Using GPU to Speed Up the Process of Audio Identification

Natalia Miranda, Fabiana Piccoli
Universidad Nacional de San Luis
San Luis - Argentina

Edgar Chávez[(1)(2)]**, Antonio Camarena-Ibarrola**[(1)]
(1) U. Michoacana México
(2) CICESE
México

e-mail: {ncmiran, mpiccoli,echavez}@unsl.edu.ar {elchavez,camarena}@umich.mx

***Abstract*—Retrieving audio signals by content in large repositories require a stable and persistent representation of the audio. Such a representation is called an Audio Fingerprint (AFP). A robust AFP should be an invariant of the audio signal and it should represent the signal despite severe degradations it may suffer. The robustness of the AFP is inversely proportional to the compute effort put in obtaining the AFP from the audio, and hence impose limits on the processing throughput.**

On the other hand the Graphics Processing Unit (GPU) provides high performance computing through the threading model. Its main characteristics are high computational power, constant development and low cost.

In this work, we propose the use of massive parallel algorithms implemented in the GPU to speedup the computing of a robust AFP. This parallel system will be part of a comprehensive system, which will allow to determinate the audio fingerprints of several audio signals generated simultaneously. Experimental results on the corresponding speedup are presented.

I. Introduction

Audio identification consists in the ability to pair audio signals of the same perceptual nature, this demands an object representation stable and persistent to different natural degradation of the objects. Such robust representation is called an Audio-Fingerprint (AFP). Two audio signals are perceptually similar if they have the same AFP. AFP's are mature technologies used as software commodities by a very large number of applications of economic importance. Among other tasks, AFPs are used in broadcast monitoring [1], signal identification [2], [3] and automatic score following [4].

In designing AFPs for such use, there is a tension between two competing goals. On the one hand a robust *feature* generally implies a dense representation of the audio, and correspondingly a robust *fingerprint* generally implies a denser representations of a song. On the other hand, a dense AFP imply more compute cycles to obtain the representation.

In some applications an audio collection, represented by their AFP, is queried against an unknown audio sample. To avoid comparing with all the audio sample in the collection it is possible to build a metric index to satisfy proximity queries. There are some applications where the situation is reversed and the audio collection is given on-line and it need to be compared against a single audio sample. An example application with this behavior is the monitoring of radio broadcasting. The goal is to *listen* to a large set of audio streams (the broadcasting stations in a city) and wait for the appearance of a particular audio stream, such as a commercial advertising, in any one of the streams. In this case it is not even possible to obtain all the AFP of all the audio streams in real time using a single CPU. The Graphics Processing Unit (GPU) represents a good alternative to speedup the AFP process. GPU's are portable, affordable massive parallel devices, originally conceived to speed up online rendering. In present days they can provide up to 50 times the processing power, compared to the host computer [5].

Since its inception, the GPU was used as a dedicated device for speeding up graphics processing applications, 3D video gaming, rendering, etc. [6], [7]. The progress of the GPU was faster than for CPU, probably due to a smaller instruction set and single precision arithmetic [8], [7]. The GPU is in many senses a portable super computer. Certain type of tasks can be solved using a massive parallel model, with a multi-core processor,

shared memory and hyper threading support.

The GPU programming evolved from hacking graphics specific settings and programs to a more structured C-like programming environment. The most successful model is provided by the *Nvidia* graphics card, with a driver hiding the low-level details and differences between different graphics card models. This model is dubbed Compute Unified Device Architecture (CUDA) with a GPU-CPU interface, thread synchronization, data types, etc. [9], [10], [11].

The goal of this work is to improve the throughput for online processing of a multi-stream source. The paper is organized as follows: in sections II, III and IV, we review the characteristics of an AFP, the entropy of a signal and how to compare sounds. The section V compares the CPU and GPU programming. In section VI , we analyse the characteristic of GPU-CPU system parallel. Finally, we show the experiments, obtained results and conclusions.

II. Characteristics of an AFP

A good AFP must include the following characteristics:

- Robust to signal degradations such as noise mixing, equalization, cropping and time shifting.
- Compact and determined with as little computational effort as possible.
- Scalable, that is, it should be able to operate with very large databases, conditioned by a good indexing technique.

An audio-fingerprinting system consists of *Feature extraction* and *Audio-fingerprint modeling* modules. In the *Feature extraction* module, the perceptual features of the audio signal are extracted and they are taken by the *Audio-fingerprint modeling* module with the purpose of building an AFP, that is adequate for *Information Retrieval* purpose.

Therefore, the first thing an audio-fingerprinting system has to do is to extract features from the signal. Most AFP systems however, extract signal features in the frequency domain using a variety of linear transforms such as the Discrete Cosine Transform, the Discrete Fourier Transform, the Modulation Frequency Transform [12] and some Discrete Wavelet Transforms like Haar's and Walsh-Hadamard's [13].

Looking for more relevant features of audio signals a variety of perceptual features have been assessed such as the Mel-frequency Cepstral coefficients (MFCC) [14]; Loudness [15]; the Joint Acoustic and Modulation Frequency (JAMF) [12], [16]; the Spectral Flatness Measure (SFM) [17]; the Spectral Crest Factor (SCF) [17]; tonality [18] and chroma values [19]; among others[20]. [21] proposes to compute the audio entropy for every signal second.

Finally the features obtained from *Feature extraction* module are modeled in a way that best serves the purpose of the application for which the audio fingerprints have been designed. Examples of existing audio models are: *Trajectories* [2]; *Statistics* [22]; *Codebooks* [23]; *Strings* [24]; *Hidden Markov Models (HMM)* [25], [26]; *Gaussian Mixture Models (GMM)* [27], [28].

The technique described here differs to these approaches in that it does not reply on specific domain knowledge, and is therefore more widely applicable.

III. A relevant perceptual feature: Signal Entropy

The entropy of a signal is a measure of the amount of information the signal carries. If X is a random variable representing the signal, and we want a unique value to identify it, then Shannon's entropy is a good candidate. Small perturbations on the sample values of X produce smaller perturbations on the measured entropy. If the sample values of X are denoted by $\{x_i\}$ then entropy is defined as

$$H(X) = -\sum_i p(x_i) ln(p(x_i)),$$

where $p(x_i)$ is the probability for the signal X to take value x_i.

Since the audio signal is additive, we will fix our attention to the modulation (the change) of the entropy over time. If we compute the entropy values in a sliding window of the signal, the sequence of values encode the changes of the audio entropy over time. If the volume (the energy) of the audio is increased or decreased, the corresponding entropy curve is also shifted preserving the relative changes.

Adjusting shifts to match signals is an easy task, the vertical shift disappears if we take the derivative of the signal, or even more if only the sign of the derivative is retained. Unfortunately, other interesting distortions, like re-recording, are not profile invariant. A similar effect is observed when the signal is equalized.

The Time-domain Entropy Signature (TES) is a sequence of binary values, one per each frame, indicating the sign of the derivative of the entropy profile. This AFP was compared with Haitsma et al AFP [2] in [21] obtaining good results for low pass filtering, lossy compression and volume changes. For re-recording or equalization the results were not as good. In this work, the entropy calculation is undertaken in the frequency domain, with logarithmic bands used to offset the effect of equalization and other distortions.

A. The Multiband Spectral Entropy Signature

The distortion observed in the time domain for re-recording or equalization can be reverted if we divide the signal in subbands using for example the logarithmic Bark scale of 24 critical bands. After the band division, if we compute the entropy profile of each suband separately the corresponding bands will have vertical shifts only, even for distortions like equalization or re-recording. Thru this process, we obtain the Multiband Spectral Entropy Signature ($MBSES$)

This is illustrated in figure 1 where only some of the 24 bands are shown to avoid overcrowding the figure.

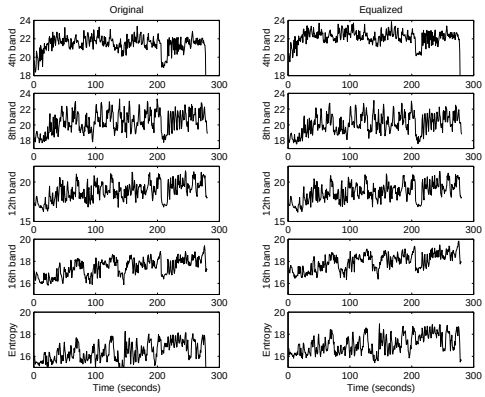


Fig. 1. Entropy profiles for individual bands in the Bark scale.

B. Binary Encoding the Signature

For each frame we keep only an indication of whether the spectral entropy is increasing or not for each band. Equation (1) states how the bit corresponding to band b and frame n of the AFP is determined using the entropy values of frames n and $n-1$. The same property of compactness noted in TES is retained in the spectral version. Only 3 bytes (i.e., 24 bits) are needed for each frame of audio signal.

$$F(n,b) = \begin{cases} 1 & if \ \ [h_b(n) - h_b(n-1)] > 0 \\ 0 & Otherwise \end{cases} \quad (1)$$

IV. Comparing Audio Signal

So far we have a binary array for each song or audio in the collection. The Hamming distance between two same sized excerpts accounts for the perceptual similarity between them. The smaller the Hamming distance the higher the perceptual similarity, as it was discussed above. If we want to know if an excerpt occur in some song in the collection we need to scan, in principle, all the collection to find the alignment with the smaller Hamming distance.

The sequential scan with the MBSES does not scale well. As a formative experiment, we used off-the-shelf desktop hardware to scan a database of pre-computed MBSES. The database comprised of approximately ten thousand songs from a wide range of genres (from country to classic). With these signatures pre-loaded into memory, we are able to scan roughly 17 hours of audio per second when using audio excerpt of 5 seconds, and 10 hours of audio per second with a 10 second excerpt. Nevertheless, to scale to collections with millions of songs—as is the case with iTunes for instance, with an ever growing set of users—a more efficient indexing method is needed. This motivates the use of a general index to speed up searches.

A. Probabilistic Pairing Pseudo Metric

Lets assume we have a *base* distance $d(x,y)$[1] to compare similar sized audio samples x and y of sizes m and n respectively with $m \sim n$. It can be the case the base distance require $m = n$, as for example the Hamming distance. If the case of the *edit* distance, sizes m and n need to be just comparable.

The *probabilistic pairing pseudo metric* (PPPM) $D(x,y)$ is a generalization of the base distance $d(x,y)$ defined as follows: If $n < m$:

$$D(x,y) = \min_{x(i,i+n)} d(x(i,i+n), y(1,n)) \ \forall \ 1 < i < m \quad (2)$$

Otherwise:

$$D(x,y) = D(y,x)$$

In other words we use a sliding window of the smaller object over the larger one and use the minimum as the value of the distance.

The function defined in Equation 2 does not strictly satisfy the triangle inequality, although it does satisfy it with high probability since the case where it is not satisfied is rarely found.

V. GPU and CPU Processing

A single PC with one or multiple cores cannot be compared in performance with CUDA, because hundreds of thousands of threads can be attended simultaneously. Our proposal is to used CUDA to boost the throughput in audio processing. One possible application is to monitor

[1] $d(x,y)$ is a distance between bit vectors, to fix ideas we can think in the Hamming distance between bit vectors

simultaneously, with a single PC, the hundreds of radio broadcastings in a large city, or to listen for hundreds of simultaneous queries for query by content in audio databases. Audio databases and audio monitoring are specially suited for the massive parallel model provided by CUDA.

A GPU can be considered as a multicore processor allowing a large number of fine grained threads [29]. The GPU is different from other parallel architectures in the flexible local resource assignment, either memory or register, for the threads. Each stream multiprocessor can execute a variable number of threads, it is a programming decision the resource assignment. Performance can be boosted by optimizing the assignment of resources.

The whole model consist in a traditional CPU based station and one or more coprocessors, the massive parallel compute devices. Each coprocessor apply the same model of Simple Instruction Multiple Data (SIMD). All computing units execute the same code (not necessarily synchronized) over the different set of data. The threads share the same global memory.

CUDA is a computing environment allowing software developers to create isolated programming components. Each component solve a problem in a dedicated GPU device applying massive parallel data processing. CUDA provides a programming model facilitating application development on the GPU.

A CUDA program is a C/C++ extended with a set of instructions. This instructions specify parallel code and data structures to be executed in the device. Those computing units are named *kernels*. A kernel describe the work of a single thread and can be executed by hundreds of them. There are some restrictions on the kernels, they cannot execute recursive calls, static variables cannot be declared and the number of arguments cannot be variable.

A complete CUDA program have different phases to be executed either on the CPU or the GPU. When the phase have low or null parallelism it is assigned to the CPU. If the phase is massively parallel it is implemented as a kernel and executed over the GPU.

At the beginning and end of a program the host make a transfer from/to the global data or the GPU data space. Threads are organized in a three level hierarchy: *Grid* the top level consisting in a block of threads, *Block* mid level consisting in a group of threads stablished by the software developer, and the lower level *Threads* which can synchronize the task and share data inside the same block. The number of grids, blocks and threads affect the performance of the tasks, each application have an optimal selection for these parameters. As a rule of thumb these parameters are determined by experimentation.

VI. Parallel Multi-MBSES

Figure 2 illustrate the parallel architecture for the digital signature dubbed $MBSES_p$. As said before the problem is particularly well suited for massive parallel processing.

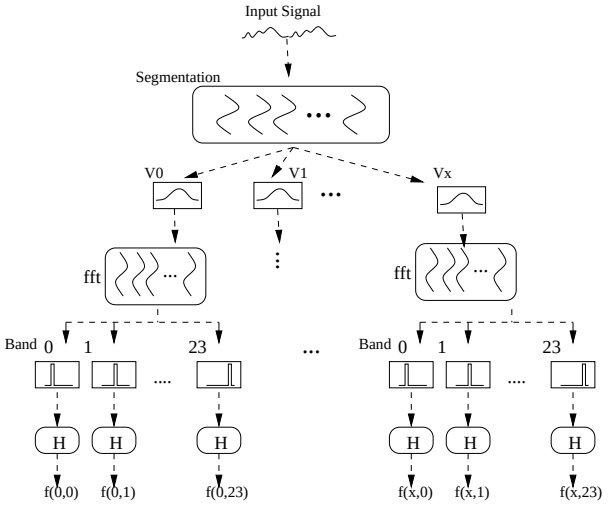


Fig. 2. Architecture of $MBSES_p$

Multi signal processing is sketched in *Multi-*$MBSES_p$, where additionally to parallel processing of a single signal, multiple signals can be processed at once, each one of them performing the same task with different data. Figure 3 illustrates.

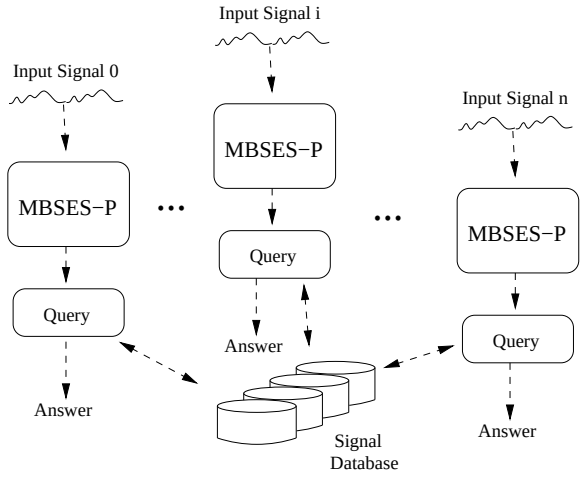


Fig. 3. Multi signal $MBSES_p$ system

In both schema $MBSES_p$ and *Multi-*$MBSES_p$ the massive parallel architecture can be applied. Several parameters need to be adjusted. In this work we discuss three crucial parts of the processing, computing: the Hanning window, the fast Fourier transform and computing the signature entropy based on histograms.

A. The Hanning Window

Computing the Hanning window is an inner product, and hence is suitable for massive parallel processing. All the threads will perform the same operation and the final algorithm is a pure data parallel procedure with no cross-talk between threads. The usual consideration should be applied, if the number of threads is smaller than the data size, then each thread will take care of a subset of the vector.

B. Fast Fourier Transform

In the CUDA repository there is a library for parallel computing the FFT, this routine is called the CUFFT. We implemented the FFT from scratch based on the original algorithm of Cooley and Tukey [30]. The inverse and direct FFT can be computed changing a single parameter. The sample is divided in two subsets of size half the original size, using the Danielson Lanczos theorem. This process is repeated recursively or iteratively until the set is of cardinality two.

We fixed 512 threads to be executed in parallel, computing the bit-reverse vector in a first stage and in a second stage we properly computed the FFT. For the bit-reverse, each even index element in the first part of the vector is swapped with a corresponding even index element in the second part of the vector. Each swap is computed by a different thread. For a vector of size N we need $\frac{N}{4}$ threads. If N is larger than the number of available threads T, then each thread will swap $\frac{\frac{N}{4}}{T}$ elements.

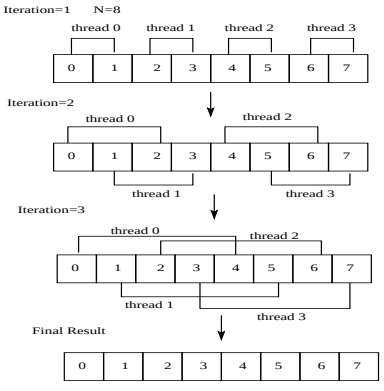


Fig. 4. GPU based FFT computation

The second phase is where the FFT computation takes place properly. Since it is not possible to apply recursive calls, the solution is iterative. Each thread, in each iteration, makes the proper computation with the corresponding pair. If the number of threads is smaller than the vector size each thread will take care of a fraction of the data. In figure 8(b) the procedure is sketched for each iteration.

C. Entropy Signature

Computing the entropy of a signal requires some estimation of the Probability Density Function (PDF). Such estimation may be accomplished using parametric methods, non parametric methods and histograms. Parametric methods [31] are advisable when the distribution is known a priori and the amount of data involved is not large. In non parametric methods, no assumptions are made about what distribution the PDF belongs to, the PDF is shaped by the data which is in turn smoothed by some kernel. Non-parametric methods are computationally expensive and so not frequently used for realtime pattern recognition applications. The histogram is the method of choice, it is a simple and fast approach to estimate entropy. In this case, the confidence of the histogram method is ensured by the fact that thousands of audio samples will be used to compute the histogram.The probability p_i for value v_i to be a sample read from the audio stream is computed using Laplace's formula $p_i = \frac{f_i}{N}$, where f_i is the number of times that value v_i occurs in the sequence $x = x_1, x_2, .., x_N$, N is the frame size.

The Bark scale defines 25 critical bands, the first 24 corresponding to the bands of hearing. The last band, the 25th band, is discarded since only the youngest and healthiest ears are able to perceive it. For any given band b, the elements of the time-domain frame of the signal (after computing the FFT) corresponding to b are used to build two histograms, one for the real parts and another one for the imaginary parts of these elements. The histograms are used to estimate the probability distribution functions. The entropy for real and imaginary parts are computed separately and operated together to obtain the i-component of TES. Figure 5 illustrates.

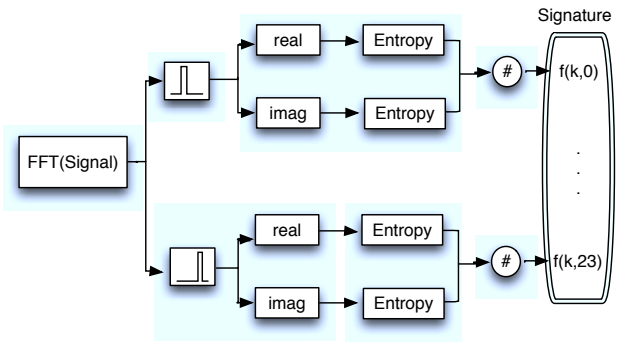


Fig. 5. Entropy computation

Shared memory per block	16KB
Registers per block	8KB
Max. number of threads per block	512
Max. sizes of each dimension of a block	512 x 512 x 64
Max. sizes of each dimension of a grid	65535 x 65535 x 1

Fig. 6. Common Characteristics among Three GPU

GeForce	8500 GT	9500 GT	9500M GS
Global Memory	512MB	256MB	512MB
Multiprocessors	2	4	4
Central Clock	450 MHz	500 MHz	475 MHz

Fig. 7. Differences among Three GPU

In the next paragraph, we describe the necessary steps to compute the entropy signature with histograms on GPU.

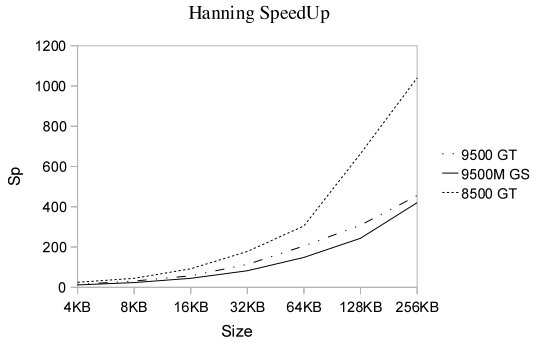
1) Histogram: Computing the histograms is tricky and involve many tasks, the first is the discretization of the current signal frame, the continuous values have to be converted to discrete values. This task implies obtaining the max and min (M_m) over all frame component (real and imaginary parts). M_m employs the max number of threads, 512, to obtain the max and min value of a subset of the frame. Each thread works over the same number of data points. After that, half of the threads are dedicated to calculate the max and the other half compute the min. Once acquired min and max, every signal elements are transformed to a value between them. This task is similar to computing the Hanning window, the threads will perform the same operation over a subset of frame data, without communication among them. We use 8 bits to represent each data element of the frame, hence each discrete data element will take one value between 0 and 255. The second task is to properly compute the histogram. In this process, the distribution of the computation between multiple execution threads is made by subdividing the frame data between them, processing of the subset by each dedicated execution thread, and storing the result into a certain number of sub-histograms. Finally all the sub-histograms need to be merged into a single histogram. Between each inner step it is necessary to synchronize the threads.

VII. Experimental Results

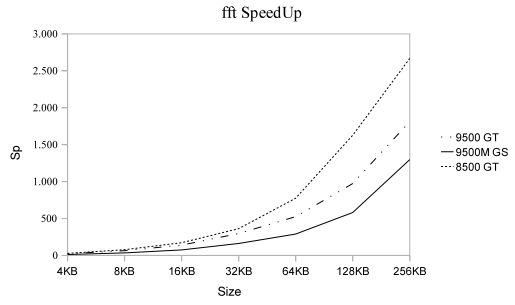
For a comparative analysis we selected a sequential CPU implementation of the algorithms, the fastest machine available for experiments has the following characteristics: Intel core 2 Duo E8400, with 3GB of RAM. We used three different GPU models for comparison. The 8500 GT, 9500 GT and 9500 GS. They had the following common characteristics (see table in Figure 6).

With the following differences (see table in Figure 7).

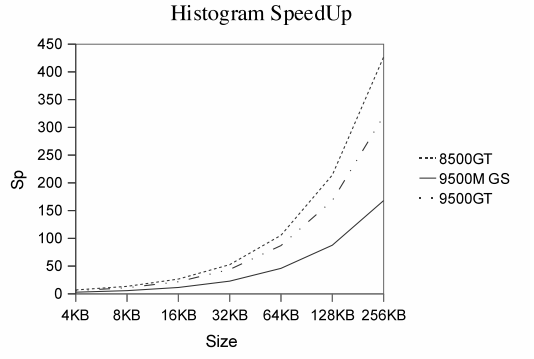
The results shown for the speedup are the average over several runs. Figures 8(a), 8(b) and 8(c) show the speedup for the Hanning window computation, the FFT and Histogram for differents frame sizes. In all the cases we used the maximum number of available threads. We can observe a significant improve respect to sequential process, independent of GPUs characteristics.



(a) Hanning



(b) fft



(c) Histogram

Fig. 8. Speedup of Hanning, FFT and Histograms computation

We compared our implementation with the state GPU based library CUFFT, available in the CUDA showroom.

We used the CUFFT as a black box. Our implementation surpass the efficiency of the state of the art. Figure 9 shows the comparison to four frame size.

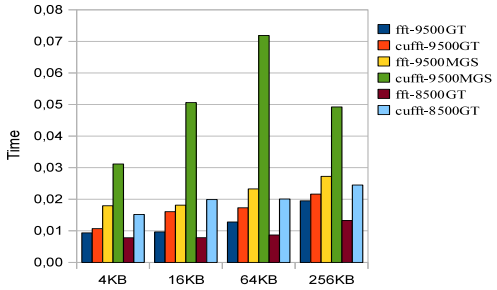


Fig. 9. FFT vs CUFFT

In all cases our implementation was faster than CUFFT, which is the state of the art.

Finally, we analyzed the overall speedup. At the moment of writing this note, the $MBSES_p$ is mixed implementation, we used a CPU/GPU model, the first three tasks were resolved over GPU (final $MBSES$) , the last task was computed in the CPU. When the third task ends, the data had to be moved from the GPU to the main CPU memory. Although, data transfers impose a severe restriction on the performance, the results are good. Figure 10 shows the overall speedup when frame size is 16KB, this size is equivalent to a frame duration of 370ms. In the conclusions and future work section we discuss the expected speedup in a pure GPU implementation when no data transfers are made for intermediate results.

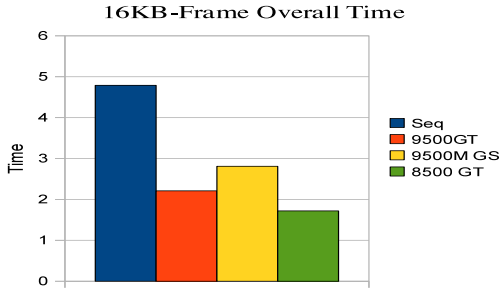


Fig. 10. Overall Time of $MBSES_p$

Figure 11 displays the sequential and parallel GPU times of nine different audio signals. The selected signals have diverse durations (in seconds, they are are indicated in the label: $Sig - \#second$). The frame size is 16KB. We can observe a significant improvement with respect to the overall time of computing the audio signature implemented on three different GPU boards.

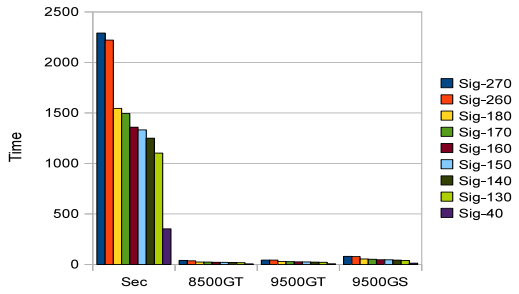


Fig. 11. Sequential vs parallel GPU times of nine signals

The signal is processed in frames of 370 ms, this frame size ensures an adequate time support for entropy computation according to our experiments. The frame sizes normally used in audio-fingerprinting ranges from 10 ms to 500 ms according to [20]. The frame size used in [2], [3] is precisely 370 ms.

Conclusions and Future Work

In this work, we sketched the basic characteristics of the $MBSES_p$ system. This system describe the Multi-band Spectral Entropy Signature ($MBSES$) following the CUDA programming model. We presented results supporting about a 2.5 speedup to compute a single frame, this accounts for the overall time needed to transfer a single frame to the GPU RAM, computing the FFT and the entropy and transferring the result to the CPU. If the whole audio signal can be accommodate in the GPU RAM then the time to compute a single frame will be in the same order of magnitude of computing all the frames signatures for the audio without transferring partial results to the GPU. In a conservative estimation if an audio signal of 1462 frames of 16Kb length (a standard song) can be computed in 4165,63 milliseconds in the sequential model, the corresponding GPU implementation will use 0,055 accounting to a speedup of 4 to 5 orders of magnitude (considering the memory and bus and thread arbitration).

Of independent interest is our FFT implementation that is faster than the CUDA implementation of the FFT (CUFFT), a more thorough comparison to detect the weak points of the CUFFT is needed to recommend our implementation over the CUFFT. The FFT is widely used in many areas of application and we believe our implementation can be of interest to practitioners. We are also implementing a massive parallel version of the metric indexes suitable for audio indexing. We are aiming a pure GPU implementation of the system.

REFERENCES

[1] S. Shin, O. Kim, J. Kim, and J. Choil, "A robust audio watermarking algorithm using pitch scaling," in *14th International Conference on Digital Signal Processing*, vol. 2, pp. 701 – 704, 2002.

[2] J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system," in *International Symposium on Music Information Retrieval ISMIR*, 2002.

[3] A. Wang, "An industrial strength audio search algorithm," in *International Conference on Music Information Retrieval (ISMIR)*, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003, 2003.

[4] A. Camarena-Ibarrola and E. Chavez, "Real time tracking of musical performances," in *MICAI*, vol. To appear, 2010.

[5] D. Lloyd, C. Boyd, and N. Govindaraju, "Fast computation of general fourier transforms on gpus," in *IEEE International Conference on Multimedia and Expo*, p. 5:8, April 2008.

[6] I. Buck, "Gpu computing with nvidia cuda," *SIGGRAPH '07: ACM SIGGRAPH 2007 courses ACM,*, 2007. New York, NY, USA.

[7] D. Luebke and G. Humphreys, "How gpus work computer," *EEE Computer*, vol. 40, p. 96:100, Feb 2007.

[8] M. Lieberman, J. Sankaranarayanan, and H. Samet, "A fast similarity join algorithm using graphics processing units," in *ICDE 2008. IEEE 24th International Conference on Data Engineering 2008*, p. 1111:1120, April 2008.

[9] M. Joselli, M. Zamith, E. Clua, A. Montenegro, A. Conci, R. Leal-Toledo, L. Valente, B. Feijo, M. Dórnellas, and C. Pozzer, "Automatic dynamic task distribution between cpu and gpu for real-time systems," in *11th IEEE International Conference on Computational Science and Engineering, 2008 (CSE '08)*, p. 48:55, July 2008.

[10] W. Chen and H. Hang, "H.264/avc motion estimation implementation on compute unified device architecture (cuda)," in *IEEE International Conference on Multimedia and Expo 2008* (IEEE, ed.), p. 697:700, April 2008.

[11] D. Luebke, "Cuda: Scalable parallel programming for high-performance scientific computing," in *5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI 2008*, p. 836:838, May 2008.

[12] S. Sukittanon and E. Atlas, "Modulation frequency features for audio fingerprinting," in *IEEE, International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2, (University of Washington USA), pp. 1773–1776, 2002.

[13] S. Subramanya, R. Simha, B. Narahari, and A. Youssef, "Transform-based indexing of audio data for multimedia databases," in *International Conference on Multimedia Applications*, 1999.

[14] S. Sigurdsson, K. B. Petersen, and T. Lehn-Schioler, "Mel frequency cepstral coefficients: An evaluation of robustness of MP3 encoded music," in *International Symposium on Music Information Retrieval (ISMIR)*, 2006.

[15] E. Zwicker and H. Fastl, *Psycho-Acoustics. Facts and Models*. Springer, 1990.

[16] S. Sukittanon, A. L.E., J. Pitton, and K. Filali, "Improved modulation spectrum through multi-scale modulation frequency decomposition," in *IEEE, International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 4, (University of Washington USA), pp. 517–520, 2005.

[17] J. Herre, E. Allamanche, and O. Hellmuth, "Robust matching of audio signals using spectral flatness features," *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 127–130, 2001.

[18] R. P. Hellman, "Asymmetry of masking between noise and tone," *Perception and Psychophysics*, vol. 11, pp. 241–246, 1972.

[19] S. Pauws, "Musical key extraction from audio," in *International Symposium on Music Information Retrieval ISMIR*, 2004.

[20] P. Cano, E. Battle, T. Kalker, and J. Haitsma, "A review of algorithms for audio fingerprinting," *Multimedia Signal Processing,IEEE Workshop on*, pp. 169–167, December 2002.

[21] A. C. Ibarrola and E. Chavez, "A robust entropy-based audio-fingerprint," in *IEEE International Conference on Multimedia and Expo (ICME2006)*, pp. 1729–1732, July 2006.

[22] O. Hellmuth, E. Allamanche, M. Cremer, T. Kastner, C. Neubauer, S. Schmidt, and F. Siebenhaar, "Content-based broadcast monitoring using MPEG-7 audio fingerprints," in *International Symposium on Music Information Retrieval ISMIR*, 2001.

[23] E. Allamanche, J. Herre, O. Helmuth, B. Froba, T. Kasten, and M. Cremer, "Content-based identification of audio material using MPEG-7 low level description," in *International Symposium on Music Information Retrieval ISMIR*, 2002.

[24] A. Guo and H. Siegelmann, "Time-warped longest common subsequence algorithm for music retrieval," in *5th International Conference on Music Information Retrieval (ISMIR)*, 2004.

[25] E. Batlle, J. Masip, and E. Guaus, "Amadeus: a scalable HMM-based audio information retrieval system," in *First International Symposium on Control, Communications and Signal Processing*, pp. 731– 734, March 2004.

[26] E. Batlle, J. Masip, E. Guaus, and P. Cano, "Scalability issues in an hmm-based audio fingerprinting," in *IEEE International Conference on Multimedia and Expo (ICME 2004)*, pp. 735 – 738, July 2004.

[27] A. Ramalingam and S. Krishnan, "Gaussian Mixture Modeling using short time fourier transform features for audio fingerprinting," in *IEEE International Conference on Multimedia and Expo (ICME2005)*, pp. 1146 – 1149, July 2005.

[28] H. Lin, Z. Ou, and X. Xiao, "Generalized time-series active search with kullbak-leibler distance for audio fingerprinting," *IEEE Signal Processing Letters*, vol. 13, pp. 465 – 468, August 2006.

[29] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, and W. Hwu, "Optimization principles and application performance evaluation of a multithreaded gpu using cuda- principles and practice of parallel programming," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, (USA), p. 73:82, 2008.

[30] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex fourier series," *Math. Comput.*, vol. 19, p. 297301, 1965.

[31] J. Bercher and C. Vignat, "Estimating the entropy of a signal with applications," *IEEE Transactions on Signal Processing*, vol. 48, pp. 1687–1694, June 2000.