Multi-Core Platforms for Beamforming and Wave Field Synthesis

Article *in* IEEE Transactions on Multimedia · May 2011

DOI: 10.1109/TMM.2010.2098397 · Source: IEEE Xplore

CITATIONS
19

READS
192

3 authors:

Dimitris Theodoropoulos
Technical University of Crete
44 PUBLICATIONS **216** CITATIONS
SEE PROFILE

Georgi Kuzmanov
Delft University of Technology
97 PUBLICATIONS **1,426** CITATIONS
SEE PROFILE

Georgi Nedeltchev Gaydadjiev
Imperial College London
247 PUBLICATIONS **2,581** CITATIONS
SEE PROFILE

Multi-core Platforms for Beamforming and Wave Field Synthesis

Dimitris Theodoropoulos, *student member IEEE*
Georgi Kuzmanov, *member IEEE*
Georgi Gaydadjiev, *member IEEE*

***Abstract*—Immersive-Audio technologies are widely used to build experimental and commercial audio systems. However, most of them are based on standard PCs, which introduce performance limitations and excessive power consumption. To address these drawbacks, we explore the implementation prospectives of two Immersive-Audio technologies; the beamforming (BF) and the Wave Field Synthesis (WFS). We target two popular multi-core platforms, namely Graphic Processor Units (GPUs) and Field Programmable Gate Arrays (FPGAs). We identify the most computationally intensive parts of both applications and employ the CUDA environment to map them onto a Quadro FX1700, a GeForce 8600GT, a GTX275 and a GTX460 GPU. Furthermore, we design our custom multi-core hardware accelerators for both algorithms and map them onto Virtex6 FPGAs. Both GPU and FPGA implementations are compared against OpenMP-annotated software running on a Core2 Duo at 3.0 GHz. Experimental results suggest that middle-range GPUs process data equally well as the Core2 Duo for the BF, and approximately two times faster for the WFS. However, high-end GPU and FPGA solutions provide an order of magnitude better performance for BF, and approximately two orders of magnitude better performance for WFS than the Core2 Duo. Ultimately, single-chip GPU and FPGA implementations can provide more power-effective solutions, since they can drive more complex microphone and loudspeaker setups than PC-based approaches.**

I. Introduction

Various advanced audio systems have been proposed by researchers from the acoustic data processing domain. The majority of such systems are based on Immersive-Audio algorithms that significantly enhance the sound realism. Normally, a high-end Immersive-Audio rendering system consists of a *sound rendering* module, which utilizes Immersive-Audio algorithms, like the Wave Field Synthesis (WFS) [1] or Ambisonics [2], to render all audio sources through a loudspeaker array. Furthermore, there are cases where real-time acoustic sources from a remote location need to be rendered through the loudspeaker array (e.g. teleconferences). For that reason, *sound acquisition* modules apply spatial filtering techniques, such as BF [3], to record and extract all sources by utilizing a microphone array.

The majority of the experimental systems reported in literature up to date are implemented using standard desktop PCs. The primary reason is because a General Purpose Processor (GPP) can be easily programmed to execute Immersive-Audio applications, thus considerably shortening the development time. However, this approach introduces processing bottlenecks and excessive power consumption. In order to alleviate these problems, in our work we investigate the potentials of other off-the-shelf hardware platforms, such as Graphic Processor Units (GPUs) and Field Programmable Gate Arrays (FPGAs) to build Immersive-Audio systems. More specifically, the contributions of this paper are the following:

- We analyze the potentials of BF and WFS for multi-core implementations;
- We specify parallel algorithms to map BF and WFS onto GPUs;
- We improve previously proposed hardware accelerators of BF and WFS and map them onto Virtex6 FPGAs;
- We provide an extensive design space exploration on BF and WFS with respect to three multi-core platforms - GPUs, FPGAs and GPPs, under different I/O setups;
- Experimental results for each targeted platform suggest that high-end GPUs and FPGA-based solutions provide an order of magnitude better performance for BF and approximately two orders of magnitude better performance for WFS than OpenMP-annotated software executed at a Core2 Duo at 3.0 GHz;
- Preliminary power estimations indicate that a GPU implementation is more power efficient compared to a GPP approach. Furthermore, an FPGA solution consumes an order of magnitude less power than contemporary GPUs.

The rest of the paper is organized as follows: Section II provides a brief theoretical background of the two Immersive-Audio algorithms that were used. Furthermore, various systems that utilize these algorithms are presented. In Section III and Section IV, we describe the GPU and FPGA implementations of the BF and WFS algorithms respectively. Section V compares the performance of the Immersive-Audio algorithms when mapped onto the three different platforms. Finally, in Section VI we conclude this paper.

II. Background and Related Work

In this section, we briefly present the theoretical background on BF and WFS, and provide a few references to related state-of-the-art.

Manuscript received April 1, 2010; revised September 23, 2010, accepted November 19,2010

The authors are with the Department of Computer Engineering, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands, e-mails: D.Theodoropoulos@tudelft.nl, G.K.Kuzmanov@tudelft.nl, g.n.gaydadjiev@tudelft.nl

See http://ce.et.tudelft.nl for current contact information.

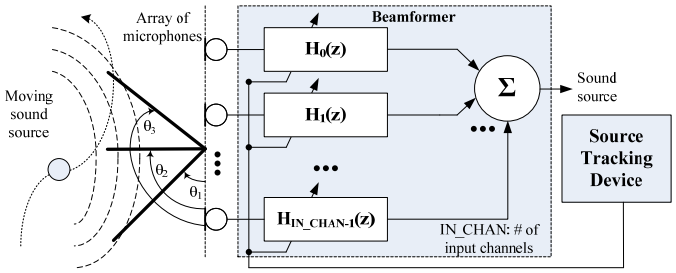

Fig. 1. A filter-and-sum beamformer.

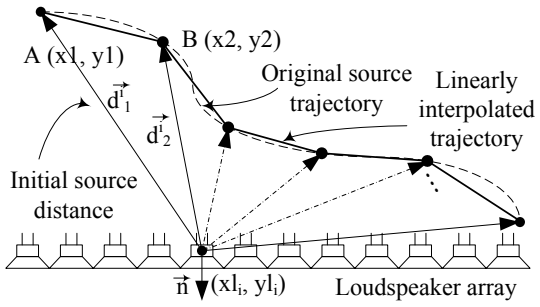

Fig. 2. Linear interpolation of a moving sound source.

A. Theoretical Background

Beamforming: BF is used to perform spatial filtering over a certain area, in order to estimate the direction of arrival of a signal. This technique allows to distinguish two signals with overlapping frequencies as long as they originate from different locations. Although there are different approaches to perform BF [3], many systems utilize a filter-and-sum approach, as illustrated in Figure 1. A microphone array samples the propagating wavefronts and each microphone is connected to an $H_i(z)$ Finite Impulse Response (FIR) filter, where i=0...*IN_CHAN*-1 and *IN_CHAN* is the number of input channels. All filtered signals are accumulated, in order to strengthen the primary audio source and attenuate any ambient noise. The FIR filters are essentially utilized as delay lines that compensate for the introduced delay of the wavefront arrival at all microphones [4]. However, in order to effectively extract a sound source, it is mandatory to reconfigure the FIR filter coefficients according to the source location. For example, as it is illustrated in Figure 1, a source is recorded for a certain amount of time inside the aperture defined by the $\theta_2 - \theta_1$ angle. A source tracking device is employed to follow the source trajectory and based on its coordinates, the FIR filters are configured with the proper coefficients set. As soon as the source moves inside the aperture defined by the $\theta_3 - \theta_2$ angle, the source tracking device will provide the new coordinates, thus the FIR coefficients will be updated with a new set. The latter process is normally referred to as "*beamsteering*".

Wave Field Synthesis: The WFS acoustic algorithm was initially proposed by Berkhout [1]. It is essentially based on Huygens' principle, which is applied by stating that a primary source wavefront can be created by the superposition of secondary audio sources, i.e. plane of speakers, that emit secondary wavefronts. However, in real world systems a plane of loudspeakers is not practical, so a linear loudspeaker array is used instead. As a result, the finite distance between the loudspeakers introduces artifacts such as spatial aliasing, truncation effects, and amplitude and spectral errors [5]. However, the WFS algorithm alleviates many problems that are inherent to other audio systems, like stereophony. Its most important advantage is that there is no "sweet spot" area restriction. In contrast to stereophonic systems that require the listeners to remain at the center of the listening area, WFS allows people to move freely inside the entire acoustic area and still experience an outstanding audio environment perception [6].

Figure 2 illustrates an example of a linear loudspeaker array setup. Each speaker has its own coordinates $(xl_i,\ yl_i)$ inside the listening area. $\overrightarrow{d_1^i}$ and $\overrightarrow{d_2^i}$ are the initial and final source distances from a particular loudspeaker respectively within a certain time frame, while $\overrightarrow{n}$ is a normal vector perpendicular to the loudspeaker array. In order to drive each one of the loudspeakers so as the rendered sound source location is at A(x1, y1), the so called *Rayleigh 2.5D operator* [7] has to be calculated:

$$Q_m(\omega, |\overrightarrow{d_1^i}|) = S(\omega)\sqrt{\frac{jk}{2\pi}}\sqrt{\frac{Dz}{z+Dz}}\frac{z}{|\overrightarrow{d_1^i}|}\frac{exp(-jk|\overrightarrow{d_1^i}|)}{\sqrt{|\overrightarrow{d_1^i}|}} \quad (1)$$

where $k = \frac{\omega}{c}$ is the wave number, c is the sound velocity, z is the inner product between $\overrightarrow{n}$ and $\overrightarrow{d_1^i}$, Dz is reference distance, $S(\omega)$ is the acoustic source, $\sqrt{\frac{jk}{2\pi}}$ is a 3dB/octave correction filter, $\sqrt{\frac{Dz}{z+Dz}}$ is the source amplitude decay and e^{-jkr} is a time delay that has to be applied to the particular loudspeaker.

When a sound source moves from point A(x1, y1) with distance $|\overrightarrow{d_1^i}|$ from a given loudspeaker to point B(x2, y2) with $|\overrightarrow{d_2^i}|$ distance from the same loudspeaker, the WFS algorithm linearly interpolates the traversed trajectory, as illustrated in Figure 2. Based on the calculated distance between A and B, it updates the source distance from the particular loudspeaker for every audio sample rendered. Finally, the source amplitude decay is computed [7] [8].

B. Related Work

Beamforming: Over the last years, various systems that utilize GPUs under different application domains have been published in the literature. In [9] the authors describe a hybrid approach that utilizes 14 Virtex4 LX25 FPGAs and a GPU connected to a desktop PC, to perform 3D-parallel BF and scan conversion for real-time ultrasonic imaging using input from 288 channels. In [10], the authors utilize a GeForce 8800 to design a delay-and-sum beamformer in the time and frequency domain, which according to the results, can achieve speedup up to 12x and 15x respectively, compared to a Xeon Quad-core processor. The authors of [11] present a hardware accelerator that utilizes microphone array algorithms based on calibrated signals and subband processing. In [12], a real-time Virtex4 FPGA-based beamformer that utilizes the QR matrix decomposition is presented. A data driven beamformer for a binaural headset is presented in [13]. The authors integrate

two microphones to the headphones and employ a Head and Torso Simulator to acquire the source signal for BF. Squarehead [14] develops the Audioscope, a dual core PC-based system, that employs 300 omnidirectional microphones for audio capturing. Acoustic Camera [15], develops PC-based BF systems, that utilize sound acquisition arrays ranging from few tens to more than hundred elements. In [16], the authors utilize an Analog Devices Digital Signal Processor (DSP) to perform Calibrated Weighted Recursive Least Squares-based BF over a two microphone array setup. An experimental video teleconferencing system based on a Texas Instruments DSP processor is presented in [17]. Finally, there are many projects that utilize different microphone array sizes and setups. One of the most famous implementations is the Large AcOUstic Data (LOUD) [18], developed in the Massachusetts Institute of Technology. The LOUD microphone array consist of 1020 elements arranged into a 2D planar setup.

Wave Field Synthesis: In [19], the authors describe an Immersive-Audio system consisting of 12 linearly placed microphones. The sound source is tracked, encoded and received from a second remote PC, which renders it using WFS through a 10-loudspeaker array. A similar system is presented in [20], which transmits real-time extracted audio sources to a remote location, and renders them using the WFS technology. An experimental PC-based WFS system has been developed at the Delft University of Technology [21], which consists of 114 loudspeakers. Another PC-based sound system that was built in IRT, Munich, called the Binaural Sky [22], actually combines both binaural [23] and WFS technologies. The SonicEmotion company [24] deploys its unit on an Intel Core2 Duo-based setup, which supports rendering up to 64 real-time sound sources, while driving a 24 loudspeaker array. Iosono [25] also follows a standard PC approach that supports up to 32 real-time sources while driving 32 speakers. Six such Iosono WFS systems were installed in 2003 in a cinema in Ilmenau, Germany to drive 192 loudspeakers. A GPU-based WFS implementation that utilizes the NU-Tech software framework [26] is discussed in [27]. The authors have developed a NU-Tech plug-in that uses the CUDA libraries for the required data calculations, and run it on a GeForce GTX285 and a Tesla C1060 GPU.

Overall scientists from the audio domain society use primarily desktop PCs to build experimental and consumer Immersive-Audio systems. However, these approaches introduce processing bottlenecks and excessive power consumption. In order to alleviate these problems, we conduct various experiments of the widely used BF and WFS applications to GPUs, FPGAs and GPPs regarding performance and power consumption.

III. Algorithms Proposed for GPUs

In this section we describe how we mapped the two Immersive-Audio applications onto GPUs [28]. Both BF and WFS applications can be considerably parallelized, thus making the GPUs a suitable target platform.

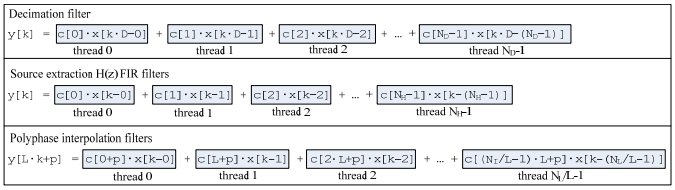

Fig. 3. Decimation, source extraction and interpolation filters onto GPU threads.

A. Beamforming

As indicated in Section II, BF is mainly based on FIR filters that compensate for the introduced delay of the sound wavefront arrival at all microphones [4]. In the general case, it consists of the following four phases:

1) Resample the input signal sample rate by a certain factor.
2) FIR filtering of each input signal with a specific set of coefficients.
3) Accumulation of all filtered signals to strengthen the main sound source.
4) Restore the extracted source signal sampling frequency back to the original one.

In the current BF application, we perform decimation and interpolation in phases 1 and 4 respectively. It is observed that in three out of four of the above phases FIR filtering is required, thus it is the dominant operation of the BF application. FIR filtering has the potential to be considerably accelerated when mapped onto GPUs, since the latter provide hundreds of processing elements that can calculate data concurrently. Therefore, we decided to develop a flexible computational kernel that could be reused with minor changes to efficiently map the decimation, FIR filtering and interpolation processes onto the GPU. Furthermore, an additional GPU kernel was developed to perform the accumulation of all filtered signals. Algorithm 1 illustrates how the BF application is mapped onto the GPU. Variable *IN_CHAN* represents the total number of input channels (microphones) available to the sound system, while *SOURCES* is the number of present sources that need to be extracted. We use the *GPU* annotation to designate the parts of the application that are executed by the GPU.

Input signal decimation: The initial sampling frequency f_s in our BF application is 48 kHz and a downsampling factor D of 4 is used to decimate the signal. However, before downsampling any signal, it is mandatory to filter it in order to avoid any aliasing effects. Since $D = 4$, the resulting signal will have f_s = 12 kHz, which is its new sampling frequency. This means that if the applied filter before downsampling eliminates all components above $\frac{12}{2}$ kHz = 6 kHz, which is the Nyquist frequency, then the final decimated signal will have no aliasing effects. Therefore, a decimation FIR filter with size of 242 taps is applied to cut off any component above 6 kHz.

In order to efficiently filter and downsample a signal, the following formula is used:

$$y[k] = \sum_{tx=0}^{N_D-1} c[tx] \cdot x[k \cdot D - tx] \tag{2}$$

Algorithm 1 BF implementation to GPU

Require: Input signals from IN_CHAN microphones
Ensure: Extracted $SOURCES$ audio sources

1: Move input data to GPU main memory
2: Store H(z) filter coefficients to GPU main memory
3: **for** $c = 0$ to $IN\_CHAN - 1$ **do**
4: GPU: Decimate channel c
5: **end for**
6: **for** $s = 0$ to $SOURCES - 1$ **do**
7: **for** $c = 0$ to $IN\_CHAN - 1$ **do**
8: GPU: Extract source s from channel c
9: **end for**
10: GPU: Accumulate signals from all channels
11: GPU: Upsample source s signal
12: **end for**
13: Move all extracted sources signals to CPU memory

TABLE I
SAMPLE, COEFFICIENT AND OUTPUT INDICES FOR THE BF APPLICATION.

GPU kernel	sample index	coefficient index	output index
decimation	bx·D+by·gxdim-tx	tx	bx+by·gxdim
H(z) filter	bx+by·gxdim-tx	angle·N_H·IN_CHAN+C_cur·N_H	bx+by·gxdim
interpolation	$\frac{bx}{L}$+by·gxdim-tx	tx	bx+by·gxdim

where *k* is the current filter output sample, N_D is the decimator filter size, *tx*=0...N_D-1, *c[]* represents the filter coefficients, *x[]* represents the current status of the filter delay line, and *D* is the downsampling factor. The first row of Figure 3 illustrates how (2) is mapped onto different GPU threads, in order to calculate a single output sample of the decimated signal. Each multiplication is mapped onto a different GPU thread and all of them are concurrently executed. As soon as all multiplications are performed, all results are accumulated and stored back to the GPU main memory. The second and third lines of Figure 3 are discussed in paragraphs "Source extraction" and "Source signal interpolation" later on.

Figure 4 depicts the grid of thread blocks that was designed to efficiently map the calculation of all decimated samples onto a GPU kernel. Input signals are divided into 1024-sample chunks. Since the decimator FIR filter size is 242, we launch the same number of GPU threads per block, that is *txdim* = 242 in Figure 4. This way we can map the calculation of each output sample of the decimated signal onto one thread block. Within each thread block variable *dOut* is used to store the corresponding output sample. Each thread calculates a proper sample and coefficient index to load the respective sample and filter coefficient from the GPU external memory to *x[]* and *c[]* arrays inside the shared memory of a GPU multiprocessor. We use the variables *bx* and *by* as coordinates of every thread block within the grid, thus *bx*=0...*gxdim*-1 and *by*=0...*gydim*-1. Since each thread block is responsible for calculating a single sample of the decimated signal, the illustrated grid in Figure 4 can process up to *gxdim·gydim* samples. Based on these variables, each thread calculates the correct sample, coefficient and output indices that are shown in the second row of Table I. As soon as the two values are multiplied, all threads are synchronized and data are accumulated to the

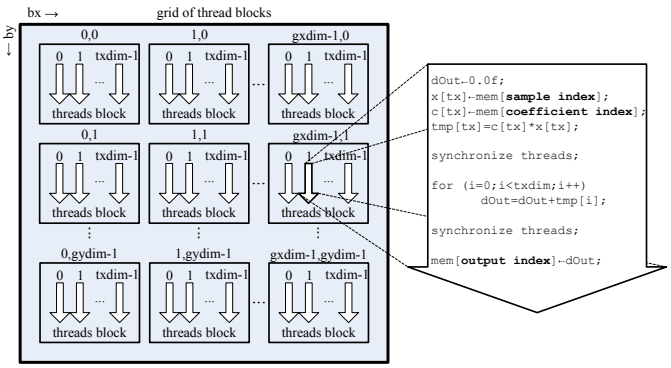

Fig. 4. Grid of thread blocks that are dispatched during the FIR filter calculations onto the GPU.

dOut variable. Threads are then again synchronized and the final result is stored to the main GPU memory based on the output index that each thread calculates.

Source extraction: In order to efficiently map the FIR filter operations onto the GPU, we used the same approach as in decimation. Each H(z) FIR filter is represented by:

$$y[k] = \sum_{tx=0}^{N_H-1} c[tx] \cdot x[k - tx] \quad (3)$$

where *k* is the current filter output sample, N_H is the H(z) filter size, *tx*=0...N_H-1, *c[]* represents the filter coefficients, and *x[]* represents the current status of the filter delay line. The second row in Figure 3 illustrates how (3) can be separated into different GPU threads, where each thread block is responsible for calculating a single output sample. Each multiplication is assigned to a unique GPU thread and all of them are executed concurrently. All results are accumulated, and the final value is stored back to the GPU main memory.

We designed a *gxdim* by *gydim* grid of thread blocks, as illustrated in Figure 4, where *bx* and *by* are the coordinates of each thread block within the grid. Each FIR filter is 128 taps long. Therefore, we use 128 GPU threads per block, thus *txdim* = 128 in Figure 4 for the source extraction. Finally, the third row of Table I illustrates how the new sample and coefficient indices are calculated within each GPU thread. The *angle* variable designates the source aperture, *IN_CHAN* is the total number of input channels, and *C_cur* is the current channel that is being processed.

The considered BF application can recognize 19 source apertures. Furthermore, since each FIR filter consists of 128 taps, we need to store 128 $\frac{coefficients}{aperture}$·19$\frac{apertures}{channel}$·IN_CHAN $channels$ = 2432·IN_CHAN $coefficients$, where each coefficient is in single precision floating point format. During our tests, we utilized from *IN_CHAN*=8 to 212 input channels, thus the total amount of data was between 77824 and 2062336 bytes respectively. Since current GPUs constant memory is not enough to fit all coefficients, we decided to store them to the GPU main memory.

Source signal interpolation: Signal interpolation consists of two consecutive stages: signal upsampling and filtering. Upsampling a signal by a factor *L* introduces undesired spectral images at multiples of the initial sampling frequency,

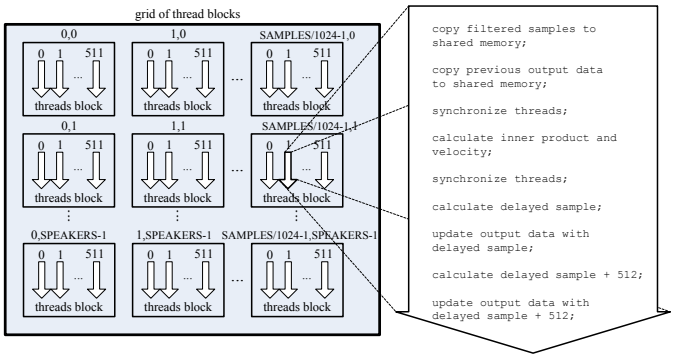

Fig. 5. Grid of thread blocks dispatched during the GPU WFS calculations.

Algorithm 2 Wave Field Synthesis implementation to GPU

Require: $SOURCES$ audio sources to be rendered to $SPEAKERS$ loudspeakers

Ensure: $SPEAKERS$ audio signals to drive all loudspeakers

1: Move input data to GPU main memory
2: **for** $s = 0$ to $SOURCES - 1$ **do**
3: GPU: Apply FIR filtering to source s
4: GPU: Calculate all $SPEAKERS$ signals for source s and accumulate with previous ones
5: **end for**
6: Move all $SPEAKERS$ signals back to CPU main memory

and for this reason, the upsampled signal needs to be filtered [29]. We upsampled the extracted source signal by inserting *L*-1 zeros between its samples. Following a polyphase filter approach [29], we can use the following equation to describe all *L* polyphase filters that are utilized to efficiently upsample and filter the input signal:

$$y[k \cdot L + p] = \sum_{tx=0}^{\frac{N_L}{L}-1} c[tx \cdot L + p] \cdot x[k - tx] \qquad (4)$$

where *k* is the current filter output sample, N_L is the interpolator filter size, *tx*=0...$\frac{N_L}{L}$-1, *c[]* represents the filter coefficients, *x[]* represents the current status of the filter delay line, *L* is the upsampling factor, and *p*=0...*L*-1 is the corresponding polyphase filter.

The third row in Figure 3 illustrates how (4) can be efficiently mapped to different GPU threads. Again, each multiplication is assigned to a different GPU thread, while all threads are executed concurrently. Based on the polyphase filter approach [29], the interpolation FIR filter length should be a multiple of *L*. The BF application that we used during our experiments utilizes a FIR filter with N_L=242 taps long, while the upsampling factor *L* = *D* = 4. In order to make the FIR filter length multiple of 4 we added two more taps at the end of its delay line with both coefficients being equal to 0, thus increasing the FIR filter to 244 taps. Since *L* = 4, we use *p* = 4 polyphase filters to upsample the extracted source signal, each one being $\frac{244}{4}$=61 taps long. The multiplication results are accumulated, and the computed value is stored back to the GPU main memory. We designed a *gxdim* by *gydim* grid of thread blocks, as depicted in Figure 4, to calculate *gxdim·gydim* output samples. We use again the variables *bx* and *by* as coordinates of every thread block within the grid. Since each polyphase filter is 61 taps long, we launch 61 GPU threads per block, thus *txdim* = 61. The fourth row of Table I suggests how the GPU kernel for the signal interpolation calculates the corresponding sample, coefficient and output indices.

B. Wave Field Synthesis

We have already proposed a GPU implementation of the WFS acoustic algorithm in [30]. However, an improvement of the current implementation compared to previous work is the execution of the FIR filtering on the GPU and not on the CPU. We moved the FIR filter execution to the GPU because tests reported in the literature indicate more than an order of magnitude performance improvement [31] against a CPU approach.

Algorithm 2 sketches how the WFS is mapped onto the GPU. The variable *SPEAKERS* represents the number of loudspeakers of the audio system. The variable *SOURCES* is the number of audio sources that need to be rendered through *SPEAKERS* loudspeakers. Again, we use the *GPU* annotation to designate the GPU kernels developed. In order to filter all audio sources, we used the same FIR filter GPU kernel that was described in BF. However, in this case, the FIR filter consists of 64 taps, so 64 threads are launched within each thread block. Figure 5 illustrates the WFS implementation on the GPU. Input signals are divided into chunks of 1024 samples. We designed a 2D-grid of thread blocks, where each block consists of 512 GPU threads. Each thread within a block is responsible for calculating 2 output samples, thus when a thread block is executed, 1024 samples are calculated. Assuming there are *SAMPLES* input samples to be processed and rendered through $SPEAKERS$ loudspeakers, we need $\frac{SAMPLES}{1024} \cdot SPEAKERS$ thread blocks for each loudspeaker. Therefore, we designed the 2D-grid with dimensions $\frac{SAMPLES}{1024}$ by $SPEAKERS$.

IV. Custom Hardware Designs

In this section, we describe our custom implementations of BF and WFS applications in hardware. Initially, the BF hardware accelerator was proposed in [32] and the WFS one in [33]. However, for the current paper both hardware accelerators have been redesigned and tested on the newer Virtex6 family FPGAs.

Beamforming: Figure 6 depicts the Fabric Coprocessor Module-BF (FCM-BF)[1], which receives 16-bit signed audio samples. All calculations are performed in a fixed-point arithmetic format. The accelerator utilizes the Auxiliary Processor Unit (APU) interface to communicate with the host processor and can be customized according to the number of system input channels, the decimation and interpolation factor, and

[1]The Xilinx terminology is employed.

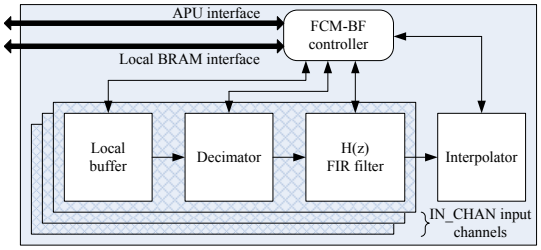

Fig. 6. Hardware implementation of the filter-and-sum beamformer.

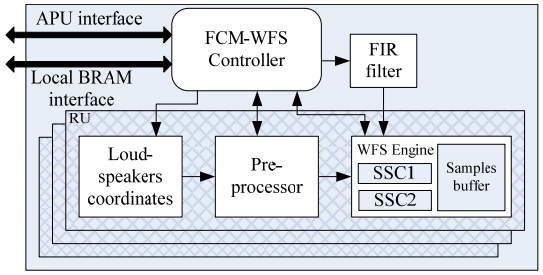

Fig. 7. Hardware implementation of the WFS rendering module.

the sampling frequency. The gridded area in Figure 6 wraps all modules that have to be replicated for every input channel. A *local buffer* is used to temporarily store 1024 16-bit audio samples for the particular channel before they are processed, which are fetched through the *local BRAM interface*. As soon as all samples from all input channels are stored to their corresponding local buffers, the *FCM-BF controller* initiates the samples processing.

All input channels are processed concurrently since there is no data dependency among them. The *decimator* is used to downsample the input data. Every *H(z) FIR filter* includes a specific set of FIR filter beamsteering coefficients banks. Based on the source aperture, the appropriate FIR coefficient set is selected to apply a certain delay to the downsampled data. The output signals of all *H(z) FIR filters* are accumulated in order to strengthen the source signal. The beamformed signal is upsampled by the interpolator to the initial sampling frequency. The current FCM-BF implementation requires 68871 clock cycles to process all 1024 samples for all channels. It employs an internal fixed point format [32], which provides data that are 99.6% accuracy compared to the software implementation.

Wave Field Synthesis: Figure 7 illustrates the FCM-WFS. We consider 16-bit signed audio samples, while all computations are performed in a fixed-point arithmetic format. The accelerator can be parameterized according to the number of system output channels, i.e. loudspeakers, and is connected to the host processor via the APU interface. As soon as the host processor invokes the WFS hardware accelerator, the *FCM-WFS Controller* accesses a local memory module to fetch 1024 audio samples. These samples are processed from a 3dB/octave frequency correction *FIR filter* and stored to the *Samples buffer* inside the *WFS engine* module. The WFS algorithm allows parallel data processing for each individual loudspeaker. This fact is efficiently exploited by the FCM-WFS, since it distributes the required processing tasks for a certain set of loudspeakers to different RUs (Rendering Units). For example, if there is a loudspeaker array setup consisting of 32 elements, then two RUs could be employed to process each one of the 16 elements, assuming enough hardware resources. However, if there is limited available area, then the WFS hardware accelerator can be reconfigured to employ a single RU to process the entire loudspeaker array.

The gridded area in Figure 7 illustrates the RU internal modules. The *Loudspeakers coordinates* is a small ROM that keeps the coordinates of all loudspeakers that will be processed from a particular RU. Within each RU, all loudspeakers are processed iteratively, whereas in every iteration 1024 16-bit audio samples are computed. The *Preprocessor* module calculates the initial and final source distances from a particular loudspeaker during a time frame that is equal to render 1024 samples. For example, assuming a sampling frequency of 48000 Hz, the time frame will be approximately 1024/48000 $\approx$ 21.33 msec. Furthermore, based on the initial and final source distances, the *Preprocessor* calculates also the source velocity and amplitude decay within this particular time frame. The *Preprocessor* requires 142 clock cycles to complete data processing. The *WFS Engine* module uses the source velocity and amplitude decay to process all 1024 audio samples. In order to reduce the processing time, within the *WFS Engine* there are two Samples Selection Cores (SSCs) submodules that process concurrently 2 samples per clock cycle, requiring in total 531 cycles to process 1024 samples. An internal fixed point format [33] provides data accuracy of 99.4% compared to the software implementation. All calculated audio samples are stored back to the local BRAM.

TABLE II
VIRTEX6 FCM POST-PLACE-AND-ROUTE RESULTS

	Slices	DSP Slices	BRAM (bytes)	FCM Freq. (MHz)
BF (1 chan.)	174	2	23040	300
WFS (1 RU)	1232	26	92160	200

A complete Immersive-Audio reconfigurable processor was designed and prototyped on a Xilinx Virtex4-based ML410 board. The processor is separated into two parts, the *Sound Acquisition* and *Sound Rendering*. The *Sound Acquisition* part utilizes a PowerPC as host processor and the BF hardware accelerator. The *Sound Rendering* part utilizes the WFS hardware accelerator, which is connected to a second PowerPC. Further details can be found in [34].

V. PERFORMANCE ANALYSIS

We used the Xilinx ISE 11.4 and EDK 11.4 CAD tools to develop our custom hardware accelerators. Table II presents the resource distribution among different processor modules and the maximum operating frequency of the complete FCM-BF and FCM-WFS accelerators. We should note that for all Virtex6 implementations, we considered the measured bandwidth of 6400 MB/sec, reported in [35]. Based on these data, Table III suggests how many BF channels and WFS RUs could fit in the Virtex4 FX140 chip, and the smallest and largest chip of each Virtex6 FPGA family.

Regarding the GPUs that were used during our experiments, we chose chips that would represent a wide range of the mar-

TABLE III
MAXIMUM NUMBER OF FCMS FOR BF AND WFS

FPGA	FCM-WFS fit	FCM-BF fit
6VLX75T	7	31
6VLX760	36	144
6VSX315T	35	140
6VSX475T	**53**	**212**
6VHX250T	24	100
6VHX565T	36	182
V4FX140	12	62

TABLE IV
GPUS SPECIFICATIONS FOR ALL EXPERIMENTS.

GPU chip	Core (Mhz)	CUDA core (Mhz)	Memory BW (GB/s)	# of CUDA cores	FF-XIV Score
FX1700	207	414	12.8	32	694
8600GT	540	1190	22.4	32	898
GTX275	633	1404	127	240	3817
GTX460	675	1350	115.2	336	3716

ket. For this reason, we used a middle-range Quadro FX1700, a slightly faster GeForce 8600GT, a high-end GeForce GTX275, and the newest GTX460 that utilizes the Fermi architecture [28] with 2 levels of on-chip cache. Table IV presents the specifications of the GPUs considered. It can be observed that the 8600GT has the same number of CUDA cores compared to the FX1700 chip. However, the 8600GT has higher memory bandwidth (BW) and faster Core and CUDA core clocks compared to the FX1700. Furthermore, the GTX275 and the GTX460 consist of 240 and 336 CUDA cores respectively, an order of magnitude higher memory bandwidth, and faster clocks compared to the other two GPUs. In order to test the processing capabilities of each GPU, we installed each one of them to our system and ran the official Final Fantasy XIV (FF-XIV) benchmark for NVidia GPUs[2] on low resolution. The achieved benchmark score of each GPU is shown to the rightmost column of Table IV. As we can observe, the 8600GT scored ∼30% higher than the FX1700, while the GTX275 performed 2.7% better than the GTX460. We should note that no over-clocking was applied to any of the GPUs considered.

In order to test the performance of all platforms, we provided source signals consisting of 540672 audio samples, divided into 528 1024-sample chunks. Assuming a sampling frequency of 48000 kHz, each iteration should be calculated in $\frac{1024}{48000}\ \frac{samples}{samples/sec} \approx 21.33$ msec. Since there are in total 528 iterations, every hardware platform that is considered for a real-time audio system must complete all data calculations within 528 $iterations \cdot 21.33\ \frac{msec}{iteration} \approx 11.264$ sec.

A. Beamforming results

It was noted in Section II that there are many BF implementations that utilize relatively small microphone arrays. To challenge our approach, we conducted experiments for all hardware platforms with 8 and 16 channels and up to 16 sources. Figure 8 depicts the execution times for setups consisting of 8 input channels, mapped onto all platforms considered. The secondary Y-axis of the same figure illustrates the obtained speedup against the Core2 Duo. It can be observed

[2]http://www.finalfantasyxiv.com/media/benchmark/na/#1

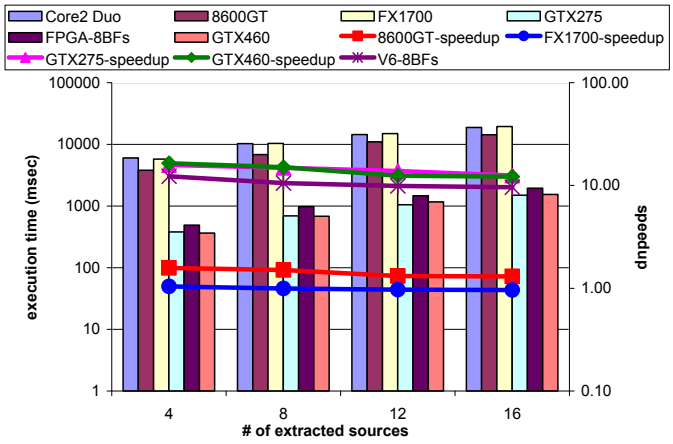

Fig. 8. BF execution times for 8 channels and speedup against Core2 Duo.

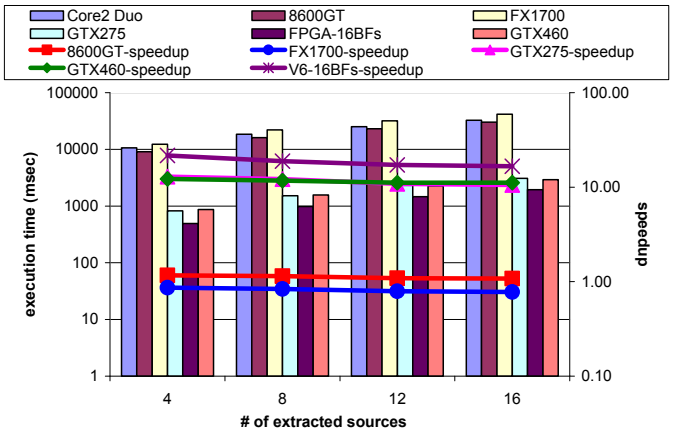

Fig. 9. BF execution times for 16 channels and speedup against Core2 Duo.

that all platforms considered can process all data in real time, thus within 11.264 sec, for up to 8 sources. In the case of 12 sources, the Core2 Duo and the FX1700, which is the simplest of all GPUs, fail to process data within the time constraint. The 8600GT cannot process all data fast enough when there are 16 sources. In contrast, the GTX275, the GTX460 and the FPGA implementations could be used to build a real-time system that is capable to extract up to 16 sources. An interesting observation that can be made from the execution times is that the GTX275 performs equally well to the GTX460.

Similar results were obtained when running the BF application with 16 input channels, as illustrated in Figure 9. However, the FX1700 failed to process data in real-time for all cases, while the Core2 Duo and 8600GT could be used for a real-time BF system when there are up to 4 sources. Again, the GTX275, the GTX460, and the FPGA implementations were able to perform real-time processing in all cases.

All execution times reported above include the external memory access delays. Each platform has its own memory hierarchy, which impacts the overall execution time differently. For example, the Core2 Duo has to access the external memory through the Front Side Bus (FSB) and the motherboard chipset (usually referred to as North Bridge), while GPUs and FPGAs have direct connection with their own external memory. Therefore, we measured the memory impact, excluded it from

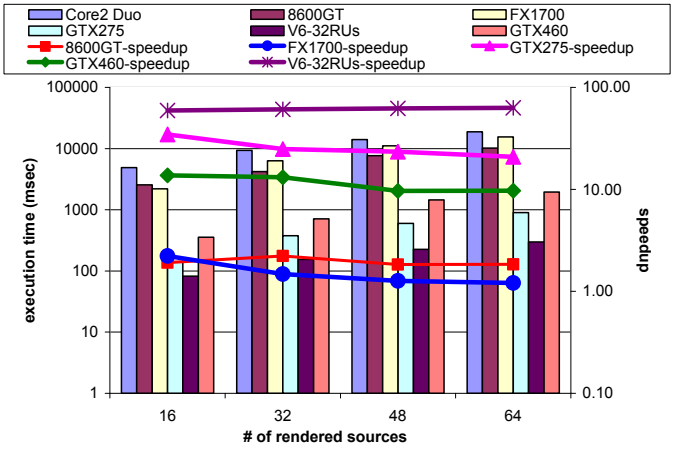

Fig. 10. WFS execution for 32 loudspeakers and speedup against Core2 Duo.

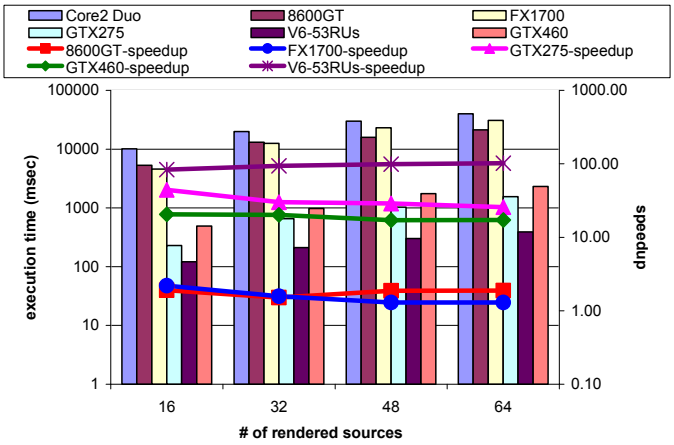

Fig. 11. WFS execution for 96 loudspeakers and speedup against Core2 Duo.

the total execution time, and compared the best hardware platforms in terms of processing speed. As indicated in the previous comparison, the GTX275, the GTX460 and the FPGA implementations performed always an order of magnitude better than the other platforms. However, they are fabricated using a different integration technology; the GTX275 uses an older 55 nm one, while the GTX460 and the FPGA are fabricated using a 40 nm technology. In order to normalize this technology factor, we referred to the International Technology Roadmap for Semiconductors (ITRS) 2005 report [36] to calculate the percentage ratio of the GTX275 processing time reduction, assuming that the chip would be fabricated using a 40 nm technology. Throughout this paper, we refer to it as *optimized GTX275*. Based on the ITRS report and aware of all the odds of such an estimation, we could safely assume that the 40 nm technology has potentials for 30% lower execution time than the 55 nm one. Furthermore, we used the CUDA Visual Profiler 3.1 to measure the achieved memory throughput, and subtracted it from the total execution time. We calculated that an average of 54.1% of the total execution time throughout our experiments was spent on accessing the GPU external memory. The remaining time, which is the *actual* GPU processing time, was multiplied by a factor of 0.7, in order to be normalized to the 40nm hypothetical GPU fabrication. For the GTX460, the CUDA Profiler 3.1 does not provide the achieved memory throughput. For this reason, it has been excluded from our comparison of the processing times.

Columns 1 to 12 in Table V summarize the ratio t_{GTX275}/t_{FPGA} between the GTX275 and the FPGA processing times, where t_{GTX275} and t_{FPGA} are the GTX275 and FPGA processing times respectively. In columns with the **opt** suffix, the optimized GTX275 is used. If the ratio is smaller than 1.0, the GPU is faster, otherwise the FPGA performs better. We performed additional experiments from 8 up to 212 channels, which is the maximum number of FCM-BF channels the largest Virtex6 can fit. As it was already indicated in Table II, each FCM-BF channel occupies only 2 XtremeDSP slices and 174 regular slices. In Table V, it can be observed that, when the input channels are 32 or more, the FPGA processes data faster than the optimized GTX275, because all FCM-BFs work concurrently on all input channels. For example, as it is also suggested in Table V, in the case of 212 input channels, a Virtex6 implementation could process data up to 9 times faster, when 4 sources are extracted, compared to the optimized GTX275 approach.

B. *Wave Field Synthesis results*

We conducted experiments with setups of 32 and 96 loudspeakers. Within each different setup, we measured the performance of all platforms when rendering from 16 to 64 audio sources. For the FPGA implementation, we consider the largest Virtex6 FPGA chip that can fit up to 53 Rendering Units (RUs) and each RU can drive one loudspeaker. In cases of more loudspeakers than RUs, the loudspeaker array is processed concurrently in chunks by 53 RUs.

Figure 10 provides the execution time of all platforms when driving a 32-loudspeaker array. The secondary Y-axis of the same figure illustrates the achieved speedups for all platforms against the Core2 Duo at 3.0 GHz. For up to 32 sources, all platforms considered could be employed in a real-time WFS audio system. The Core2 however fails to meet the timing constraints when the number of audio sources increases to 48. When there are 64 sources, the FX1700 GPU also processes data slower than the required audio rate. In contrast, the GTX275, GTX460 and FPGA implementations can process the same data in real-time. Furthermore, Figure 10 suggests that the Virtex6 FPGA can achieve a speedup of more than an order of magnitude compared to the Core2 implementation. In this case, the FPGA implementation consists of 32 RUs.

Similar results were obtained when running the WFS application under a 96-loudspeaker setup, as suggested in Figure 11. The Core2 Duo implementation could be used for no more than 16 sources. Furthermore, the two middle-ranged GPUs could be used to build a real-time audio system when its specifications require less than 32 audio sources. However, a GTX275, GTX460 or an FPGA solution could be used even when there are 96 loudspeakers. Moreover, the GTX275 and GTX460 implementations could process data one order of magnitude faster than the Core2 one, while the

TABLE V
RATIO t_{GTX275}/t_{FPGA} BETWEEN GTX275 AND FPGA PROCESSING TIMES FOR BF AND WFS.

Sources	Beamforming channels												Sources	WFS loudspeakers					
↓	8	8-opt	16	16-opt	32	32-opt	64	64-opt	128	128-opt	212	212-opt	↓	32	32-opt	64	64-opt	96	96-opt
4	0.31	0.22	0.69	0.49	*1.64*	*1.16*	*3.56*	*2.52*	*7.64*	*5.41*	*12.81*	*9.07*	16	0.43	0.31	*1.10*	0.78	*1.46*	*1.04*
8	0.31	0.22	0.70	0.50	*1.58*	*1.12*	*3.27*	*2.32*	*6.73*	*4.77*	*11.28*	*7.99*	32	0.64	0.45	*1.59*	*1.13*	2.16	*1.53*
12	0.33	0.23	0.74	0.53	*1.54*	*1.09*	*3.13*	*2.22*	*6.53*	*4.63*	*10.78*	*7.64*	48	0.68	0.49	*1.66*	*1.18*	*2.26*	*1.60*
16	0.36	0.25	0.75	0.53	*1.54*	*1.09*	*3.10*	*2.20*	*6.36*	*4.50*	*10.58*	*7.50*	64	0.77	0.54	*1.88*	*1.33*	*2.54*	*1.80*
Column # →	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**

reconfigurable hardware solution achieved a speedup of two orders of magnitude.

From the above results, we can conclude that the latest generation GPUs, such as the GTX275 and the GTX460, and the custom hardware solutions are the most suitable to build high-end Immersive-Audio real-time systems. In order to have a better evaluation of the actual processing times, we measured the time spent on memory transactions and subtract it from the total execution time. Using the CUDA Visual Profiler 3.1 to measure the actual memory throughput of the WFS application when executed on the GTX275, we calculated that an average of 59.7% of the total execution time throughout our experiments was spent on accessing the GPU external memory. In order to normalize the performance of the GTX275 execution time by the technology scaling factor 0.7, as explained in Section V-A.

Columns 14 to 19 in Table V summarize the ratio t_{GTX275}/t_{FPGA} between the GTX275 and the FPGA processing times, where t_{GTX275} and t_{FPGA} are the GTX275 and FPGA processing times respectively. In columns with the **opt** suffix, the optimized GTX275 is used. We performed additional experiments with 32, 64 and 96 loudspeakers. It has been observed that, in the case of a 32-element loudspeaker array the technology optimized GTX275 process data almost two times faster than the Virtex6 design, even when rendering 64 sources. The FPGA performs poorer than the GPU, because there are only 32 RUs, thus it is under-utilized. The GPU processes data faster than the FPGA also when the loudspeaker array is increased to 64 elements and there are less than 26 sources to be rendered. However, if the number of sources that have to be rendered in real-time is more than 26, then the hardware design processes data faster. Finally, under the scenario of 96 loudspeakers and 64 real-time sources, the FPGA is approximately 80% faster than the optimized GTX275 GPU.

C. Discussion

In Table VI, we provide the specifications of various audio-system setups that can be found in the literature on three competitive technologies - DSPs, FPGAs and GPPs. The first three rows refer to systems that utilize the BF technology. The presented approaches *calculate in real-time all filters coefficients* and are based on FPGAs and DSP processors, which require much lower power than a GPP approach. For example, in [11], the authors employ a four-element microphone array and map their design onto a V4SX55 FPGA. According to the paper, every instance of the proposed beamformer can process 43463 samples/sec and up to 7 instances can fit into the largest Virtex4 SX family. Therefore, since the sampling frequency is 16 kHz, $\lfloor(43463 \cdot 7)/16000\rfloor$ = 19 sources can be extracted in real time. In [16] the authors use an ADSP21262 processor, which does not consume more than 250 mA. Thus, since the voltage supply of ADSP21262 is 3.3 V [37], we can assume that the design power requirement is approximately 3.3 V·0.25 A = 0.825 W. Furthermore, according to the paper, the DSP is 50% utilized when processing data from two microphones at 48 kHz sampling rate, or alternatively 48000 samples/sec/input·2 inputs = 96000 samples/sec. Thus, we can assume that 192000 samples/sec can be processed in real-time with 100% processor utilization, which means $\lfloor 192000/48000\rfloor$ = 4 sources can be extracted in real-time. In [17], the authors present another low-power solution based on a TMS320C6201 DSP that has a typical power requirement of 1.7 W [38]. Unfortunately, the authors do not mention the processor utilization, thus we can not derive the maximum number of real-time sources that could be extracted.

TABLE VI
RELATED WORK SUMMARY FOR BF AND WFS IMPLEMENTATIONS.

Line	Reference	Algorithms	In-Chan	Out-Chan	Platform	Sources
1	[11]	BF	4	N/A	V4SX55	19
2	[16]	BF	2	N/A	ADSP21262	4
3	[17]	BF	16	N/A	TMS320C6201	N/A
4	Virtex4	BF	62	N/A	V4FX140	61
5	Virtex6	BF	62	N/A	6VSX475T	92
6	GTX275-40nm	BF	62	N/A	GPU	46
7	Virtex6	BF	212	N/A	6VSX475T	92
8	GTX275-40nm	BF	212	N/A	GPU	12
9	[22]	WFS	N/A	22	x86	N/A
10	[24]	WFS	N/A	24	x86	64
11	[25]	WFS	N/A	32	x86	32
12	Virtex4	WFS	N/A	384	V4FX140	233
13	Virtex6	WFS	N/A	384	6VSX475T	863
14	GTX275-40nm	WFS	N/A	384	GPU	98
15	[19]	BF, WFS	12	10	x86	N/A
16	[20]	BF, WFS	26	24	x86	N/A

For completeness reasons, in line 4 we provide the number of real-time sources that can be extracted under the maximum number of input channels supported by the largest chip of an older V4FX FPGA family, *considering only the processing time*. Under the same setup, in lines 5 and 6 we provide the number of sources that can be extracted in real-time when using the largest FPGA from the 6VSX family and the GTX275. Furthermore, in lines 7 and 8 we show the number of supported sources when using the 6VSX and GTX275 respectively under a 212 input-channel setup.

Lines 9 through 11 of Table VI summarize three systems that utilize the WFS technology to render audio sources. In [22], the authors employ an array of 22 loudspeakers and all data processing is done by a standard Linux PC. The paper mostly focuses on experiments regarding the correct sound localization and does not provide data regarding the maximum number of real-time sources that can be rendered. The next two systems are from the SonicEmotion and Iosono companies and

based also on GPPs. A single SonicEmotion system supports up to 64 sources when driving 24 loudspeakers. A single Iosono system supports up to 32 sources when driving 32 loudspeakers. In case a larger loudspeaker array is required, then additional such units must be cascaded, leading to excessive power consumption. Again for completeness reasons, in line 12 we provide the number of real-time sources supported when utilizing the maximum number of RUs that can fit to the largest FPGA of an older V4FX family, *considering only the processing time*. Under the same scenario, in lines 13 and 14, we show the maximum number of real-time sources that are supported using the 6VSX475T and the optimized GTX275.

Finally, the last two rows of Table VI present two academic systems that utilize both BF and WFS. Again, these systems are based on GPPs and constrain the number of input and output channels to 26 and 24 respectively. Furthermore, power consumption is again very high since multiple PCs are used to process all input and output channels.

Based on the discussion above and the experimental results provided in Section V, we can conclude that DSP approaches can provide a low power solution, however with limited processing power capabilities. GPP implementations can support more channels than DSPs, however they still can not accommodate complex I/O setups. Therefore, multi-core platforms that can process data concurrently are much more well-suited for advanced Immersive-Audio systems. As it was mentioned in Section II-A, the BF is a parallelizable application. Multiple instances of beamformers can process data under different source apertures and extract all audio sources concurrently. Consequently, parallel multi-core architectures are well suited for the BF application, which was confirmed by the results above. It has been shown that, a high-end GTX275, a GTX460 and a FPGA implementation that employs parallel beamformers, can process data more than an order of magnitude faster compared to a Core2 Duo at 3.0 GHz, even though the CUDA cores and the FCM-BF clocks are $\sim$2 and $\sim$10 times lower respectively. Even better results were obtained when we conducted experiments with the WFS application mapped onto multi-core architectures. From the results we observed that high-end GPUs can process WFS data more than an order of magnitude faster than the Core2 Duo. Furthermore, a highly tiled reconfigurable FCM-WFS implementation consisting of many RUs, improved performance by two orders of magnitude against the GPP.

We also considered power consumption and platform hardware cost. Power consumption increases dramatically when the number of microphones or the loudspeaker array becomes very large. For example, as indicated in Section II-B, in the cinema of Ilmenau, Germany, six desktop PCs are employed to drive a 192-loudspeaker array. Since a Core2 Duo-based PC consumes approximately 65 Watts of power when fully utilized [39], a system with 6 such PCs would require approximately $6 \cdot 65 = 390$ Watts of power *only for the processors*. In contrast, a single GTX460 GPU consumes a maximum of 160 Watts [40]. However, in this case, a less powerful host processor would suffice, since it would not be used for data processing of the algorithms. Thus a GTX460-based platform has the potential to reduce power consumption by $\sim$2.4 times. Furthermore, the entire system cost would be also reduced, since it would substitute six PCs with one only GPU-based unit. Alternatively, employing a Virtex6 FPGA would further reduce power consumption. Based on the Xilinx XPower utility, a 16-channel FCM-BF and a 1-RU FCM-WFS prototypes would consume approximately 6 and 5.3 Watts when mapped onto a Virtex6 SX475T. Therefore, we can conclude that an FPGA based Immersive-Audio design would provide the lowest power consumption solution among all tested platforms at the price of higher system cost.

VI. Conclusions

In this paper, we explored the architectural prospectives of the BF and the WFS technologies. We mapped both of them onto GPUs, and also designed custom hardware accelerators. We compared the GPU approaches against our Virtex6-based FPGA solutions and an OpenMP-annotated software run on a Core2 Duo. We also explored the processing potential of high-end GPUs and FPGAs for BF and WFS. Based on the conducted experiments, we concluded that high-end GPUs and FPGA solutions are the best choices to develop complex Immersive-Audio systems utilizing tens of input and output channels. Besides high performance, single-chip GPU and FPGA implementations can provide more power-effective solutions compared to traditional PC-based designs. Ultimately, we believe that this work can serve as a guide for choosing the correct approach and hardware platform to engineers, who consider designing high-end Immersive-Audio systems.

Acknowledgment

This research is partially supported by Artemisia iFEST project (grant 100203), Artemisia SMECY (grant 100230), FP7 Reflect (grant 248976), FP6 hArtes (IST-035143) and the Dutch Technology Foundation STW, applied science division of NWO (project DCS.7533).

References

[1] A. Berkhout, D. de Vries, and P. Vogel, "Acoustic Control by Wave Field Synthesis," in *Journal of the Acoustical Society of America*, vol. 93, May 1993, pp. 2764–2778.

[2] M. A. Gerzon, "Periphony: With-Height Sound Reproduction," in *Journal of the Audio Engineering Society*, vol. 21, 1973, pp. 2–10.

[3] B. V. Veen and K. Buckley, "Beamforming: a versatile approach to spatial filtering," in *IEEE ASSP Magazine*, vol. 5, April 1988, pp. 4–24.

[4] Bill Kapralos and Michael Jenkin and Evangelos Milios, "Audio-visual localization of multiple speakers in a video teleconferencing setting," in *International Journal of Imaging Systems and Technology*, vol. 13(1), October 2003, pp. 95–105.

[5] J. Daniel, R. Nicol, and S. Moreau, "Further Investigations of High Order Ambisonics and Wave Field Synthesis for Holophonic Sound Imaging," in *114th Convention of Audio Engineering Society*, March 2003, pp. 58–70.

[6] G. Theile and H. Wittek, "Wave field synthesis: A promising spatial audio rendering concept," in *Acoustical Science and Technology*, 2004, pp. 393–399.

[7] J. van Dorp Schuitman, "The Rayleigh 2.5D Operator Explained," Laboratory of Acoustical Imaging and Sound Control, TU Delft, The Netherlands, Tech. Rep., June 2007.

[8] P. Vogel, "Application of Wave Field Synthesis in Room Acoustics," Ph.D. dissertation, TU Delft, The Netherlands, 1993.

[9] K. Wall and G. R. Lockwood, "Modern Implementation of a Realtime 3D Beamformer and Scan Converter System," in *2005 IEEE Ultrasonics Symposium*, September 2005, pp. 1400–1403.

[10] C.-I. C. Nilsen and I. Hafizovic, "Digital Beamforming using a GPU," in *IEEE International Conference on Acoustics, Speech and Signal processing*, May 2009, pp. 609–612.
[11] Ka-Fai Cedric Yiu and Chan Hok Ho and Yao Lu and Xiaoxiang Shi and Wayne Luk, "Reconfigurable acceleration of microphone array algorithms for speech enhancement," in *Application-specific Systems, Architectures and Processors*, 2008, pp. 203–208.
[12] "Implementing a Real-Time Beamformer on an FPGA Platform," in *XCell journal*, Second Quarter 2007, pp. 36–40.
[13] Ivan Tashev and Michael L. Seltzer, "Data driven beamformer design for binaural headset," in *International Conference on Acoustice, Echo and Noise Control*, September 2008.
[14] Squarehead Technology, "http://www.sqhead.com."
[15] Acoustic Camera, "http://www.acoustic-camera.com."
[16] Zohra Yermeche and Benny Sallberg and Nedelko Grbic and Ingvar Claesson, "Real-time implementation of a subband beamforming algorithm for dual microphone speech enhancement," in *IEEE International Symposium on Circuits and Systems*, May 2007, pp. 353–356.
[17] Mark Fiala and David Green and Gerhard Roth, "A panoramic video and acoustic beamforming sensor for videoconferencing," in *IEEE International Conference on Haptic, Audio and Visual Environments and their Applications*, October 2004, pp. 47–52.
[18] E. Weinstein, K. Steele, A. Agarwal, and J. Glass, "LOUD: A 1020-Node Modular Microphone Array and Beamformer for Intelligent Computing Spaces," in *MIT/LCS Technical Memo MIT-LCS-TM-642*, April 2004.
[19] J. Beracoechea, S. Torres-Guijarro, L. García, and F. Casajús-Quirós, "On building Immersive Audio Applications Using Robust Adaptive Beamforming and Joint Audio-Video Source Localization," in *EURASIP Journal on Applied Signal Processing*, June 2006, pp. 1–12.
[20] H. Teutsch, S. Spors, W. Herbordt, W. Kellermann, and R. Rabenstein, "An Integrated Real-Time System For Immersive Audio Applications," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, October 2003, pp. 67–70.
[21] J. van Dorp Schuitman, L. Hörchens, and D. de Vries, "The MAP-based wave field synthesis system at TU Delft (NL)," in *1st DEGA symposium on wave field synthesis*, September 2007.
[22] D. Menzel, H. Wittek, G. Theile, and H. Fast, "The Binaural Sky: A Virtual Headphone for Binaural Room Synthesis," in *International Tonmeister Symposium*, October 2005.
[23] C. Kyriakakis, "Fundamental and Technological Limitations of Immersive Audio Systems," in *Proceedings of the IEEE*, vol. 86, May 1998, pp. 941–951.
[24] SonicEmotion Company, "http://www.sonicemotion.com."
[25] Iosono Company, "http://www.iosono-sound.com."
[26] NU-Tech Framework, "http://www.leaff.it/content.php?id=31."
[27] A. Lattanzi1, E. Ciavattini1, S. Cecchi, L. Romoli, and F. Ferrandi, "Real-Time Implementation of Wave Field Synthesis on NU-Tech Framework Using CUDA Technology," in *Audio Engineering Society*, May 2010.
[28] NVidia Corporation, "CUDA programming guide version 3.1.1," Tech. Rep., July 2010.
[29] R. G. Lyons, *Understanding Digital Signal Processing*. Prentice Hall, 2001.
[30] D. Theodoropoulos, C. B. Ciobanu, and G. Kuzmanov, "Wave Field Synthesis for 3D Audio: Architectural Prospectives," in *ACM International Conference on Computing Frontiers*, May 2009, pp. 127–136.
[31] M. P. McGraw-Herdeg, D. P. Enright, and B. S. Michael, "Benchmarking the NVidia 8800GTX with the CUDA Development Platform," in *High Performance Embedded Computing*, September 2007.
[32] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev, "A Reconfigurable Beamformer for Audio Applications," in *IEEE Symposium on Application Specific Processors*, July 2009, pp. 80–87.
[33] ——, "Reconfigurable Accelerator for WFS-Based 3D-Audio," in *IEEE Reconfigurable Architecture Workshop*, May 2009, pp. 1–8.
[34] ——, "A 3D-Audio Reconfigurable Processor," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 2010.
[35] Xilinx Inc., "ML605 Hardware User Guide," January 2010.
[36] International Technology Roadmap for Semiconductors, "http://www.itrs.net/home.html," 2005.
[37] Analog Devices Inc, "SHARC Processor ADSP-21262," May 2004.
[38] Texas Instruments Inc, "TMS320C62x/C67x Power Consumption Summary," July 2002.
[39] Intel Corporation, "http://ark.intel.com/Product.aspx?id=33910."
[40] NVidia Corporation, "http://www.nvidia.com/object/product-geforce-gtx-460-us.html."

Dimitris Theodoropoulos (S'06) was born in Athens, Greece. In 2003 and 2006 he obtained his Diploma (5-year degree) and M.Sc degree respectively in Electronic and Computer Engineering department of the Technical University of Crete, Greece. In 2007, he joined the Computer Engineering department of the Delft Technical University, where he is currently working towards his Ph.D. Dimitris Theodoropoulos is a member of the Technical Chamber of Greece from 2003. His current research interests include: reconfigurable computing, immersive-audio applications, embedded systems, computer architecture, cryptography.

Georgi Kuzmanov (S'95–M'05) was born in Sofia, Bulgaria. He received the M.Sc. degree in computer systems from the Technical University of Sofia, Bulgaria, in 1998, and the Ph.D. degree in computer engineering from Delft University of Technology (TU Delft), Delft, The Netherlands, in 2004. He is currently with the Computer Engineering Laboratory, TU Delft. Between 1998 and 2000, he was an R&D engineer with "Info MicroSystems" Ltd., Sofia. His research interests include reconfigurable computing, media processing, computer arithmetic, computer architecture and organization, vector processors, and embedded systems.

Georgi Gaydadjiev is currently a faculty member at the Computer Engineering Laboratory, Microelectronics and Computer Engineering Department of Delft University of Technology, The Netherlands. His research and development experience includes more than 20 years in hardware and software design. Before joining TU Delft he worked with System Engineering Ltd. in Pravetz, Bulgaria and Pijnenburg Microelectronics and Software B.V. in Vught, The Netherlands. His research interests include embedded systems design, advanced computer architectures, reconfigurable computing, hardware/software co-design, VLSI design, and computer systems testing.