Belloch Rodríguez, JA.; Gonzalez, A.; Martínez Zaldívar, FJ.; Vidal Maciá, AM. (2011). Real-time massive convolution for audio applications on GPU. Journal of Supercomputing. 58(3):449-457. doi:10.1007/s11227-011-0610-8.

Noname manuscript No.
(will be inserted by the editor)

Real-Time massive convolution for audio applications on GPU

Massive convolution on GPU

Jose A. Belloch · Alberto Gonzalez · F.J. Martínez-Zaldívar · Antonio M. Vidal

Received: date / Accepted: date

Abstract Massive convolution is the basic operation in multichannel acoustic signal processing. This field has experienced a major development in recent years. One reason for this has been the increase in the number of sound sources used in playback applications available to users. Another reason is the growing need to incorporate new effects and to improve the hearing experience. Massive convolution requires high computing capacity. GPUs offer the possibility of parallelizing these operations. This allows us to obtain the processing result in much shorter time and to free up CPU resources. One important aspect lies in the possibility of overlapping the transfer of data from CPU to GPU and vice versa with the computation, in order to carry out real-time applications. Thus, a synthesis of 3D sound scenes could be achieved with only a peer-to-peer music streaming environment using a simple GPU in your computer, while the CPU in the computer is being used for other tasks. Nowadays, these effects are obtained in theaters or funfairs at a very high cost, requiring a large quantity of resources. Thus, our work focuses on two mains points: to describe an efficient massive convolution implementation and to incorporate this task to real-time multichannel-sound applications.

Keywords Massive convolution · Multichannel audio processing · FFT · GPU ·

1 Introduction

A basic operation in multichannel acoustic signal processing is Massive Convolution. It consists in carrying out simultaneously many convolutions of different audio channels. This provides a multichannel convolution that allows to achieve with different filters well known acoustic effects like: 3D spatial sound, crosstalk cancellation, room compensation [1], loudspeakers equalization, etc. [2].

Jose A. Belloch, Alberto Gonzalez, F.J. Martínez-Zaldívar
Institute of Telecommunications and Multimedia Applications
Universidad Politecnica de Valencia Tel.: +34-96-3877007 ext 73008
E-mail: jobelrod@iteam.upv.es, {agonzal,fjmartin}@dcom.upv.es

Antonio M. Vidal
INCO2-DSIC, Universidad Politecnica de Valencia (Spain)
E-mail: avidal@dsic.upv.es

Up to now, most of these effects could be achieved only in theaters or funfairs, always using very powerful computers and consuming a large amount of energy. The use of GPU (Graphics Processing Unit) makes possible to achieve these amazing effects saving energy, and also, to obtain them in a personal computer even faster (Figure 1), as can be seen at [3] and [5], where some experiments comparing performance convolution in CPU and GPU have already been carried out using OpenGL [8].

Fig. 1: Effects which require plenty of resources can be achieved using a GPU.

However, in spite of obtaining good performance using a GPU, the fact of transferring data from/to the CPU to/from GPU avoids the execution of real-time applications. In this article, an algorithm with a pipeline structure is developed, which allows to perform a massive acoustic real-time convolution. As analyzed in this article, massive convolution requires the calculation of several FFT simultaneously. There are various libraries that implement efficient FFT algorithms. They allow to obtain the Discrete Fourier Transform of a signal either in a CPU (like MKL [9] or IPP [10]) or in a GPU (like CUFFT [11] from NVIDIA).

Multichannel convolution applications are not only based on FFT, but also they require more operations like multiplications among signals. Hence, it is crucial to configure a data structure suitable for exploiting both CUFFT NVIDIA library and different parallel operations. The paper is organized as follows. Section 2 describes the convolution algorithm and how it can be developed over a GPU. In Section 3, an efficient GPU implementation of massive convolution is presented. Section 4 analyzes the performance of a possible real-time application. Finally, some conclusions are presented in Section 5.

2 Multichannel convolution on GPU

Multichannel convolution consists in carrying out many convolutions of different channels simultaneously. Depending on the desired audio effect, different combinations can be required: different filters applied to a sound source (Figure 2), one filter applied to several sound sources, or different filters applied to different sound sources. In order to understand how multichannel convolution is organized, it is important to describe the one channel convolution first.

Let us consider x an input audio signal, h an acoustic filter (unit-impulse response) and y the desired output audio signal of our system. N, M and $L = N+M-1$ [6] will be the lengths of x, h and y respectively. The execution of the convolution using a GPU is illustrated in Figure 3. In spite of the parallelism in operations that GPU offers, the transfer time penalty prevents us from running a real time application in a GPU. Moreover, if the signal x consists of several channels, then multiple convolutions would be required. On the other hand, if a CPU is used to implement a massive convolution, all our resources would be used and no more applications could be run at the same time.

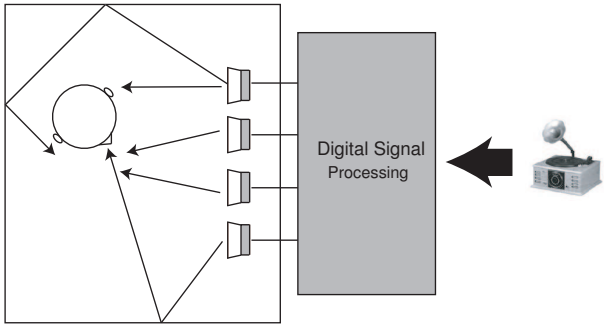

Fig. 2: Different filters applied to a sound source for audio reproduction through loudspeakers in a room

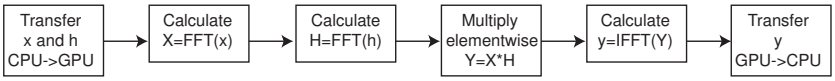

Fig. 3: Steps in order to calculate convolution of signals x and h on GPU.

2.1 Algorithm for long signals

In a real-time application, the length of signal x cannot be known a priori. Techniques are available to fragment the signal, and obtain the convolution of the whole signal from the convolution of each fragment. One of these techniques is called overlap-save [7] and it performs as follows:

1. Fragments of L samples are taken, where L is some power of two, larger than M (length of h) and at least 512 [7].
2. In the first fragment, the first $M-1$ samples will be padded with zeros.
3. From the second and following fragments, the first $M-1$ samples will be duplicated from the last $M-1$ samples of the previous fragment, see at the top of Figure 6.
4. Following the steps of the previous subsection, $y_0[n]$, $y_1[n]$, $y_2[n]$, ..., are obtained as the result of the convolution of $x_0[n]$, $x_1[n]$, $x_2[n]$, ..., with h respectively, see Figure 4.
5. From each fragment result, the first $M-1$ samples will not be valid values and will therefore be eliminated, see Figure 5.

3 Implementation on GPU

The operation described in Section 2 is applied over every signal fragment. CUFFT NVIDIA FFT library allows to execute multiple FFT 1D at the same time. Therefore, it is possible to obtain as many FFT 1D as rows of a given matrix. In order to exploit this property, it is necessary to configure a matrix signal with all the signal parts. Figure 6 illustrates the formation of this signal matrix. CUDA toolkit versions [12] enable the use CUFFT [11] with the property *concurrent copy and execution*. Therefore, the latency of transferring data from the CPU to the GPU and vice versa can be overlapped by computation time. This will enable not only high speedup of the convolution, but also, the use of real-time applications.

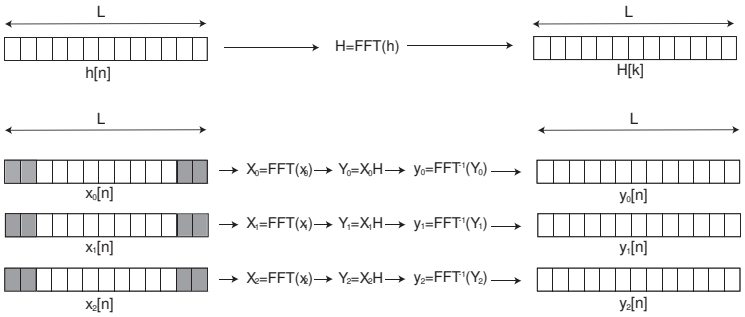

Fig. 4: Convolution of each fragment is calculated following the convolution theorem [6].

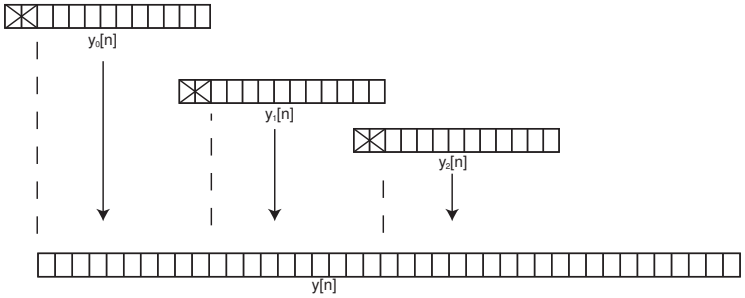

Fig. 5: As long as convolutioned fragments are obtained, output signal *y* is being formed.

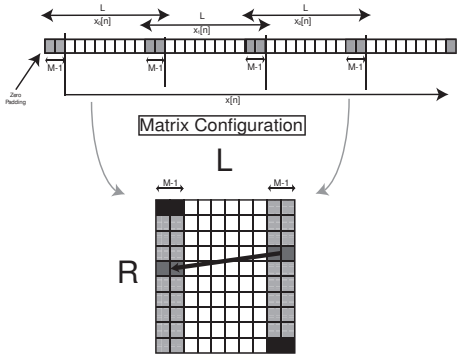

Fig. 6: A signal matrix is built from signal parts.

Therefore, the configured signal matrix of Figure 6 with R rows and L columns could be considered as a buffer, which is being built as the incoming audio samples arrive. The first $M-1$ values of one row will coincide with the last $M-1$ values of the previous row, except for the first configured matrix at the algorithm beginning whose first $M-1$ values from the first row will be zeros. The last $M-1$ samples from the last row of the matrix will be kept in an internal buffer in order to occupy the first $M-1$ positions of the next matrix to be filled. The unit-impulse response h will have been sent to the GPU before sending the first

matrix. As shown in Figure 4, and described in sections 2 and 3, vector h will be padded with zeros up to L samples (length of each fragment of signal x), then a FFT will be carried out obtaining vector H, and finally an elementwise multiplication with each fragment of X (x in the frequency domain) will be done.

To carry out operations on GPU, since signal x is configured as a matrix, an h-matrix must be also configured. It consists of R replications of vector h. Over the GPU, FFT function from the CUFFT library is applied to both matrices, then an elementwise multiplication is done between them (Figure 7), and finally, the inverse FFT function from CUFFT is applied again over the result matrix. Thus, time samples of output signal are obtained.

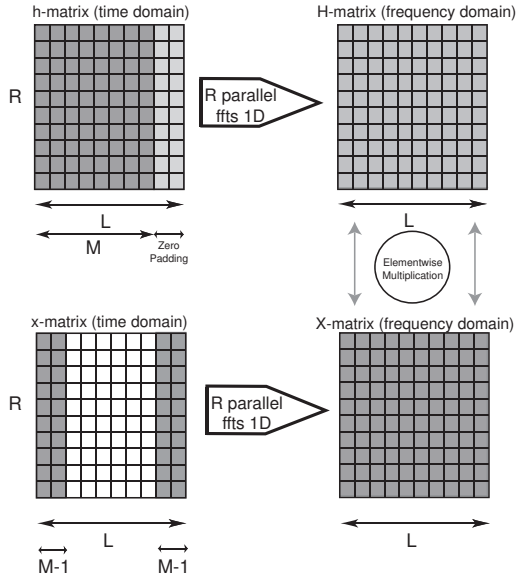

Fig. 7: FFT function from CUFFT library is applied to signal matrix and h-matrix, then an elementwise multiplication is done between them.

3.1 Scalability from one channel to multichannel

It is obvious that the hearing effects explained previously cannot be represented by either one filter or a single signal. Thus, when dealing with a stereo signal (two audio channels) or maybe with a four-channel audio signal, resources will be shared, as shown in Figure 8.

3.2 Pipelined Algorithm

The *concurrent copy and execution* property enables multichannel convolution using a four-stages pipelined model. This model uses the asynchronous transfer of matrix signals from

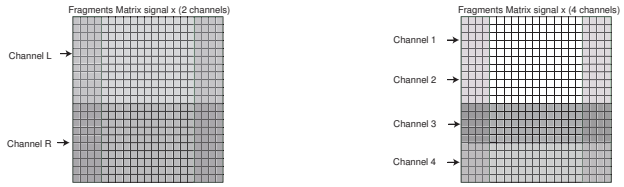

Fig. 8: The signal matrix on the left shows 2-channel resource sharing, whereas the signal matrix on the right shows 4-channel resource sharing

CPU to GPU and vice versa while other tasks are executed in parallel. In [12] it is recommended to use different *streams* in order to execute different tasks. In our case, the value of streams is between 1 and 4, *Stream 0* is not recommended for asynchronous operations.

At the beginning of the algorithm, h-matrix is configured and sent to the GPU. Then, the first buffer begins to be built. As in [4], we use a buffer size of 32 x 512. We will call this buffer: A-buffer. Using asynchronous transfer, while A-buffer is sent to GPU by *stream 1*, another buffer, B-buffer, is built simultaneously by *stream 2*. Then, *stream 1* executes the computations described in previous subsections between h-matrix and A-buffer (signal matrix), while B-buffer is transferred from CPU to GPU by *stream 2*, and a new buffer (C-buffer) is built by *stream 3*. Finally, a new D-buffer is built by *stream 4*, while C-buffer is transferred from CPU to GPU by *stream 3*, execution in GPU is carried out on B-Buffer by *stream 2* and A-buffer is transferred back to CPU by *stream 1*. A rebuilding of output signals (Section 2.1) from different channels are carried out on A-buffer, then output signals are obtained. Afterwards, A-buffer is used again. Thus, four buffers (A, B, C and D) are being used cyclically. Figure 9 shows all the stages with the time required by each of them. It must be pointed out that, the block called ”Rebuilding signals” begins once the whole buffer is back to CPU in order to avoid race condition. So, transfer to CPU and rebuilding go in the same block in the algorithm. Also, time blocks of ”Get Signal Matrix” and ”Rebuilding Signals” (Figure 9) include latency times of samples captured from external buffer A/D and delivered to external buffer D/A respectively, as it can be seen in [14], data-sheet of AC97 SoundMAX audio codec.

4 Results

Two main tests have been carried out to verify massive convolution on GPU. The first test concerns the speed-up achieved when the pipelined algorithm of Figure 9 is compared with a basic convolution algorithm, shown in Figure 3, using a signal x and an impulse-response h made up of 176400 samples and 220 coefficients respectively. The size of the configured signal matrix x was 32 x 512 elements. The time employed using basic algorithm is 13330 ms whereas in the pipelined algorithm is 625.92 ms . Therefore using the last configuration, the time spent can be halved.

The most significant test resolves around the number of audio channels that can be managed by a GPU to carry out a massive convolution. In a real time audio application, transfer and computation on GPU must spend less time than filling the sample’s data buffer. This time depends on the rate of the incoming samples, what is known as sample frequency. CD quality has an audio sample frequency of 44.1 kHz. It means that 44100 samples per

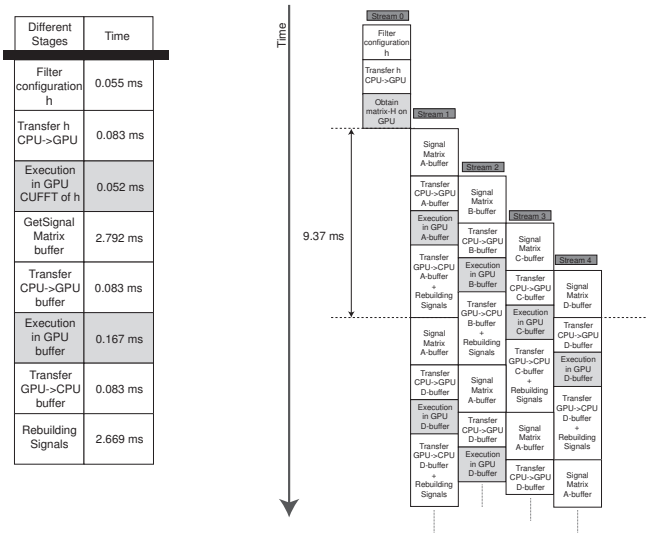

Fig. 9: Pipeline algorithm using different *streams* provided by the *concurrent copy and execution* property of GPU. On the left side, times spent by all stages of the algorithm

channel arrive within one second. Taking into account that one sample of one channel arrives every 1/44100 s, different numbers of channels can be managed by a GPU, depending on the buffer size.

Figure 9 shows that time spent on the algorithm (fill buffer, transfer CPU⇒GPU, computing, transfer CPU⇐GPU, empty buffer) is 9.37 ms. This number comes from the sum of all the steps executed by one *stream* taking into account some conflicts among adjacent *streams* when more than a transfer CPU⇔GPU exists simultaneously, as documentation from NVIDIA [13] explains. The same algorithm has been implemented sequentially and executed using a core of CPU intel i7, spending 14.98 ms.

Real samples in one row of the buffer matrix will be $L-(M-1)$ because first $M-1$ samples will be zero or duplicated (In our test, 293 samples, which arrive at 1/44100 s each). Table 1 shows that processing on GPU allows managing up to 16 audio channel simultaneously using a matrix buffer of 32 x 512. If one row of the buffer were dedicated to one channel, then, the executing time of 9.37ms would be larger than the filling buffer time of 6.6 ms thus the application would not work properly in real-time. Many incoming samples would not be processed because the A-buffer (Figure 9) would not be available to be filled of samples.

5 Conclusions

The *concurrent copy and execution* CUDA property allows to configure a pipelined algorithm, which can be used for carrying out a massive convolution. This algorithm offers much better performance than the classical algorithm of the convolution over GPU. The main advantage is that it is a scalable algorithm, even when the incoming signal x has several channels or there is more than one filter or effect to be carried out over the signals.

Table 1: Number of possible audio channels in the application using a matrix buffer of 32 x 512

Number of channels	Occupacy of rows per channel	Time employed filling buffer	Use of GPU (%)	Availability
1	32	212.6 ms	4.4%	Yes
2	16	106.3 ms	8.8%	Yes
4	8	53.15 ms	17.6%	Yes
8	4	26.9 ms	35.2%	Yes
16	2	13.2 ms	70.5%	Yes
32	1	6.6 ms	141%	No

As the results show, GPU lets dealing with 16 audio channels. With this pipelined algorithm it is clear that, with only one GPU, applications like 3D spatial sound can be achieved. Moreover the use of a single GPU provides energy saving, because the large computers used nowadays in funfairs or theaters would no longer be required to develop audio applications.

Furthermore, using GPU frees up CPU resources, providing better performance and more importantly, opening up a new way of implementing audio applications where GPUs have not previously been used before.

Acknowledgements This work was partially supported by the Spanish Ministerio de Ciencia e Innovacion (Projects TIN2008-06570-C04-02 and TEC2009-13741), Universidad Politecnica de Valencia through PAID-05-09 and Generalitat Valenciana through project PROMETEO/2009/2013

References

1. S. Spors, R. Rabenstein, W. Herbordt, Active listening room compensation for massive multichannel sound reproduction system using wave-domain adaptive filtering, J. Acoust Soc. Am., vol 122, pag 354-369, (2007)
2. Y. Huang, J.Benesty and J. Chen, Generalized crosstalk cancellation and equalization using multiple loudspeakers for 3D sound reproduction at the ears of multiple listeners, IEEE Int. Conference on Acoustics, Speech and Signal Processing page 405-408, Las Vegas, USA (2008)
3. B. Cowan, and B. Kapralos. Spatial sound for video games and virtual environments utilizing real-time GPU-based convolution. In Proceedings of the ACM FuturePlay 2008 International Conference on the Future of Game Design and Technology. Toronto, Ontario, Canada, November 3-5, (2008).
4. J.A. Belloch, A. M. Vidal, F.J.Martinez-Zaldivar and A. Gonzalez, Multichannel acoustic signal processing on GPU, Procedings of the 10th International Conference on Computational and Mathematical Methods in Science and Engineering, Vol 1, Pg 181-187, Almeria, Spain, June 26-30, (2010).
5. Brent Cowan and Bill Kapralos. GPU-Based One-Dimensional Convolution for Real-Time Spatial Sound Generation. Scholary Journals, ISSN 1923-2691, Vol 3, No 5, (2009).
6. S.S. Soliman, D.S.Mandyam and M.D. Srinath, Continuous and Discrete Signals and Systems, Prentice Hall, ISBN:0135184738 (1997)
7. A.V. Oppenheim A.S.Willsky and S.Hamid Nawab, Signals and Systems, Prentice Hall, ISBN:0138147574
8. openGL: “http://www.opengl.org/”
9. MKL library: “http://software.intel.com/en-us/intel-mkl/”
10. MKL library: “http://software.intel.com/en-us/intel-ipp/”
11. CUFFT library: “http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CUFFT_Library_3.1.pdf”
12. CUDA Toolkit 3.1: “http://developer.nvidia.com/object/cuda_3_1_downloads.html”
13. CUDA Toolkit 3.2: “http://developer.nvidia.com/object/cuda_3_1_downloads.html”
14. Datasheet of AC97 SoundMAX Codec: “http://www.xilinx.com/products/boards/ml505/datasheets/87560554AD1981B_c.pdf”