Spring 4-22-2012

Audio convolution by the mean of GPU: CUDA and OpenCL implementations

Davide Andrea Mauro
Marshall University, maurod@marshall.edu

Recommended Citation

Mauro D.A. Audio convolution by the mean of GPU: CUDA and OpenCL implementations. Proceedings of the Acoustics 2012 Nantes Conference, 23-27 April 2012, Nantes, France: pp.2863-2868, 2012.

ACOUSTICS 2012

Audio convolution by the mean of GPU: CUDA and OpenCL implementations

D.A. Mauro

Laboratorio di Informatica Musicale (LIM), Dipartimento di Informatica e Comunicazione (DICo), Università degli Studi di Milano, Via Comelico 39/41, 20135 Milano, Italy
mauro@dico.unimi.it

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.

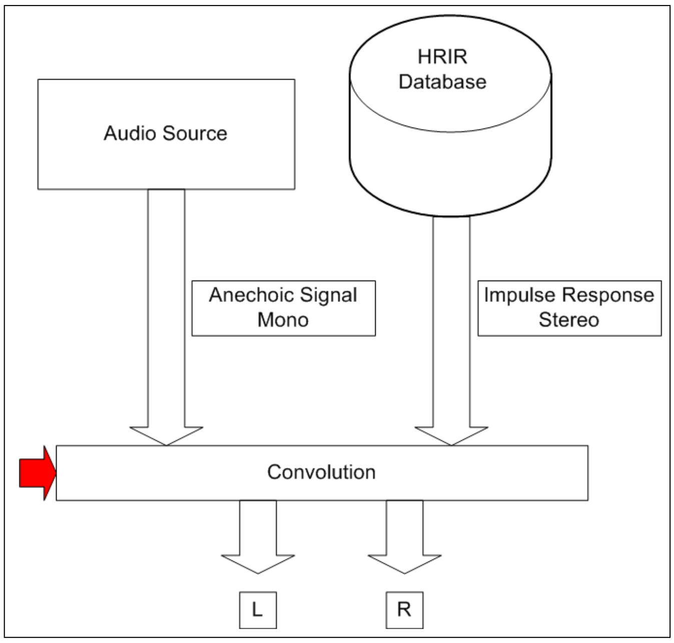

Figure 1: The workflow diagram of the system.

1 Introduction

We first introduce the core of the work in terms of conceptualization and development of a model. 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.

2 Convolution Engines

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 [2]).

We will focus on implementations of this system thanks to modern GPGPU techniques.

2.1 State of the Art

In the literature there are other systems that aim at realizing systems that achieve real-time auralization, or augmented reality. We present a brief sketch of the opportunities and the techniques employed. It is worth to cite the work of Kapralos et al. presented in [4] and [5] where the authors apply GPGPU techniques to solve the problem of convolution. The main differences are that the authors use a time domain implementation that exploits the use of OpenGL in order to process audio data. This means basically that they need to tweak the system to threat audio data as RGB bitmaps.

- TConvolutionUB~: A Max/MSP external patch from Thomas Resch that extends the possibilities given by the *buffir~* object allowing convolution with a filter that has more than 255 points.
- SIR2: An easy to use native audio-plugin to use for high quality reverberation. It's available for the plugin formats VST and AudioUnit. Its use can be stretched from a convolution reverb to a convolution engine for auralization given the flexibility of the program itself.
- djbfft: A library for floating-point convolution. The current version provides power-of-2 complex FFTs, real FFTs at twice the speed, and fast multiplication of complex arrays. Single precision and double precision are equally supported.
- BruteFIR: An open-source convolution engine, a program for applying long FIR filters to multi-channel digital audio, either offline or in realtime, by Anders Torger [8]. Its basic operation is specified through a configuration file, and filters, attenuation and delay can be changed at runtime through a simple command line interface. The author states that the FIR filter algorithm used is an optimized frequency domain algorithm, partly implemented in hand-coded assembler, thus throughput is extremely high. In real-time, a standard computer can typically run more than 10 channels with more than 60000 filter taps each. It makes use of the partitioned convolution and overlap-save methods that are introduced in the following subsection.
- AlmusVCU: From the author of BruteFIR this is a complete system that aims at an integrated environment for sound spatialization. It has been designed primarily with Ambiophonics in mind and contains all processing needed for a complete Ambiophonics system.
- Aurora Plugin: From Angelo Farina, is a suite of plugins for Adobe Audition: room acoustical impulse responses can be measured and manipulated, for the recreation of audible, three-dimensional simulations of the acoustical space.

2.2 Convolution in the Time Domain

This approach can be mathematically described by the formula:

$$y(k) = \sum_{j=1} x_1(j)x_2(k-j+1) \qquad (1)$$

Where x_1 and x_2 are the input sequences of length m and n and y is the output sequence of length $k = m + n - 1$.

When $m = n$, which is the normal case for other implementations, this gives:

$$\begin{aligned} w(1) &= u(1)v(1) \\ w(2) &= u(1)v(2) + u(2)v(1) \\ w(3) &= u(1)v(3) + u(2)v(2) + u(3)v(1) \\ &\cdots \\ w(n) &= u(1)v(n) + u(2)v(n-1) + \cdots + u(n)v(1) \\ &\cdots \\ w(2n-1) &= u(n)v(n) \end{aligned} \tag{2}$$

The computational complexity for the time domain approach is $O(n^2)$.

This is the underlying approach to every other method. Implementing a FIR (Finite Impulse Response) filter is obviously the easiest idea but as can be seen from the complexity as the input size increase it could become impossible to process data in real-time.

2.3 Convolution in the Frequency Domain

Thanks to the convolution theorem we can express the convolution of two sequences as the multiplication of their Fourier transforms. Here the general layout for the frequency domain approach is introduced. The approach that can be schematized as follows:

- Zero-Pad input vectors x_1 and x_2 of length m and n so the length of the sequences becomes $m + n - 1$;
- Perform FFT of the input vectors;
- Perform the pointwise multiplication of the two sequences;
- Perform the IFFT of the obtained sequence.

The computational complexity for the frequency domain approach is $O(n \log(n))$.

2.3.1 Overlap-add algorithm

Since the size of the input 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 $x[n]$:

$$y[n] = x[n] * h[n] := \sum_{m=-\infty}^{\infty} h[m]x[n-m] = \sum_{m=1}^{M} h[m]x[n-m] \tag{3}$$

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

$$x_k[n] := \begin{cases} x[n+kL] & n = 1,\ 2,\ \ldots,\ L \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

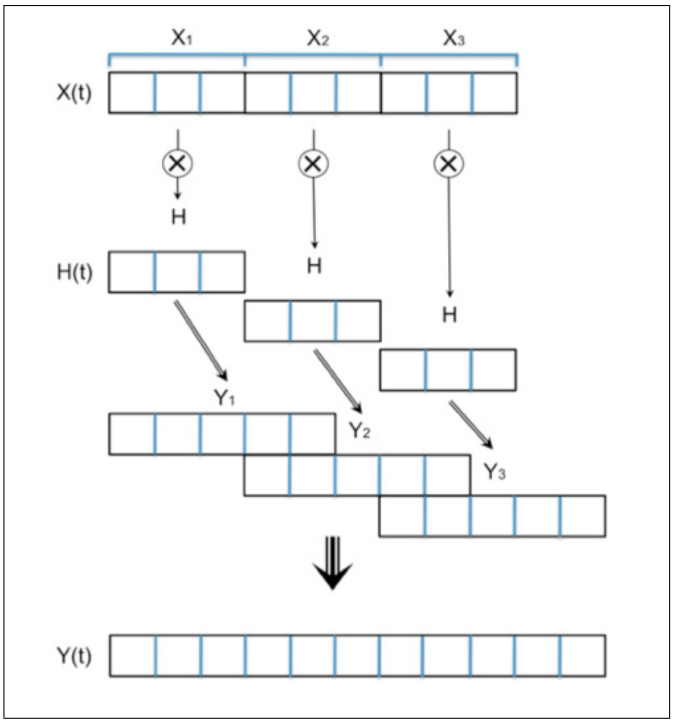

Figure 3: Schematic view of the overlap-add convolution method.

where L is an arbitrary segment length.

$$x[n] = \sum_k x_k[n - kL] \tag{5}$$

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

$$y[n] = \left(\sum_k x_k[n-kL]\right) * h[n] = \sum_k (x_k[n-kL] * h[n]) \tag{6}$$

The method is depicted in Figure 3

It is particularly useful 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.

3 Reference CPU 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 [7]) 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

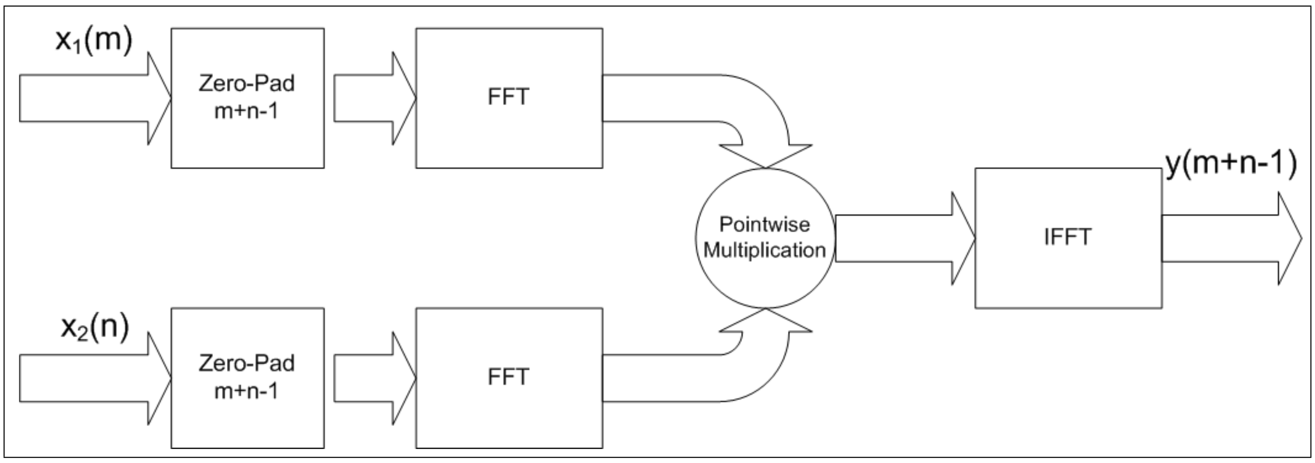

Figure 2: A scheme of convolution in frequency domain.

not be applicable for long sequences due to memory constraints or implementation limits. Source code for both the Matlab implementations are available from the author.

The last CPU implementation is written in C++ and is based on the FFTW3 library (see [6]). It is based on the architecture presented in Figure 2 and implements both modalities (Direct and OLA) previously discussed.

The FFTW library itself is based on Cooley-Tukey algorithm [3]. 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.

3.1 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.

3.2 An OpenCL convolution engine

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.

4 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 [1]). It is a command line tool that compiles and executes both on Microsoft Windows, Apple Mac OS X, and Linux as long as they have, or there exists a version of:

- Libsndfile for I/O;
- FFTW3 library for CPU implementation;
- CUDA Framework;
- OpenCL driver.

The program can be adapted by removing functionalities provided by any subset of the previous requirements by removing the components that make use of that prerequisite. The source code is available from the author at
`http://www.lim.dico.unimi.it/CGPUconv`.

4.1 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. To make a reliable comparison we choose to use as input signals a logarithmic sine sweep and its TRM (time reversal mirror) so the output should be the δ function (Dirac delta function) or, to be more precise, the limited bandwidth approximation of the sinc (*sinus cardinalis*) function.

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \tag{7}$$

$$\int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{8}$$

$$sinc(x) = \frac{\sin(x)}{x} \tag{9}$$

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.

	Direct	OLA
CPU	-	9699
CUDA	-	6181
OpenCL	7486	6699

Table 1: Performance comparisons. Time in ms.

The algorithms were executed on an OS X 10.6.8 equipped Apple Macbook Pro 13.3" (MacBookPro5,5), Intel Core 2 Duo processor @2,53 GHz, 8 GB Ram, NVIDIA GeForce 9400GM VRAM 256 MB shared memory. OpenCL drivers are provided by the operating system (1.1 compatible), and the CUDA framework is version 4.0.

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 ~ 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 4, 5.

Results are depicted as a function of the number of input samples, averaged over 100 runs.

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 s (sample rate 96 kHz)

- Input: 17703123 samples (~3'10")
- Kernel: 96000 (~1")

Please note that "-" occurs when there is not enough free video RAM to handle the data. The idea here is to have a system that can run on most home computer so the relatively old and low 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.

5 Summary and Discussion of the results

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) 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.

Acknowledgments

The author gratefully wish to acknowledge their stimulating discussions with G. Haus director of the Laboratory of Music Informatics (LIM) of the Università degli Studi di Milano and the many researchers and graduate students of LIM. The author also would like to thank Lorenzo Picinali and Brian F.G. Katz for extensive conversations concerning these researches.

This work has been partially funded by the *Enhanced Music Interactive Platform for Internet User (EMIPIU)* project.

References

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

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

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

[4] 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*, pages 166–172, Toronto, Ontario, Canada, November 3-5 2008.

[5] B. Cowan and B. Kapralos. Real-time GPU-based convolution: a follow-up. In *Proceedings of the ACM FuturePlay @ GDC Canada 2009 International Conference on the Future of Game Design and Technology*, pages 25–26, Vancouver, British Columbia, Canada, May 12–13 2009.

[6] 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.

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

[8] A. Torger. BruteFIR - an open-source general-purpose audio convolver. *http://www.ludd.luth.se/torger/brutefir.html.*

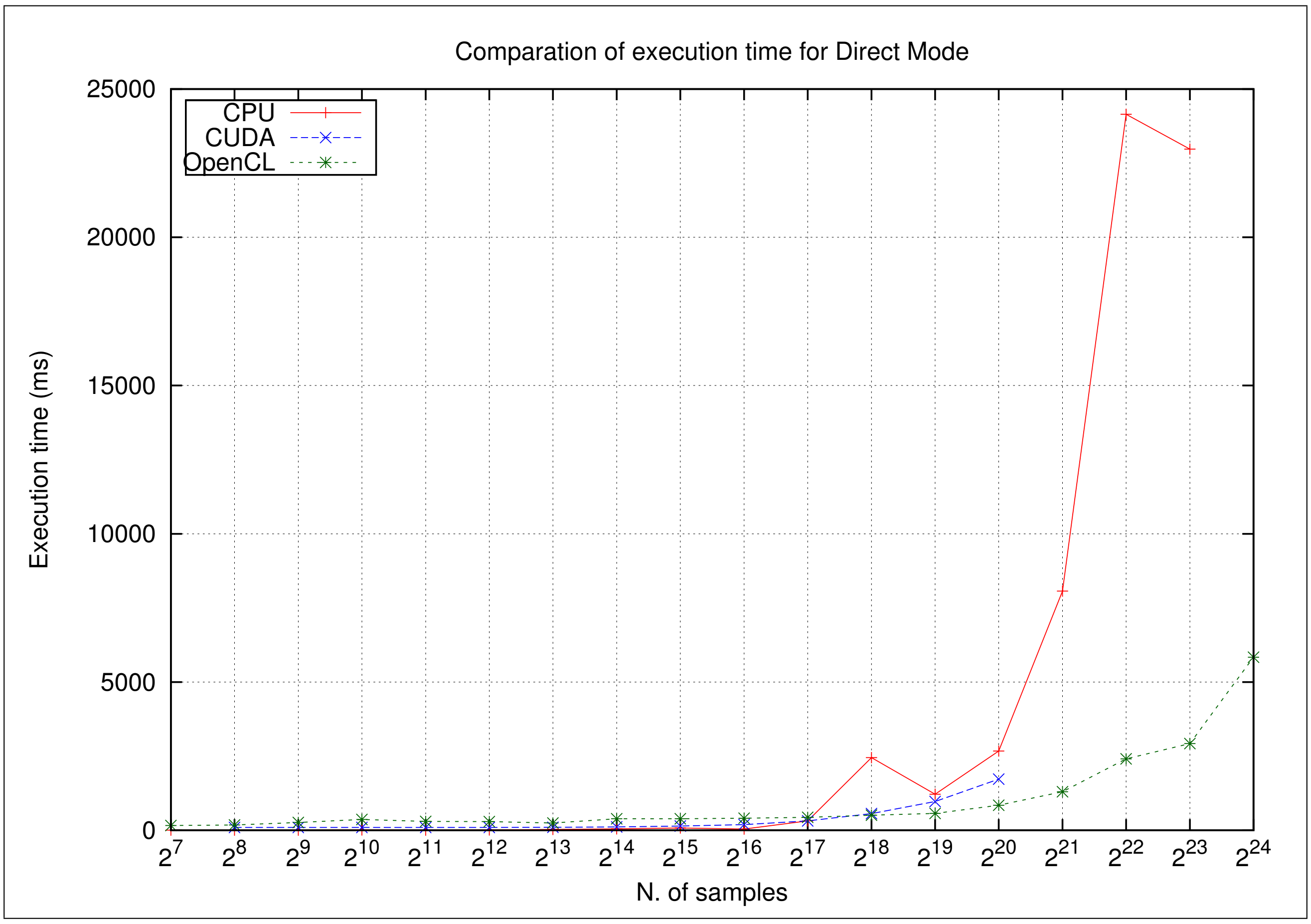

Figure 4: Execution time for Direct mode depending on input size.

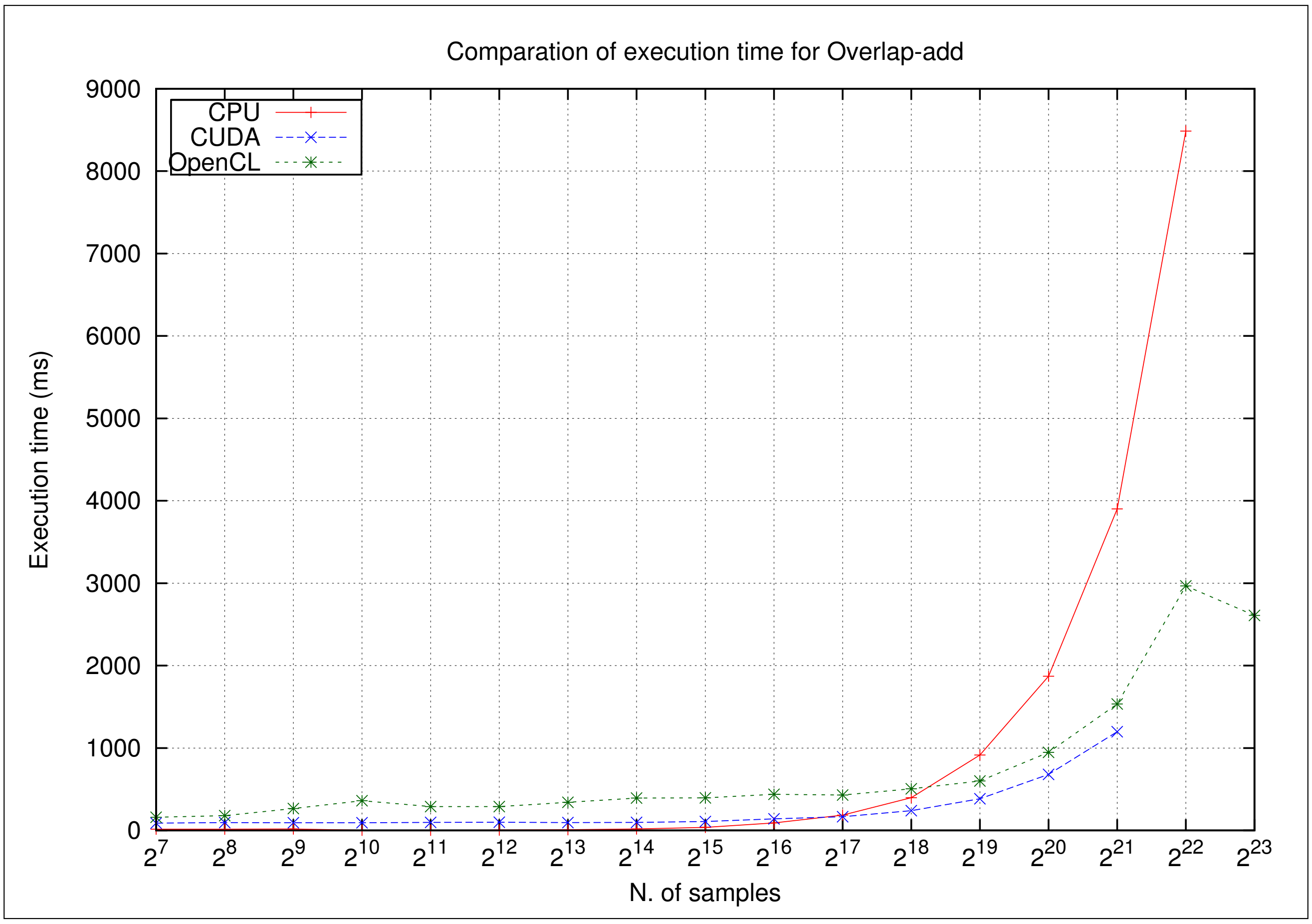

Figure 5: Execution time for Overlap-add depending on input size.