3-2013

Audio convolution on GPUs: a follow-up

Davide Andrea Mauro
Marshall University, maurod@marshall.edu

Audio convolution on GPUs: a follow-up

Davide A. Mauro[1,2]
[1] *Laboratorio di Informatica Musicale (LIM), Universitá degli Studi di Milano, Milan, Italy*
[2] *Institut Mines-Télécom, TÉLÉCOM ParisTech, CNRS-LTCI, Paris, France*
davide-andrea.mauro@telecom-paristech.fr

Introduction

This paper focuses on the use of GPGPU (General-Purpose computing on Graphics Processing Units) for audio processing. This is a promising approach to problems where a high parallelization of tasks is desirable. Within the context of binaural spatialization we will develop a convolution engine having in mind both offline and real-time scenarios, and the support for multiple sound sources. Details on implementations and strategies used with both dominant technologies, namely CUDA and OpenCL, will be presented highlighting both advantages and issues. Comparisons between this approach and typical CPU implementations will be presented as well as between frequency (FFT) and time-domain approaches. Results will show that benefits exist in terms of execution time for a number of situations.

Convolution Engines

Even if the process is well known and understood in terms of mathematics, the realization of implementations that work in real-life scenarios is not trivial. One of the greatest obstacle is the computational complexity that convolution requires both in the time and frequency domain approaches. This means that the problem could be theoretically solved but the computer architecture does not allow it to be solved in a reasonable time for some practical cases of interest. As shown in Figure 1 the system requires as input an anechoic signal (monophonic) and a impulse response (stereo) and the overall output will be two channel spatialized sound that can feed both headphones or loudspeakers (with crosstalk cancelation algorithms [3]). For a review of the state of the art in convolution engines please refer to [7].

Overlap-add algorithm

Since the size of the filter kernel can become very high, it is not convenient to use a single window to transform the entire signal so a number of methods can be implemented to overcome this. We choose to use a method called Overlap-add (OA, OLA). It is an efficient way to evaluate the discrete convolution of a very long signal $x[n]$ with a finite impulse response (FIR) filter $h[n]$. The concept is to divide the problem into multiple convolutions of $h[n]$ with short segments of $x[n]$:

$$y[n] = x[n]*h[n] := \sum_{m=-\infty}^{\infty} h[m]x[n-m] = \sum_{m=1}^{M} h[m]x[n-m] \tag{1}$$

where $h[m] = 0$ for m outside the region $[1, M]$.

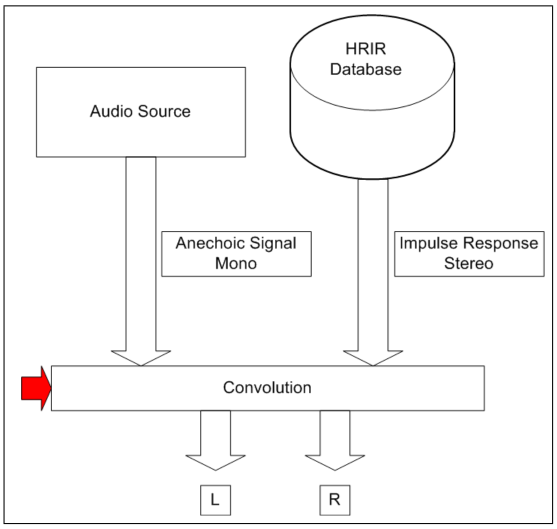

Figure 1: The workflow diagram of the system.

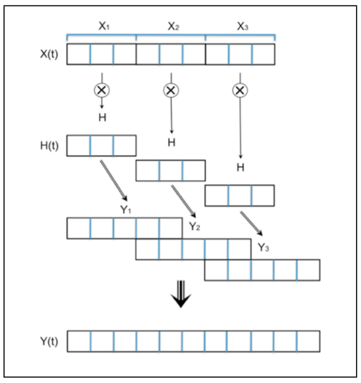

Figure 2: Schematic view of the overlap-add convolution method.

$$x_k[n] := \begin{cases} x[n+kL] & n = 1,\ 2,\ \ldots,\ L \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where L is an arbitrary segment length.

$$x[n] = \sum_k x_k[n-kL] \tag{3}$$

So $y[n]$ can be written as a sum of convolutions:

$$y[n] = \left(\sum_k x_k[n-kL]\right) * h[n] = \sum_k (x_k[n-kL] * h[n]) \tag{4}$$

The method is depicted in Figure 2

It is particularly interesting for our tasks since it works on independent pieces of input and thus is well suited for a parallelized approach such as one that employs a GPU.

Implementations

In order to make comparisons with the GPU implementations that we will present we need a reference implementation that can serve as a basis in terms of execution time and bitwise precision. For this reason three different prototypes have been developed that use different algorithms.

The first two prototypes are Matlab scripts that use both a Time Domain and a Frequency Domain approach. Since the computational complexity for the Time Domain approach is $O(n^2)$ this can not be used when the filter kernels are big. In our experiments, according to a Max/MSP implementation that will be introduced in the following section, we choose to limit the size to 256 samples.

The frequency domain implementation (presented in [6]) will be used to validate the results in terms of bitwise precision. Since Matlab is mainly intended as a prototyping environment there is no focus on performance and every other implementation can outperform our Matlab testbase by orders of magnitude. Moreover, this implementation works only in "direct mode"; this implies that a single FFT is performed for the entire signal and therefore the algorithm may not be applicable for long sequences due to memory constraints or implementation limits.

One of the CPU implementation is written in C++ and is based on the FFTW3 library (see [5]). It is based on the architecture previously presented and implements both modalities (Direct and OLA) previously discussed.

The FFTW library itself is based on Cooley-Tukey algorithm [4]. As presented by the authors, the interaction of the user with FFTW occurs in two stages: planning, in which FFTW adapts to the hardware, and execution, in which FFTW performs useful work for the user. To compute a DFT, the user first invokes the FFTW planner, specifying the problem to be solved. The problem is a data structure that describes the "shape" of the input data - array sizes and memory layouts - but does not contain the data itself. In return, the planner yields a plan, an executable data structure that accepts the input data and computes the desired DFT. Afterwards, the user can execute the plan as many times as desired.

A CUDA convolution engine

For the CPU implementation with CUDA we were able to implement both Direct and OLA algorithm. We consider the benefits of both approaches in the following section while presenting performance comparisons. For FFT we use a library called CUFFT which is actually based on FFTW3 library with some other optimizations specifically designed for GPUs. One of the current issue is the CUFFT limit of 64 millions of points.

An OpenCL convolution engine

The FFT used is based on Apple implementation (`http://developer.apple.com/library/mac/#samplecode/OpenCL_FFT/Introduction/Intro.html`). One of the current limitations is that the factorization algorithms works only for powers of 2 (radix-2). So the payload should be adapted to make the sum with the length of the filter kernel to be the closest greater power of 2.

The cGPUconv prototype

From a number of the previously cited prototypes we derived a single application that allows the user to choose between a CPU- or a GPU-based algorithm and between a direct mode (a single window for the entire signal) and an Overlap-add mode. It is structured as a "wrapper" around the single module that has the capability of opening audio files and writing them back to disk thanks to libsndfile (see [2]). There is also an early-stage support for realtime convolution acquiring sound from audio input and playing it back through audio output. It is a command line tool that compiles and executes both on Microsoft Windows, Apple OSX, and Linux applications as long as they have, or there exists a version of:

- Boost Library;
- PortAudio;
- Libsndfile for I/O;
- FFTW3 library for CPU implementation;
- CUDA (Versions 4 or 5) Framework;
- OpenCL (1.1) driver.

The program can be adapted removing functionalities provided by subsets of the previous requirements by removing the components that make use of those prerequisites. The source code is available from the author at
`https://code.google.com/p/cgpuconv/`.

Performance Comparisons

Performances of these algorithms depends on the size of input. Therefore, to characterize the "trade-off", we tested them with different input sizes. We then compute the time spent on the convolution procedure, excluding the load procedure that reads from audio files and the write to disk procedure for the results, which are collateral to our primary goal. A special case is represented by the first execution for both the CUDA and OpenCL implementation where for the former there exists some extra time devoted to the load of the environment while for the latter, apart from the aforementioned setup, we have to take into account the time that the driver allocate to compile kernel functions.

The algorithms were tested on a number of different platform:

- A1: Apple Macbook Pro 13.3" (MacBookPro5,5), OSX 10.6.8. Intel Core 2 Duo processor @2,53 GHz, 8 GB Ram, NVIDIA GeForce 9400GM 256 MB vRAM shared memory. OpenCL 1.1, CUDA 4.0.
- A2: Asus M50S, Ubuntu 12.10 32bit. Intel Core 2 Duo processor @2,50 GHz, 4 GB Ram NVIDIA

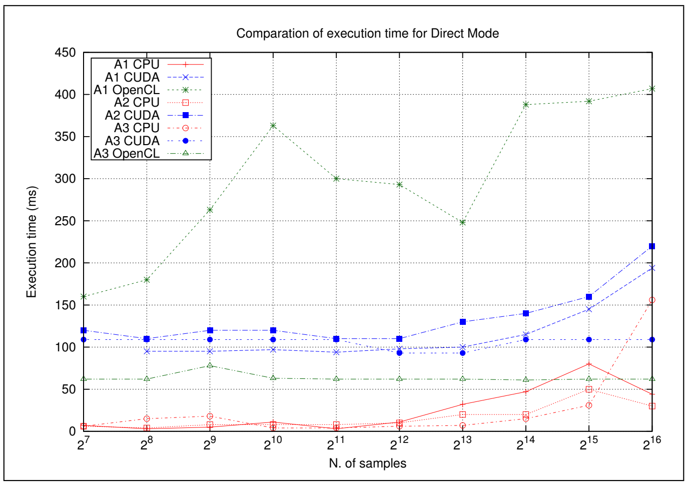

Figure 3: Execution time for Direct mode depending on input size. Part 1

GeForce 9500M G, 512 MB vRAM dedicated memory. CUDA 5.0.

- A3: ASUS CG8250 Windows 8 64bit. Intel Core i7-2600 processor @3,40 GHz, 8 GB Ram NVIDIA GeForce GTX 560 Ti, 1024 MB vRAM dedicated memory. OpenCL 1.1, CUDA 5.0.

All the audio files are high quality PCM uncompressed files and have a sample rate of 96 kHz and a quantization word of 24 bit. With this bit depth the theoretical dynamic range is $\sim$ 144 dB.

For each algorithm we measured the difference computed between the signal under test and the reference (coming from the Matlab implementation) with a phase inversion. So the difference on a sample by sample basis gives us a new signal that can be used as a degree of similarity between the two original signals. For each and every proposed approach this signal is below -122 dB FS (dB on the full scale) meaning there is no practical difference, and the result is in the order of magnitude of the noise floor.

Coming to the execution time of the algorithms we propose a summary of the results presented in Figures 3 - 4, and 5 - 6. Results are depicted as a function of the number of input samples, averaged over 100 runs, results are split in two parts for better clarity.

The exploit of CUDA 5 framework, as well as the new architectures supported by PCIxpress 3.0 standard show the continuous improvements that can be obtained using GPGPU. We also present in Table 1 results for a “real-case scenario”. We have a violin sound that is three minutes long and a reverberant impulse response of 1 second (sample rate 96kHz):

- Input: 17703123 samples (∼3’10”)
- Kernel: 96000 (∼1”)

Please note that “-” occurs when there is not enough free RAM / vRAM to handle the data. The idea behind this work is to have a system that can run on most home computer so the relatively old and low

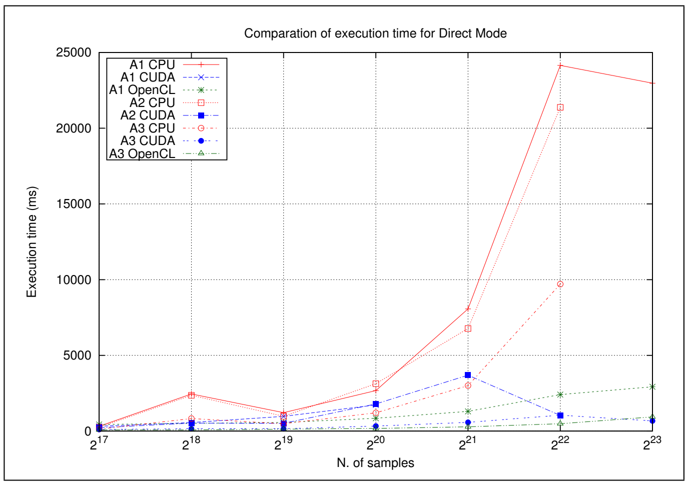

Figure 4: Execution time for Direct mode depending on input size. Part 2

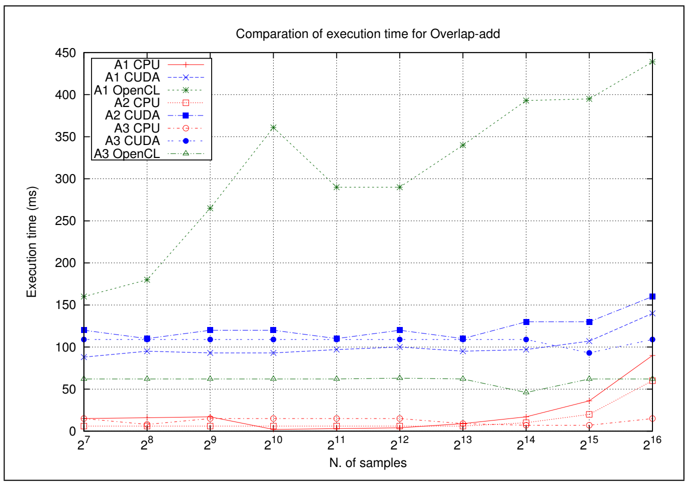

Figure 5: Execution time for Overlap-add depending on input size. Part 2

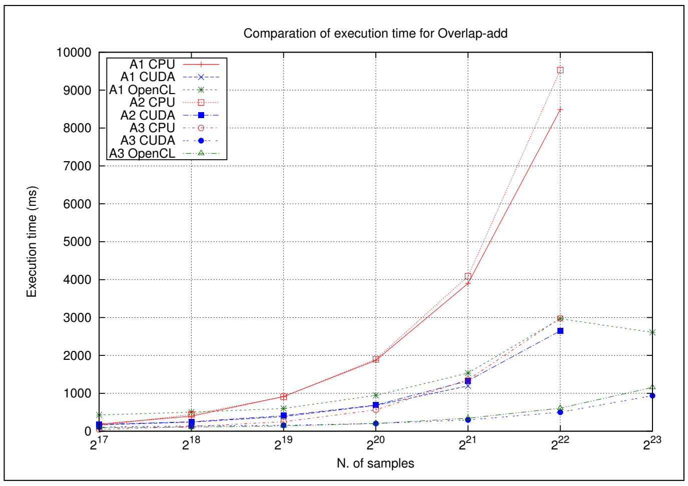

Figure 6: Execution time for Overlap-add depending on input size. Part 2

(A1/A2/A3) (ms)	Direct	OLA
CPU	-/-/-	9699/10160/3543
CUDA	-/-/-	6181/ 5930/1381
OpenCL	7486/-/658	6699/-/1526

Table 1: Performance comparisons. Time in ms.

powerful graphic card is a good example of what can be achieved with standard equipment. There are difference between implementations and this can be explained by the different way of encoding real and complex numbers. Also note that there does not exist a concept of "paging" for video RAM so if a structure is too big to fit in memory there is no automatic way to handle the situation.

Conclusions and Future Works

In this paper we presented a number of prototypes that are suitable for spatialization of sounds exploiting the potentialities of GPUs. Some issues are still present but we want to point out that the basic concepts here expressed are valid and mark a profitable direction.

Performance results suggest that for a number of real case applications there are benefits that can be at least of 1/3 of the execution time, compared to the reference CPU implementation, using old framework and outdated hardware. and can be further improved with other GPU-specific, but not hardware specific, optimizations. Benefits are increasingly evident as the size of the filter kernel grows and this is particularly useful for convolution with long reverberant impulse responses (e.g. BRIRs) that can be employed in order to render real environments.

Some of the limitations previously presents, in CUDA 4.0 framework, or like bandwidth with PCIxpress 2.0 standard are overcame by new architecture that allows to stream data back and forth from the GPU in realtime and solve a number of the previous issues.

Further improvements will consider the application of partitioned convolution, a technique first introduced by Armelloni, Giottoli, and Farina [1]. It is also possible to partition in a non-uniform manner for latency reduction purposes [9]. [8] also used this algorithm for an efficient FFT implementation on GPU.

Acknowledgments

The author would like to thank Lorenzo Picinali and Brian F.G. Katz for extensive conversations concerning these researches, and Andrea Trentini for the remote host for testing purposes. This work has been partially funded by the Enhanced Music Interactive Platform for Internet User (EMIPIU) project and by the European Commission under contract "FP7-287723 REVERIE".

References

[1] E. Armelloni, C. Giottoli, and A. Farina. Implementation of real-time partitioned convolution on a DSP board. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 71–74. IEEE, 2003.

[2] E. Castro Lopo. Libsndfile [computer software]. *Retrieved December*, 28:2005, 2005.

[3] E. Y. Choueiri. Optimal crosstalk cancellation for binaural audio with two loudspeakers. 2011.

[4] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*, 19(90):297–301, 1965.

[5] M. Frigo and S. Johnson. The design and implementation of FFTW3. *Proc. IEEE (Special Issue on Program Generation, Optimization, and Platform Adaptation)*, 93:216–231, 2005.

[6] D. A. Mauro. Effetti della distanza nella spazializzazione e localizzazione binaurale. B.A. Thesis, Università degli Studi di Milano, July 2006.

[7] D. A. Mauro. *On Binaural Spatialization and the use of GPUs for audio processing.* PhD thesis, Università degli Studi di Milano, March 2012.

[8] M. Rush. Modeling a GPU-based Convolution Engine EEC 289Q.

[9] F. Wefers and M. Vorländer. Potential of non-uniformly partitioned convolution with freely adaptable fft sizes. In *Audio Engineering Society Convention 133*, 2012.