FAST PARALLEL AUDIO FINGERPRINTING IMPLEMENTATION IN RECONFIGURABLE HARDWARE AND GPUS

José Ignacio Martínez[1], *Jaime Vitola* [2], *Adriana Sanabria* [2], *César Pedraza* [3]

[1] Universidad Rey Juan Carlos, DATCCCIA, Móstoles Madrid, Spain.
[2] Universidad Santo Tomás, Facultad de Ingeniería Electrónica, Bogotá, Colombia.
[3] Universidad Santo Tomás, Facultad de Ingeniería de Telecomunicaciones, Bogotá, Colombia.

ABSTRACT

One of the main challenges that Music Information Retrieval (MIR) faces is performance. This paper presents an algorithm based on fingerprinting techniques implemented in a low-cost embedded reconfigurable platform. This fast algorithm is even faster when implemented in parallel for a GPU platform. The hit rate of the implementations is practically 100 % and the response time is two times faster than the response time of a top class PC, which means MIR times of up to 65 audio tracks in real time.

1. INTRODUCTION.

There has always been an interest in distinguishing an element from a group by using its own characteristics. One well-known and established technique is the human fingerprint: is unique for every human, can be read, stored and used easily in different environments. In connection with this idea, the legal rights management agencies have a great interest in checking if an audio track has been played in a radio station. That is why there has been quite a lot of work to develop different digital signature techniques for title and author name identification. One key point of these techniques is that the audience has to be able to listen to the audio tracks with the same audio quality (no audible alteration introduced by the digital signature) and the system be able to detect and identify the audio track in real time [2].

Generally speaking, the identification of a digital signature usually means acquiring and processing an audio signal, extracting a string of bits, matching that information with a database and signaling (yes or no) to the outside world. All of these mean that is a complex process with a high computational load and, in order to be useful, has to be robust, accurate, reliable, scalable, proportional and fast.

The paper is organized in seven sections. Section 2 describes how to retrieve audio in terms of content. Section 3 proposes a fast algorithm for real-time digital signature retrieval and comparison of audio tracks. Section 4 and section 5 respectively presents how to increase the algorithm performance implementing it in a CPU and an FPGA environment. Section 6 shows several tests that compare the performance of those implementations. This paper finishes with the conclusions and future work for this proposal.

2. BACKGROUND.

The analysis of an audio signal is made up of two main steps: extraction and identification of the digital signature. For the first step, extraction, we comment three main references: [3], [11] and [12]. [3] implements an audio fingerprinting technique called AudioDNA, which plays with AudioGenes to extract the information. This technique starts computing some characteristic values of 10ms frames of the audio tracks. After, carries on with a Hamming window convolution, an FFT, computing the MFCCs (Mel-Frequency Cepstral Coefficients) and a CDT. The AudioGenes are described as HMM (Hidden Markov Models).

Sert et al. [11] work with the whole audio track instead of audio track frames. This method takes advantage of the repetition of certain parts of the audio tracks (chorus) and uses a technique called ASF (Audio Spectrum Flatness) that can also be applied to frames in terms of a factor power of two called decim. [12] implements a different fingerprinting algorithm called PCA (Principle Component Analysis). For the second step, identification, we comment two main references: [1] and [3]. The digital fingerprint identification is made up of the database and the comparator that matches the signature with the database. That comparison is usually done with cross correlation algorithms. [1] splits the main database of 100000 audio tracks in 10 sub databases, starts 10 parallel independent processes and chooses from them the best answer. These techniques are developed to be as robust as possible because they have to face radio station distortions like audio compression for improving the signal level or signal pitching (playing the audio tracks faster) to save memory [3]. The main drawback of these algorithms is that they are not suitable for low-cost implementations. This paper presents a new simple algorithm based on the FFT and correlation for audio fingerprinting extraction. The FFT provide the algorithm more audio time shifting independen-

978-1-4244-8848-3/11/$26.00 ©2011 IEEE

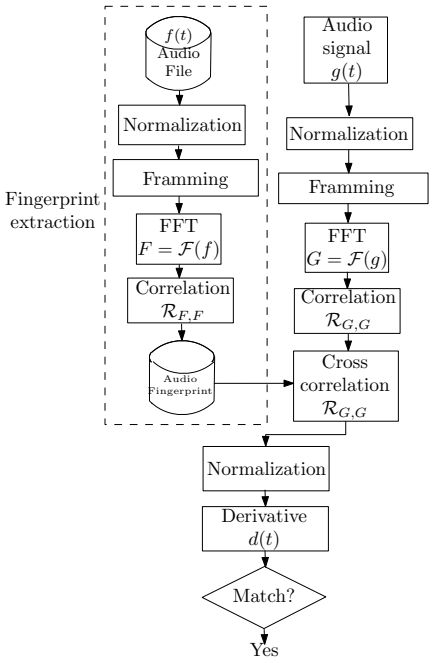

Fig. 1. Algorithm Front-end.

ce [10, 7, 3]. The correlation allows to get optimal values for comparison of two audio tracks when a pattern search is performed.

The algorithm was implemented in software on a medium-class PC, and then the implementation is improved in performance using GPUs as well as in low-cost reconfigurable platforms.

3. ALGORITHM.

The algorithm shown in Figure 1 is made up of the two parts already described: extraction of the digital signature of the audio track and identification of the audio track comparing the digital signature with the signature's database.

In the extraction step of the digital signature of the audio track the audio signal is digitalized and stored. Afterwards, the audio signal is split into smaller audio clips (windows) typically of 16 ms to 200 ms in order to be easily analyzed [2]. The frequency components of these frames are computed with a Fourier Transformation, so that the temporal dependency is avoided. The next step computes the auto-correlation of the entire frames' spectrum. This process allows us to determine how similar an audio track is in terms of a set of data (digital signature) equal to its number of frames.

In the identification step, the new audio signal is digitalized with a sample rate equal to the original audio track, and is also equally framed and computed its frequency components. Afterwards and for every frame there is a correlation computation of the frequencies with the original audio track spectrum, so that finally, the correlation set is compared with the original audio track digital signature. This procedure is repeated for every now frame capture in real time.

Table 1. Software algorithm profile.

Function	Description	% total
XCORR	Cross correlation	94 %
FFTW	Fourier FT of n points	1 %
MaxVector	Vector Maximum	< 1 %
AudioCapture	Input data	5 %

4. GPU IMPLEMENTATION.

4.1. Software Algorithm.

The algorithm was implemented in C and is able to identify audio clips of up to 30 seconds in real time using an audio card. The FFT computation was built with one of the fastest FFT software modules included in this library called FFTW [5]. The profiling of the algorithm proved that the performance bottleneck are the FFT and the cross correlation functions (table 1) .

With this profile in mind the performance of the algorithm would be much better if the XCORR and FFTW would be parallelized. Obviously, FFTW only needs to be improved when we think in embedded system implementations. This improvement in performance allows either finding the digital signature in a much bigger audio database or increasing the number of sources of retrieval in real time.

4.2. Algorithm parallelism.

Today, GPUs are one of the main approaches in shared memory parallel computing because they are able to concurrently run different code segments in the several cores included in one low-cost chip [8]. The programming model of the GPUs forces to use shared memory as communication mechanism for the processing elements (PEs) and threads as code elements to run in the PEs [4]. In order to develop general purpose applications for GPUs, NVIDIA delivers a software platform called CUDA with a whole set of tools for developing, debugging and running C Code in the GPUs. Now, we describe how the slower software functions have been improved in this programming model environment. The methodology proposed by Foster in [6] – partitioning, clustering, communication and mapping – was followed to parallelize the software algorithm. The partitioning splits the data domain into the different processors of the architecture. The clustering puts together those data with inter dependencies. The communication decides how to the data flows and

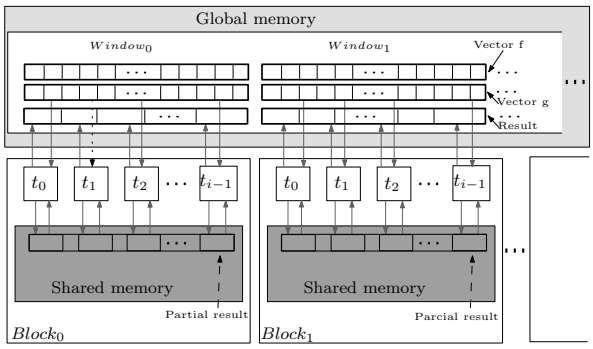

Fig. 2. XCORR implementation on GPU

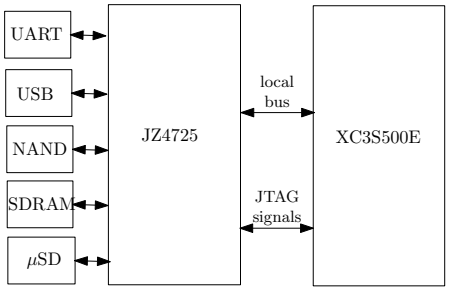

Fig. 3. SIE Platform.

works in the shared memory hierarchy. Finally, the mapping assigns every set of data to a core processing unit that would run its functions.

4.3. Correlation.

As was mentioned above, the most expensive function in terms of computational resources is the Correlation function. Every time an audio window w arrives is compulsory to compute the correlation for all the N frames of the audio track digital signature, so therefore the number of operations involves in two frames f and g, in an interval of i is around $N * j * (i * 2 + 1)$ multiplications, as can be shown in the equation 1 where f * denotes the complex conjugate of f.

$$R_{f,g}(w,i) = \sum_j f^*_{w,j} . g_{w,i+j} \quad (1)$$

It can also be noticed that there is data independency for the correlation computation in each i interval, so that means that the system can be split in a way that each scalar product of the f and g frames can be implemented in a t thread of the parallel platform. Therefore, the optimal would be launching $N * (i * 2 + 1)$ threads, each one with j multiplications.

Figure 2 shows how the function has been mapped into the graphic architecture. Each t thread executes a segment of code called $kernel$, this is in charge of the scalar multiplications of the two input frames, for a shift case. These way, the whole cross correlation of two audio frames can be solved launching a set of i threads or block that will be executed by a multiprocessor inside the GPU. Other i thread blocks would run to get to the whole N correlations of the two audio tracks, so that it can be said if they are similar or not.

4.4. Fourier Transformation.

Although the FFT only takes 1 % of the whole execution time of the algorithm, it makes sense to improve its performance in the GPU. NVIDIA's CUDA platform has a set of highly efficient software libraries for their own GPUs. In this paper, we worked with the CUFFT library, based on the FFTW algorithm [5], similar to our C implementation. The main reason why this library was chosen was because of its high performance and its appropriateness for the CUDA platform.

5. FPGA IMPLEMENTATION.

The algorithm was also implemented in a low-cost reconfigurable platform with a multimedia processor and a Spartan 3E FPGA. This platform, called SIE and developed with the hardware copyleft in mind with about USD $70, helps to solve computational demanding problems for embedded systems [9].

Figure 3 shows how the 32 bits processor JZ4725 (optimized for mobile systems) communicates with the outside world and the XC3S500E FPGA. The whole system has SDRAM, NAND and micro-SD memory controllers and communications with the outside world through an USB2 bus. The system provides all the development tools for the users to easily use the Linux kernel 2.6 (tool chain, file system, debugging, etc.) As mention above, the slowest function is the correlation were improved to take advantage of the Reconfigurable platform resources, leaving the rest of the tasks to be run in software with the same source code. In order of maximize the processing inside the FPGA, besides the correlation core, a FFT core were inserted, accelerating in fixed point mode this step.

In a similar way as in the previous section, were accelerated fast Fourier transform and the correlation algorithm in high-performance alternative platform (FPGA). The remaining tasks were implemented by software, using the same source code.

Figure 4 shows how the improved parts of the algorithm were implemented in the FPGA hardware. There is one core for the FFT, one core for the frame magnitude computation and one core fo the Correlation. They are all connected to the JZ4725 processor through a specifically designed interface. In the algorithm run, the frame window data are transferred to the FFT and Magnitude core. The results obtained are

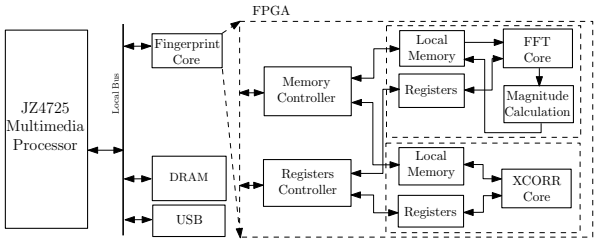

Fig. 4. Coprocessor block diagram.

sent to the processor and stored for the next comparison to the audio track spectrum. In the comparison process the two data frames are sent to the XCORR core so that it computes the correlation values.

6. EXPERIMENTAL RESULTS.

6.1. Experiment.

The experimental environment was made up of different real time tests on audio tracks of 5, 10, 20 y 30 seconds with a 3600 elapse time to determine the quality of the algorithm. Also, several tests were run for different frame length – from 16ms to 256 ms.

6.2. How the algorithm works.

It was also tested how effective the retrieval of the audio tracks worked in terms of hit rate for every frame during the whole 3600 seconds with 64 ms time frame, letting us proved that the algorithm was able to find a coincidence in time. Another test added white Gaussian noise (20 % amplitude) to the audio tracks to check if the algorithm was noise immune.

Figure 5 shows the results commented above. It is clearly observed that there is a peak in the probability for every test where the audio track was in the air. Getting a probability bigger that 0.7 % means that there is a coincidence or the audio track was in the air.

6.3. CPU vs GPU performance.

It was also time for comparing the algorithm in CPU (Intel Core 2 duo processor) and GPU (NVIDIA GeForce 240 with 96 CUDA cores) implementations. The response time of the algorithm was measured from the time a frame was acquired up to when the comparison of the digital signature of a 10 seconds audio track was finished.

It is important to mention that the response time of the system includes the program OS calls and the communication time for the processing elements. With this in mind, the GPU/CPU speedup was computed for different frame

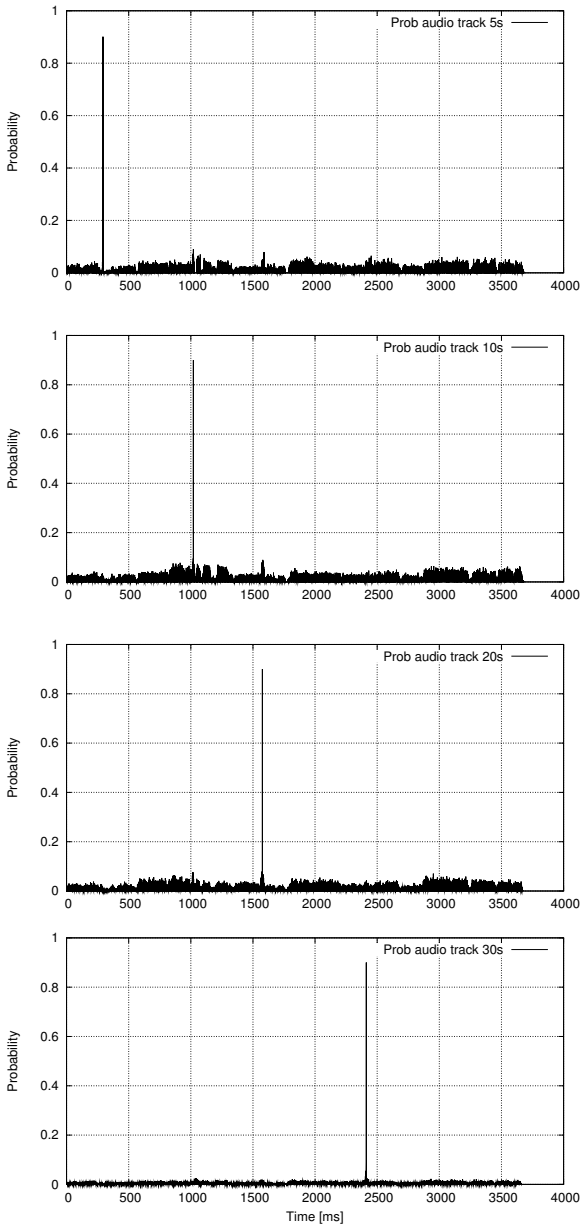

Fig. 5. Probability to find audio tracks with different time frames.

lengths, as shown in Figure 6. There is a better performance when the GPU is in use for 32 ms or bigger frame lengths.

There were other tests to measure the maximum number of files to find in every case, varying the time frame. Figure 7 shows that the best scenario is when the time frame is the biggest – 256 ms – because the total frame number N decreases, meaning less computational load. In this situation the GPU almost doubles the number of audio tracks found by the CPU, reassuring the efficiency of the parallel archi-

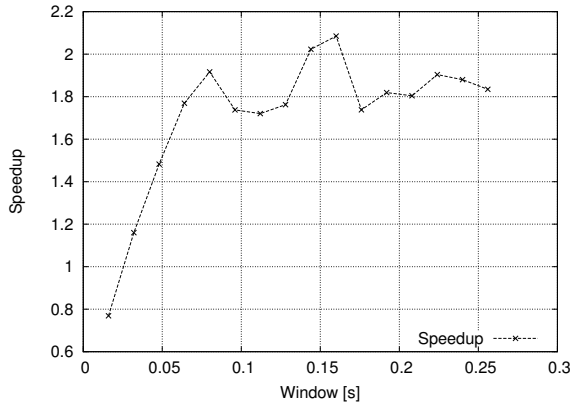

Fig. 6. Speedup de GPU vs CPU algorithm speedup for frame window.

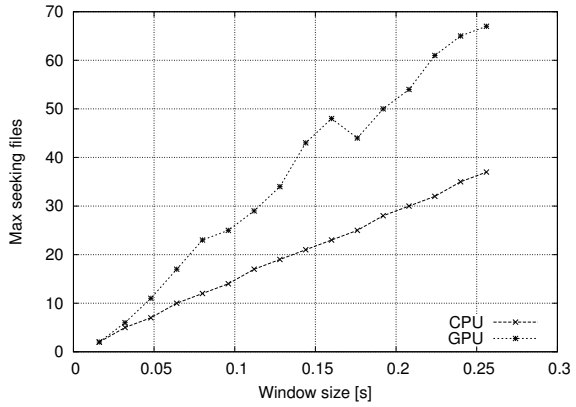

Fig. 7. Maximum number of files per window frame.

tecture for this implementation. A better performance was not carried out due the communications between device and host (GPU and CPU), besides the algorithm did not show a high parallelism when executed in real time.

6.4. SIE performance.

The SIE implementation was tested for a fixed frame window size of 64 ms. This value was chosen because it also determines an appropriate FFT core size. Due to the XC3S500E FPGA resources are quite small; there is only 1024 number of points. This allowed saving some resources for the rest of the logic needed in the processor.

Table 2 shows how fast the algorithm implemented in SIE is for a 64 ms frame window. It can be seen that, for the audio clips of 5 and 10 seconds, the response time is smaller than the frame window, therefore is appropriate for real time. But, this is not the case for the audio clips are 20 and 30 seconds long. The profile of the algorithm run in SIE showed that the transfer data rate between processor and FPGA is

Table 2. Response time for algorithm in SIE.

Size of window	Audio track			
	5s	10s	20s	30s
64ms	26.2ms	53.3ms	110.2ms	250ms

overloaded in up to a 150 % when compared with the FFT and $XCORR$ cores. This is due to that the system local bus is only 8 bits long, therefore developing a critical bottleneck in the system. However, SIE implementation represents a low cost solution for this applications and keeping in mind the power of processing, a comparison between GPU and SIE is not completely fair.

7. CONCLUSIONS AND FUTURE WORK.

This paper presented the design and test of a music information retrieval algorithm. The retrieval process was improved in 2x performance with the use of a medium-cost GPU platform. Using a GPU platform means a real improvement because the increase in performance is bigger than the increase in price when using the GPU platform to find more audio tracks. The test of the algorithm found out that when the audio tracks are short ($\geq$ 10 seconds) is better to use frame windows of 32 ms to 64 ms. But on the other hand, with bigger frame windows there is a better performance to find more audio tracks in real time. Therefore, 128 ms could be a reasonable frame window because allows to find audio tracks with enough resolution as well as at a reasonable fast pace. It was also noticeable that the algorithm performance on the GPU does not grow linearly, mainly because the algorithm in the GPU depends on the number of threads by block, which defines the efficiency of how the threads are run in parallel.

SIE implementation showed a low cost way to make audio fingerprinting with short audio tracks. As a future work, a more robust algorithm can be designed with more noise immunity and with proof of pitch changes. Also SIE can be improved implementing fast transfer techniques between the processor and the FPGA, allowing to analyse more data in less time or more tracks searching in real time.

Acknowledgment

This work was supported by the MCIN – TEC2009-10639-C04-03 (subprogram TEC) "Visión Ultra-rápida por eventos y sin fotogramas, Aplicación a Automoción y robótica cognitiva Antropomorfa"project.

8. REFERENCES

[1] Carlo Bellettini and Gianluca Mazzini. A Framework for Robust Audio Fingerprinting. *Journal of Communications*, 5(5):409–424, 2010.

[2] P Cano, E Batlle, T Kalker, and J Haibma. A Review Algorithms Audio Fingerprinting. In *IEEE International workshop on MMSP*, volume pp, pages 169–173, 2002.

[3] P Cano, E Batlle, H Mayer, and H Neuschmied. Robust sound modeling for song detection in broadcast audio. In *Proceedings of the 112th AES Convention*, volume pp, pages 1–7, 2002.

[4] NVIDIA Corporation. NVIDIA CUDA Programming guide., 2009.

[5] M Frigo and SG Johnson. FFTW: An adaptive software architecture for the FFT. *Conference on Acoustics Speech and Signal*, 1998.

[6] Foster Ian. *Designing and building parallel programs*. Addison Wesley, 1995.

[7] JG Lourens. Detection and logging advertisements using its sound. *Broadcasting, IEEE Transactions on*, 2002.

[8] D Luebke, M Harris, and N Govindaraju. GPGPU: general-purpose computation on graphics hardware. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 208. ACM, 2006.

[9] Qi-Hardware. SIE Platform, 2009.

[10] G Richly and L Varga. Short-term sound stream characterization for reliable, real-time occurrence monitoring of given sound-prints. *MELECON 2000 - Mediterranean Electrotechnical Conference*, 2005.

[11] M Sert, B Baykal, and A Yazıcı. A Robust and Time-Efficient Fingerprinting Model for Musical Audio. *Audio*, 2006.

[12] L Shen, Y Guan, Y Wu, and Y Zhao. Fast audio fingerprint search strategy for song identification. *Networking and Digital Society, International Conference*, pages 259–262, 2009.