thanks to the EEBB-I-13-06059, TEC2012-38142-C04-01, GVA/2013/134 and to the PROMETEO/2009/013 projects for funding. The work of Balázs Bank was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

[1] A strong contender of these logarithmic filter design methods in terms of computational complexity is FFT-based partitioned convolution [5]. However, the performance comparison of partitioned convolution to logarithmic filter design methods is out of the scope of the paper.

One application that is specially important in the context of multichannel acoustic signal processing using IIR filters is the equalization of a Wave Field Synthesis (WFS) system. WFS systems require high computational capacity since they involve multiple loudspeakers, such as the WFS system at the Universitat Politècnica de València (UPV) (shown in [6]) that has 96 loudspeakers, or the IOSONO WFS system (shown in [7]) that has 120 loudspeakers. Equalizing a WFS system requires such a massive amount of filtering that even when using parallel filters, a significant amount of CPU time is taken for filtering, and in some cases, real-time operation is not possible even in modern multi-core computers.

Recently, Graphics Processing Units (GPUs) have gained a strong interest in the audio community due to their massive computational power. Applications include room acoustics modelling [8, 9, 10, 11], computer-music synthesis using additive synthesis [12, 13], sliding phase vocoder [14], beamforming [15], audio rendering [16, 17, 18], and adaptive filtering [19, 20, 21] among others. There are also GPU-based implementations of WFS systems such as [22, 23], but they do not perform any equalization. Interestingly, implementing IIR filters on GPUs has not been investigated much in the audio field. A preliminary approach was presented in [24], where only the non-recursive operations are parallelized. Since fixed-pole parallel filters are among the most efficient methodologies for audio, it is a natural choice to implement them in GPUs to allow even faster filtering. Besides their computational efficiency, a further motivation for using parallel filters is their full potential for code parallelization.

Therefore, this paper presents a multichannel GPU-based implementation of parallel IIR filters that can be used for equalizing a WFS system and compares its computational performance with the performance of a powerful multi-core computer.

2. FIXED-POLE PARALLEL FILTERS

Traditionally, the parallel second-order form of digital filters has been used because of its lower sensitivity to coefficient quantization and better quantization noise performance compared to direct form IIR filters [25]. In these applications, first a direct form IIR filter is designed and then factored to a parallel form using the partial fraction expansion.

In fixed-pole parallel filters, the filter is designed directly in the second-order form by first setting the poles to predetermined positions. The advantage of fixing the poles is that now the filter design reduces to a linear-in-parameter problem which has a unique solution. Nevertheless, the most important property of parallel filters is that the pole frequencies allow a direct control of the frequency resolution: the more poles are placed in a specific frequency range, the higher resolution is obtained. For example, placing the

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

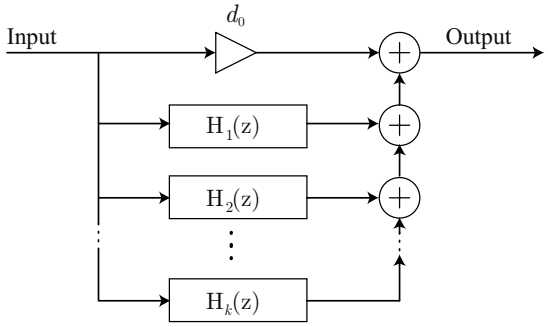

Fig. 1. Structure of the parallel second-order filter.

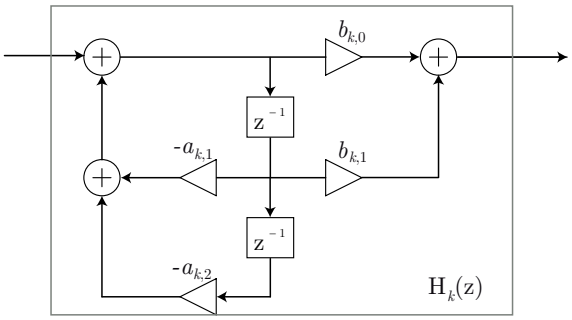

Fig. 2. Structure of the second-order section used in Fig. 1

poles according to a logarithmic frequency scale results in a logarithmic frequency resolution, and the modeled response resembles the fractional-octave smoothed version of the target [26]. For a thorough comparison of pole positioning methods see [27].

The general form of the parallel filter consists of a parallel set of second-order sections and an optional FIR filter path [28]. In this paper the FIR part is reduced to a single signal path, and the following form is used:

$$H(z^{-1}) = \sum_{k=1}^{K} \frac{b_{k,0} + b_{k,1} z^{-1}}{1 + a_{k,1} z^{-1} + a_{k,2} z^{-2}} + d_0, \qquad (1)$$

where K is the number of second-order sections. The filter structure and the second-order section are depicted in Fig. 1 and Fig. 2, respectively.

2.1. Filter design

We can assume that the poles of the parallel filter p_k are known (e.g., set to a logarithmic scale). Then the denominator coefficients are determined by the poles ($a_{k,1} = p_k + \overline{p}_k$ and $a_{k,2} = |p_k|^2$), and the filter design problem becomes linear in its free parameters (weights) $b_{k,0}$, $b_{k,1}$ and d_0.

Using the substitution $z^{-1} = e^{-j\vartheta_n}$ in (1) and writing it in matrix form for a finite set of ϑ_n angular frequencies yields [28]

$$\mathbf{h} = \mathbf{M}\mathbf{p}, \qquad (2)$$

where $\mathbf{p} = [b_{1,0}, b_{1,1}, \dots b_{k,0}, b_{k,1}, d_0]^T$ is a column vector composed of the free parameters. The first column of the modeling matrix $\mathbf{M}$ contains the all-pole transfer function of the first section $1/(1 + a_{1,1}e^{-j\vartheta_n} + a_{1,2}e^{-j2\vartheta_n})$ for the ϑ_n angular frequencies, and the second column contains its delayed version version $e^{-j\vartheta_n}/(1 + a_{1,1}e^{-j\vartheta_n} + a_{1,2}e^{-j2\vartheta_n})$ for all ϑ_n. The third and fourth columns are the all-pole transfer functions for the second section $1/(1 + a_{2,1}e^{-j\vartheta_n} + a_{2,2}e^{-j2\vartheta_n})$ and its delayed version $e^{-j\vartheta_n}/(1 + a_{2,1}e^{-j\vartheta_n} + a_{2,2}e^{-j2\vartheta_n})$ for all ϑ_n. The remaining part of matrix $\mathbf{M}$ is constructed similarly, except the last column, which belongs to the constant gain path, and it is 1 for all ϑ_n. Finally, $\mathbf{h} = [H(\vartheta_1) \dots H(\vartheta_N)]^T$ is a column vector composed of the resulting frequency response.

The optimal parameters $\mathbf{p}_{\text{opt}}$ in the mean squares sense are found by the well-known least-squares (LS) solution

$$\mathbf{p}_{\text{opt}} = (\mathbf{M}^H\mathbf{M})^{-1}\mathbf{M}^H\mathbf{h}_{\text{t}}, \qquad (3)$$

where $\mathbf{M}^H$ is the conjugate transpose of $\mathbf{M}$, and $\mathbf{h}_{\text{t}}$ is the target frequency response. Note that (3) assumes a filter specification $H_{\text{t}}(\vartheta_n)$ given for the full frequency range $\vartheta_n \in [-\pi, \pi]$. Matlab code for parallel filter design can be downloaded from `http://www.mit.bme.hu/~bank/parfilt`.

3. PARALLEL IMPLEMENTATIONS

This section describes how the parallelization is organized inside the GPU and multi-core architectures.

3.1. GPU and CUDA

The appearance of CUDA [29] has allowed to use the GPUs for applications beyond graphics rendering. GPUs have the potential of highly parallel data processing. The recent Nvidia GPU Kepler architecture [30] is composed of multiple Stream Multiprocessors (SMX), where each SMX consists of 192 pipelined cores. A GPU device has a large amount of off-chip device memory (*global-memory*) and a fast on-chip memory (*shared-memory*). The code to be executed on the GPU concurrently by multiple elementary processes, called *threads*, is written on a *kernel* function. The threads are grouped in Thread Blocks (TB). Before launching the GPU code, the programmer must define the number of TBs and its size, i.e., the number of threads. This is important, since only the threads that belong to the same TB can share data through *shared-memory*.

The total number of TBs launched in a kernel can exceed the number of SMX. At runtime, the kernel distributes all the TBs among SMXs. Each SMX can host up to 16 TBs or up to 1024 threads in total (summing up all the threads of all the TBs). If the number of TBs exceed the resources of the GPU, these TBs wait until other TBs finish their computation in order to be later hosted. Each SMX manages all the TBs jointly, and executes all the threads (regardless of the TB they belong to) in groups of 32 parallel threads called *warps* that are scheduled by *warp schedulers* for execution.

It is important to have multiple *warps* in a SMX, since it allows to hide latency. This means that, in case all threads of a *warp* must carry out a memory access that can last several clock cycles, the *warp schedulers* select other *warp* that is ready to execute in order to hide latency. The *warp schedulers* are responsible to switch among different *warps* in order to try full utilization of SMXs. Thus, it is recommended to have multiple TBs in an SMX. To this end, and following the recommendations of [29], we assign to each TB 128, 256 or 512 threads. If we use 1024-size TBs, only one TB would fit in a SMX. We will tackle different TB sizes in order to seek the best performance.

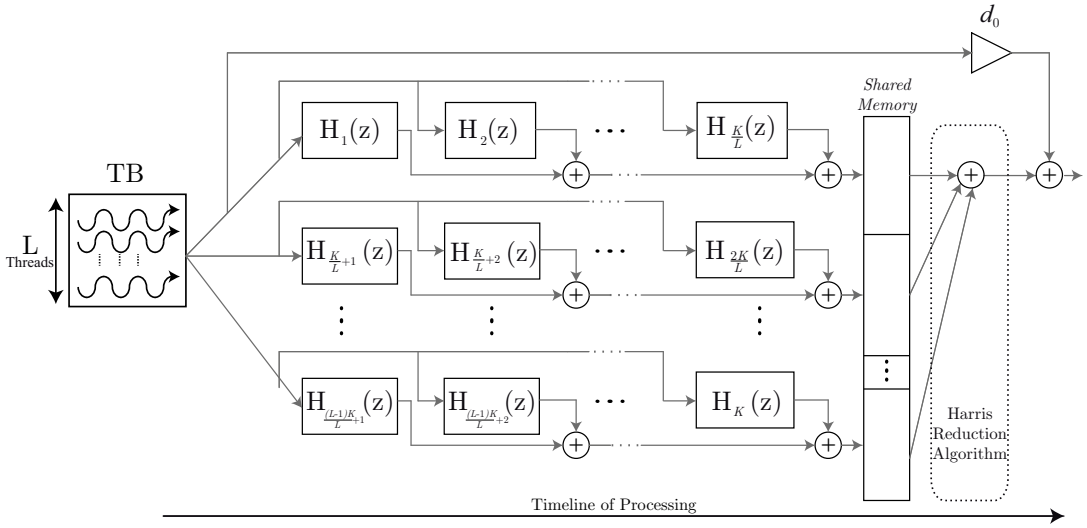

Fig. 3. GPU-based Parallel Implementation of one IIR filter processing.

3.2. GPU-based parallel implementation

If we want to equalize a WFS system composed of N loudspeakers, we need to carry out N IIR filter processes concurrently. Thus, we launch N TBs to run the CUDA kernel. Each TB of L threads ($L \in \{128, 256, 512\}$) corresponds to one IIR filter that has K second-order sections. A thread inside a TB computes $\frac{K}{L}$ sections, and stores its result in the *shared-memory*. Then, a synchronization barrier is set in order to wait that all the threads have finished. After that, the reduction algorithm described by Harris [31] is implemented. It consists in summing up in a parallel way all the values of a vector that is stored in the *shared-memory*. Finally, the FIR coefficient d_0 is executed at the end by only one of the threads of the TB. Figure 3 illustrates the described operations for one parallel IIR filter process.

3.3. Multicore-based parallel implementation

In order to assess the computational performance achieved by the GPU implementation, a comparison with a powerful multicore computer is required. To this end, we implement our algorithm using *openMP* [32]. This programming framework allows us to parallelize our algorithm using all the cores that a computer owns. The multicore computers are more suited to task-based parallelism, instead of the fine-grain parallelism, which can be easily managed by a GPU. The advantage of a multicore implementation is that data transferring from/to GPU to/from CPU is not needed.

The serial algorithm to carry out N IIR filters is composed by three nested loops: number of filters, number of samples per filter, and number of second-order sections. By using *openMP*, we implement our algorithm by distributing the iterations of the most external loop among all the cores, since each filter process is independently performed. In case that our multicore computer is composed of P processors, each processor is responsible for carrying out $\frac{N}{P}$ filters.

Comparing to the GPU-based implementation, one CPU processor computes at least the same operations as a TB. However, one TB only computes one filter process, while one CPU processor could compute more than one. GPU-based parallelization presents two levels: concurrency in computing multiple filter processes, and concurrency in computing multiple second-order sections inside one filter. In contrast, the CPU-based parallelization presents only concurrency in multiple filter processes.

4. RESULTS

We test our GPU-based implementation on an Nvidia Tesla K20c that is based on the Kepler architecture and is composed of 13 SMXs, and our multicore-based implementation on a computer composed of two SMPs (*Symmetric Multi-Processing*) Intel Xeon CPU X5680 at 3.33 GHz, which is a hexacore. Thus, our multicore computer is composed of 12 cores.

We use a standard audio card at the laboratory. The audio card uses the ASIO (Audio Stream Input/Output) driver to communicate with the CPU and provides 32, 64, and 128 samples per channel every 0.72 ms, 1.45 ms, and 2.90 ms, respectively (sample frequency f_s=44100 Hz), which we call buffer times t_{buff}. Assuming that our WFS system requires to equalize N loudspeakers, we define t_{proc} as the processing time since the N input-data buffers are available till the N output-data buffers are totally processed. Data transfer times in the case of GPU-based implementation are included in t_{proc}. The equalization of a WFS system works in real-time as long as $t_{\text{proc}} < t_{\text{buff}}$.

Figure 4 shows the results when a system is executed using a buffer size of 32 samples. Computational performance has been assessed by assuming that all filters are composed of 128 second-order sections for the first example, and with filters composed of 1024 second-order sections for the second example. We execute the system by increasing N gradually and by measuring each time t_{proc}. Note that the maximum number of filters that can be executed in real time are marked with a circle $\circ$ in Fig. 4, and their values are shown in the legend of the figure for all cases. The proposed GPU-based implementation can run 1256 filters in real time with 0.72 ms latency when the filters are composed of 128 second-order sections. In case of 1024 second-order sections, 272 equalization filters can be executed.

Very similar results are obtained if we increase the buffer size

Table 1. Maximum number of real-time IIR filters for the GPU implementation using different thread block sizes L and buffer sizes of 32 and 64 samples. The best results are bolded.

SIZE	$L = 128$	$L = 256$	$L = 512$	$L = 1024$
32	192	**272**	168	120
64	208	**280**	176	128

to 64 and 128 samples, as can be appreciated in Fig. 5. The number of filter processes that can be achieved in real time increases slightly as the buffer size does, in spite that transfer times between GPU and CPU increase. This occurs because transfer time is still not significant compared to the parallelization resources that GPUs offer. For filters composed of 128 second-order sections, GPU outperforms in 2.75, 2.6, and 2.55 times the CPU, for buffer sizes of 32, 64 and 128 samples, respectively. Otherwise, for filters composed of 1024 second-order sections, GPU outperforms in 5.23, 4.8 and 4.66 times the CPU for the same previous buffer sizes. Thus, computing filters composed of 1024 second-order is more efficient on GPU, since more computational resources of the GPU are utilized. Note that the speed-up GPU/CPU decreases slightly as long as the buffer size increases.

The CPU-based implementation runs on a powerful computer composed of 12 cores, which is a fair comparison with the GPU. In all cases, the GPU-based implementation outperforms the multicore-based implementation, which indicates that GPUs are well suited for performing massive IIR filter processes, even more when the sample buffer sizes are very short.

Regarding GPU-based computational aspects, filters composed of 128 second-order sections require to launch a CUDA *kernel* composed of TBs with L=128, where each thread performs one section. However, filters composed of 1024 second-order sections first require to test different values for L (see subsec. 3.1). Table 1 shows the maximum number of equalization filters that can be executed in real time for different values of L. It can be noticed that using the TB size of 256 threads gives the best performance.

As another performance measure, taking into the number of loudspeakers of the WFS systems referenced in sec. 1, we have tested the maximum number of second-order sections that an equalizer filter of these systems can have by using the GPU. Regarding the WFS system at the Universitat Politècnica de València (UPV), we have achieved to equalize the 96 loudspeakers using filters composed of up to 2048 second-order sections under real-time conditions. Otherwise, as IOSONO WFS system is composed of 120 loudspeakers, the massive equalization can be carried out also under real-time conditions if we use up to 1536 second-order sections. Thus, GPUs allow us to use high filter orders for equalizing large-scale WFS systems.

An additional advantage of the GPU-based implementation is that the GPU can be used as a co-processor where all audio processing is being carried out, while the CPU could be used for other tasks at the same time.

5. CONCLUSION

This work has demonstrated the power of the parallel form of IIR filters in parallel computing. This form allows us not only the robust design of high-order filters with logarithmic frequency resolution, but also a very efficient implementation on GPUs. The proposed

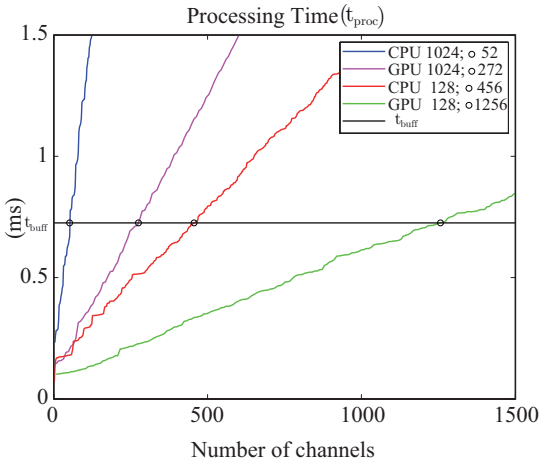

Fig. 4. Performance comparison between multi-core CPU and GPU implementations for parallel filters composed of 1024 and 128 second-order sections with a buffer size of 32 samples.

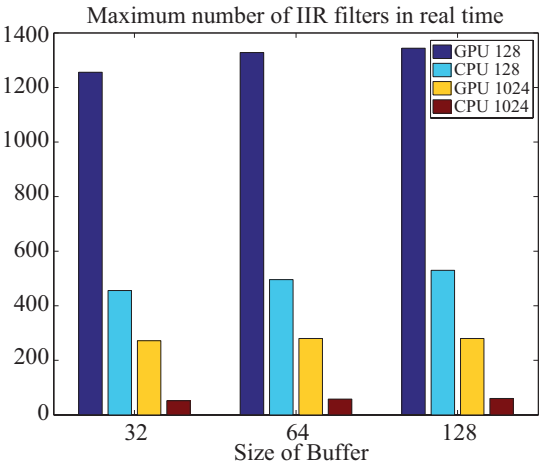

Fig. 5. Maximum number of IIR filters that can be realized in real time for the multi-core and GPU implementation for filters composed of 1024 and 128 second-order sections.

implementation can carry out up to 1256 equalizers with a filter order of 256 in real time, which means 321536 total filter order, for a buffer size of 32 samples. Many applications can be favored from this result, including a total equalization of a WFS system. In addition, we have compared the proposed GPU-based implementation with a multicore-based implementation in a powerful computer. Results show that GPU outperforms the powerful multicore computer in all example cases.

6. REFERENCES

[1] A. Härmä, M. Karjalainen, L. Savioja, V. Välimäki, and J. Huopaniemi, "Frequency-Warped Signal Processing for Audio Applications," *J. Audio Eng. Soc*, vol. 48, no. 11, pp. 1011–1031, 2000.

[2] M. Karjalainen and T. Paatero, "Equalization of loudspeaker and room responses using Kautz filters: Direct least squares design," *EURASIP J. on Advances in Sign. Proc., Spec. Iss. on Spatial Sound and Virtual Acoustics*, vol. 2007, pp. 13, 2007.

[3] B. Bank, "Perceptually motivated audio equalization using fixed-pole parallel second-order filters," *IEEE Signal Process. Lett.*, vol. 15, pp. 477–480, 2008.

[4] K. Steiglitz and L. E. McBride, "A technique for the indentification of linear systems," *IEEE Trans. Autom. Control*, vol. AC-10, pp. 461–464, Oct. 1965.

[5] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1975.

[6] "Audio and Communications Signal Processing Group at Universitat Politecnica de Valencia," *http://www.gtac.upv.es/enlaces.asp*.

[7] "IOSONO Wave Field Synthesis System," *http://www.timelab-hhi.com/en/system-description/iosono-wave-field-synthesis-system.html*.

[8] L. Savioja, "Real-time 3D finite-difference time-domain simulation of low- and mid-frequency room acoustics," in *Proc. of the Int. Conf. Digital Audio Effects*, Graz, Austria, September 2010.

[9] A. Southern, D. Murphy, G. Campos, and P. Dias, "Finite difference room acoustic modelling on a General Purpose Graphics Processing Unit," in *Proc. of the 128th AES Convention*, London, United Kingdom, May 2010.

[10] C. J. Webb and S Bilbao, "Computing room acoustics with CUDA - 3D FDTD schemes with boundary losses and viscosity," in *Proc. IEEE International Conference on Acoustics, Speech and Signal (ICASSP)*, Prague, Czech Republic, May 2011.

[11] B. Hamilton and C. J. Webb, "Room acoustics modelling using GPU-accelerated finite difference and finite volume methods on a face-centered cubic grid," in *Proc. Conference on Digital Audio Effects (DAFx-13)*, Maynooth, Ireland, September 2013.

[12] L. Savioja, V. Välimäki, and J. O. Smith, "Real-time additive synthesis with one million sinusoids using a GPU," in *Proc. of the 128th AES Convention*, London, United Kingdom, May 2010.

[13] L. Savioja, V. Välimäki, and J. O. Smith, "Audio signal processing using Graphics Processing Units," *J. Audio Eng. Soc*, vol. 59, no. 1-2, pp. 3–19, 2011.

[14] R. Bradford, J. Ffitch, and R. Dobson, "Real-time sliding phase vocoder using a commodity GPU," in *Proc. of ICMC 2011*, University of Huddersfield, United Kingdom, August 2011.

[15] J. Lorente, G. Piñero, A.M. Vidal, J.A. Belloch, and A. Gonzalez, "Parallel implementations of beamforming design and filtering for microphone array applications," in *Proc. of EUSIPCO 2011*, Barcelona, Spain, August 2011.

[16] N. Tsingos, W. Jiang, and I. Williams, "Using programmable graphics hardware for acoustics and audio rendering," *J. Audio Eng. Soc*, vol. 59, no. 9, pp. 628–646, 2011.

[17] J. A. Belloch, M. Ferrer, A. Gonzalez, F.J. Martinez-Zaldivar, and A. M. Vidal, "Headphone-based virtual spatialization of sound with a GPU accelerator," *J. Audio Eng. Soc*, vol. 61, no. 7/8, pp. 546–561, 2013.

[18] J. A. Belloch, A. Gonzalez, F.J. Martinez-Zaldivar, and A. M. Vidal, "Multichannel massive audio processing for a generalized crosstalk cancellation and equalization application using GPUs," *Integrated Computer-Aided Engineering*, vol. 20, no. 2, pp. 169–182, 2013.

[19] M. Schneider, F. Schuh, and W. Kellermann, "The generalized frequency-domain adaptive filtering algorithm implemented on a GPU for large-scale multichannel acoustic echo cancellation," *Proc. of Speech Communication; 10. ITG Symposium*, pp. 1 –4, Sept. 2012.

[20] J. Lorente, A. Gonzalez, M. Ferrer, J.A. Belloch, M. De Diego, G. Piñero, and A.M. Vidal, "Active noise conrol using Graphics Processing Units," in *Proc of the International Congress on Sound and Vibration*, Vilnius, Lithuania, July 2012.

[21] 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 *Proc. of the 52nd AES Conference*, Guildford, United Kingdom, September 2013.

[22] J.A. Belloch, M. Ferrer, A. Gonzalez, J. Lorente, and A.M. Vidal, "GPU-based WFS Systems with Mobile Virtual Sound Sources and Room Compensation," in *Proc. of the 52nd AES Conference*, Guildford, United Kingdom, September 2013.

[23] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, "Multi-core platforms for beamforming and wave field synthesis," *IEEE Transactions on Multimedia*, vol. 3, no. 2, pp. 235–245, April 2011.

[24] F. Trebien and M. M. Oliveira, "Realistic real-time sound resynthesis and processing for interactive virtual worlds," *The Visual Computer*, vol. 25, no. 5-7, pp. 469–477, 2009.

[25] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1975.

[26] B. Bank, "Audio equalization with fixed-pole parallel filters: An efficient alternative to complex smoothing," *J. Audio Eng. Soc.*, vol. 61, no. 1/2, pp. 39–49, Jan. 2013.

[27] B. Bank, "Loudspeaker and room response equalization using parallel filters: Comparison of pole positioning strategies," in *Proc. 51st AES Conf.*, Helsinki, Finland, Aug. 2013.

[28] B. Bank, "Logarithmic frequency scale parallel filter design with complex and magnitude-only specifications," *IEEE Signal Process. Lett.*, vol. 18, no. 2, pp. 138–141, Feb. 2011.

[29] "Nvidia Programming Guide," *online at: http://developer.download.nvidia.com/*.

[30] "Nvidia K20 Kepler Architecture," *online at: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.*

[31] "Optimizing Parallel Reduction in CUDA," *online at: http://developer.download.nvidia.com/assets/cuda/files/reduction.pdf*.

[32] "openMP API Specifications," *online at: http://www.openmp.org.*