Fast GPU Audio Identification

Natalia Miranda[1], Fabiana Piccoli[1], Edgar Chávez[2,3], and Antonio Camarena-Ibarrola[2]

[1] LIDIC-Universidad Nacional de San Luis, San Luis, Argentina
[2] U. Michoacana México
[3] CICESE, México

Abstract. Audio identification consist in the ability to pair audio signals of the same perceptual nature. In other words, the aim is to be able to compare an audio signal with a modified versions perceptually equivalent. To accomplish that, an audio fingerprint is extracted from the signals and only the fingerprints are compared to asses the similarity. Some guarantee have to be given about the equivalence between comparing audio fingerprints and perceptually comparing the signals.
In designing AFPs, a dense representation is more robust than a sparse one. A dense representation also imply more compute cycles and hence a slower processing speed.
To speedup the computing of a very dense audio fingerprint, able to stand stable under noise, re-recording, low-pass filtering, etc., we propose the use of a massive parallel architecture based on the Graphics Processing Unit (GPU) with the CUDA programming kit. We prove experimentally that even with a relatively small GPU and using a single core in the GPU, we are able to obtain a notable speedup per core in a GPU/CPU model. We compared our FFT implementation against state of the art CUFFT obtaining impressive results, hence our FFT implementation can help other areas of application.

1 Introduction

Audio-Fingerprints (AFPs) are essential characteristics of digital audio streams used to score the perceptual similarity between audio signals. Among other tasks, AFPs are used in broadcast monitoring, [25], automatic metadata labeling from a central database and querying by example, where an excerpt of an unknown song (possibly captured in a noisy environment, such as a bar or pub) is used to identify it [9, 30] and for automatic score following [4]. AFP's are mature technologies used as software commodities by a very large number of applications of economic importance.

In designing AFPs for such uses 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 computer 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. In this paper the goal is to obtain a better throughput for online processing of a multi-stream source.

1.1 High Performance Computing

A side effect of the gaming technology is the existence of portable, affordable, massive parallel devices conceived to speed up online rendering[3, 20]. The Graphics Processing Unit (GPU) can provide up to 50 times the processing power, compared to the host computer[18].

Since its inception, the GPU was used as a dedicated device for speeding up graphics processing applications, 3D video gaming, rendering, etc. The progress of the GPU was faster than for CPU, probably due to a smaller instruction set and single precision arithmetic[16, 20]. 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. [14, 6, 19].

2 Characteristics of an AFP

To accomplish the tasks enlisted above, in the introduction, an AFP should be robust to signal degradations such as noise mixing, equalization, cropping and time shifting. An AFP should also be compact and determined with as little computational effort as possible. An AFP system should also be scalable, that is, it should be able to operate with very large databases, conditioned by a good indexing technique.

2.1 Feature extraction

The first thing an audio-fingerprinting system has to do is to extract features from the signal. Some AFP systems extract signal features directly in time domain as in [15] where the sign of the time derivative of the signal was found to

be robust to lossy compression and low-pass filtering. In [13] the entropy of the audio signal is computed every second and from that the sign of its derivative with respect to time is coded in an extremely compact AFP which was found to be robust to lossy compression and low-pass filtering and scaling, but not equalization. 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 [28] and some Discrete Wavelet Transforms like Haar's and Walsh-Hadamard's [27].

Looking for more relevant features of audio signals a variety of perceptual features have been assessed such as the Mel-frequency Cepstral coefficients (MFCC) [26]; Loudness [31]; the Joint Acoustic and Modulation Frequency (JAMF) [28, 29]; the Spectral Flatness Measure (SFM) [12]; the Spectral Crest Factor (SCF) [12]; tonality [10] and chroma values [21] among others [5]. In [24] it was shown that Normalized SSC can be more robust than MFCC and tonality for lossy compression and equalization. In [28] it was reported that the Normalized JAMF had superior robustness than a spectral estimate for compression and equalization. In [12] it was reported that SFM had superior robustness than Loudness and SCF as well.

2.2 Audio-fingerprint modeling

Some AFP systems model the songs in a way that best serves the purpose of the application for which it has been designed. For example, *Trajectories*, also known as *traces*, are sequences of feature vectors extracted at equally spaced instants and stored in a list of vectors or in a table; *Statistics* represent an audio signal using computed properties such as mean, variance, minimum and maximum values of the feature vectors [11]; *Codebooks* store a small number of representative code vectors disregarding the temporal evolution of the audio signal; *Strings* are basically trajectories turned into long strings of integers through vector quantization enabling the use of flexible string matching techniques; *Hidden Markov Models (HMM)* model non-stationary stochastic processes (e.g., songs). The HMM model of a specific song reports the probability that the query matches the candidate song [1, 2]; *Gaussian Mixture Models (GMM)* work on the premise that songs are the result of a combination of Gaussian components [22, 17]. The technique described here differs to these approaches in that it does not reply on specific domain knowledge, and is therefore more widely applicable.

3 Entropy of a signal as a relevant perceptual feature

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 to take value x_i.

Over the time the audio signal contains different amount of entropy, distinguishing between melodic, vocal, noise, etcetera. 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. If the signal is lossy compressed or low pass filtered the corresponding entropy curve is also shifted and the relative changes are preserved as illustrated in figure 1. A horizontal shift to the right is also observed due to the mp3 compression.

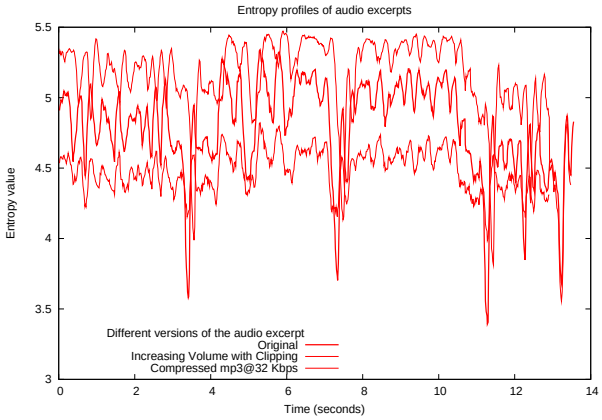


Fig. 1. Entropy curves of a excerpt of a song and a scaled (with clipping) and a lossy compressed (mp3@32Kbps) versions of it

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, as observed in figure 2. 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 [9] in [13] obtaining good results for low pass filtering, lossy compression and volume changes. For re-recording or equalization the results were not encouraging. Pursued in the work

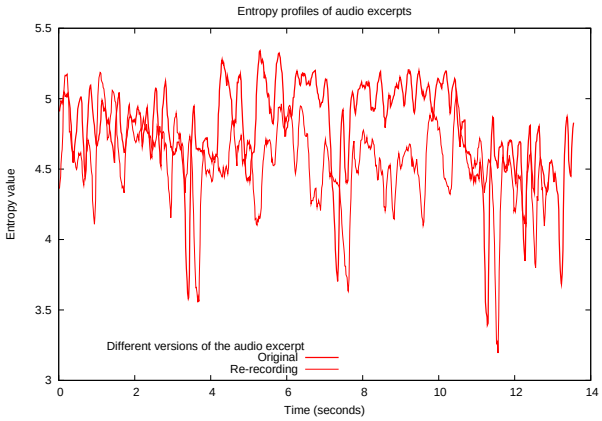


Fig. 2. Entropy curves of an excerpt of a song and a re-recorded versions of it

presented here, the entropy calculation is undertaken in the frequency domain, with logarithmic bands used to offset the effect of equalization.

4 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. This is illustrated in figure 3 where only some of the 24 bands are shown to avoid overcrowding the figure.

The subbands can be obtained with a standard filter bank tuned with the corresponding frequencies of the bark scale [8].

4.1 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.

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

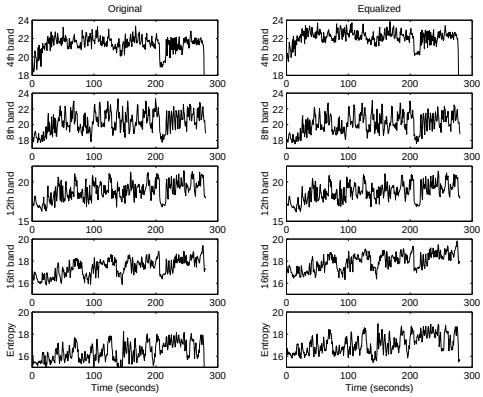


Fig. 3. Entropy profiles for individual bands in the Bark scale.

5 Comparing Songs

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 precomputed 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.

5.1 Probabilistic Pairing Pseudo Metric

Lets assume we have a *base* distance $d(x, y)$ 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 $n < m$:

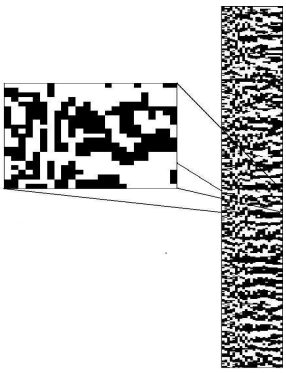


Fig. 4. An illustration of the probabilistic pairing pseudo metric. The smaller song/excerpt is shifted to search for the best match, and this match is reported as the "distance". In the figure, an actual fingerprint of a song is presented

$$D(x,y) = \min_{d(x[i,i+n]} y[1,n]] \; \forall \; 1 \leq i \leq m-n \tag{2}$$

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. Figure 4 shows how probabilistic pairing is used to shift the query (excerpt) to find the best match.

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.

6 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 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 [23]. 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 assignement. 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 sincronized) 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 componentt solve a problem over 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 instruction specify parallel code and data structures to be executed in the device. Those computing devices 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, on the other hand, 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 of the data device. 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 thread 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.

7 Parallel Multi-MBSES

Figure 5 illustrate the parallel architecture for the digital signature dubbed $MBSES_p$. As said before the problem is particularly well suited for massive parallel processing.

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 6 illustrates.

In both $MBSES_p$ and *Multi-*$MBSES_p$ schemas the massive parallel architecture can be applied. Several parameters need to be adjusted. In this work we discuss two crucial parts of the processing, computing the Hanning window and computing the fast Fourier transform.

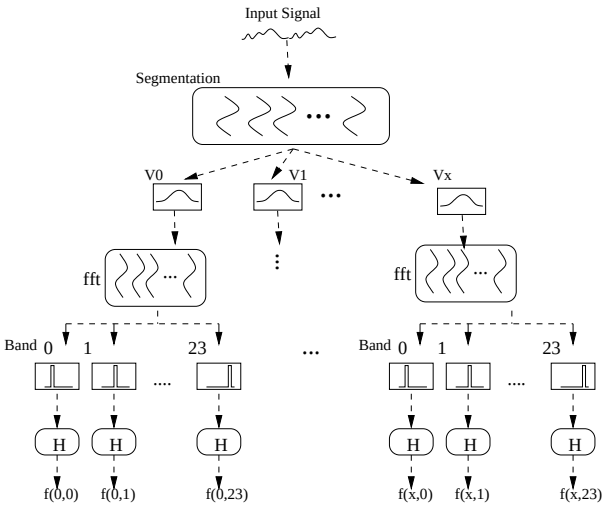


Fig. 5. Architecture of $MBSES_p$

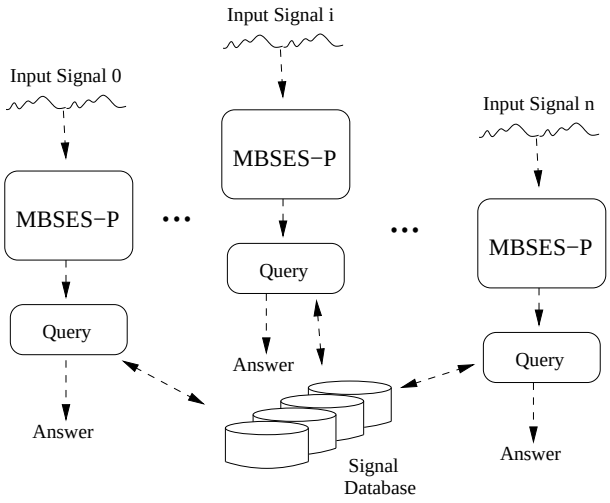
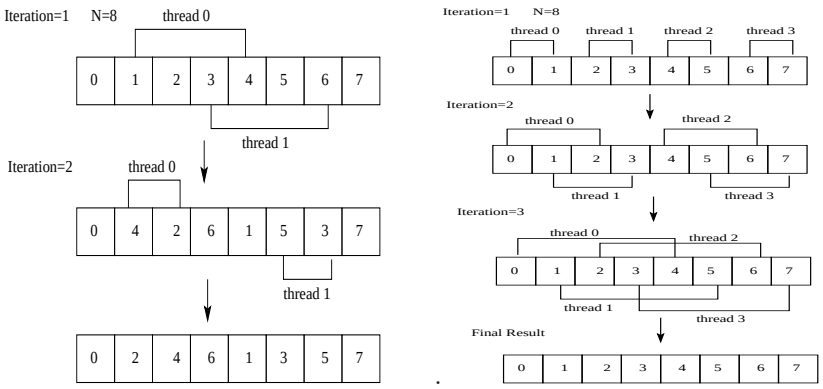


Fig. 6. Multi signal $MBSES_p$ system

7.1 Fast Fourier Transform

In the CUDA repository there is a library for parallel computing the FFT, the CUFFT. We implemented directly the FFT based on the original algorithm of Cooley and Tukey [7]. 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 in 512 threads to be executed in parallel. We first compute the bit-reverse vector in a first stage 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 much larger than the number of available threads T, then each thread will swap $\frac{\frac{N}{4}}{T}$ elements. This is illustrated in figure 7(a)



(a) Bit Reverse (b) fft

Fig. 7. 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. Figure 7(b) the procedure is sketched for each iteration.

7.2 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.

8 Experimental Results

For a comparative analysis we selected a sequential CPU implementation of the algorithms, the fastest machine available for experiments had the following

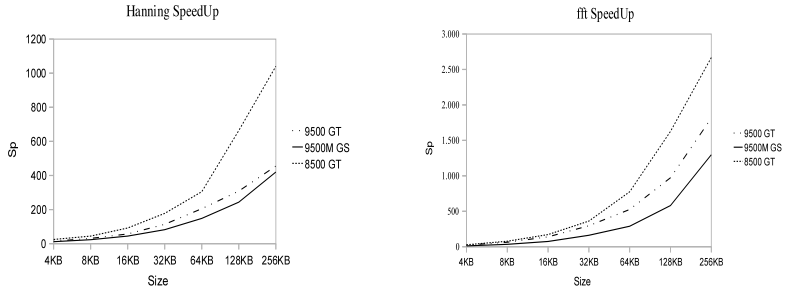
characteristics. Intel core 2 Duo E8200, with 2GB of RAM. We used three different GPU models for comparison. The 8500 GT, 9500 GT and 9500 GS. They had the following common characteristics.

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

With the following differences.

GeForce	8500 GT	9500 GT	9500M GS
Global Memory	512MB	256MB	512MB
Multiprocessors	2	4	4
Cores	16	32	32
Central Clock	450 MHz	500 MHz	475 MHz

The results shown for the speedup are the average over several runs. Figures 8(a) y 8(b) show the speedup for the Hanning window computation and the FFT. In all the cases we used the maximum number of available threads.



(a) Hanning (b) fft

Fig. 8. Speedup of Hanning and FFT computations

The overall speedup is not as impressive because our model is mixed, we used a CPU/GPU model where the data had to be moved from/to the GPU main memory. It is difficult to measure the isolated speedup but the results are encouraging. Data transfers impose a severe restriction on the performance. We are currently solving all the steps of the $MBSES_p$ in a pure GPU model.

We compared our implementation with the state GPU based library CUFFT, available in the CUDA showroom. Our implementation surpass the efficiency of the state of the art. Figure 9 shows the comparison.

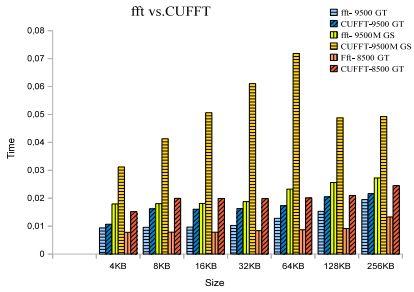


Fig. 9. FFT vs CUFFT

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

9 Conclusions, Remarks and Future Work

We have proved that even using a single core in the GPU, we are able to compute the most challenging parts of the MBSES implementation with a speedup of about 2. It is possible to assemble a GPU cluster consisting in an array of GPUs giving extreme speedups. Moreover, modern GPUs will be multicore, increasing the speedup expectatives.

We are currently working on implementing all the steps in the $MBSES_p$ computation in a pure GPU model to avoid data transfers slowing down the process. We are also implementing a massively parallel version of a main memory metric index to support proximity queries.

References

1. Elloi Batlle, Jaume Masip, and Enric Guaus. Amadeus: a scalable HMM-based audio information retrieval system. In *First International Symposium on Control, Communications and Signal Processing*, pages 731– 734, March 2004.
2. Eloi Batlle, Jaume Masip, E. Guaus, and Pedro Cano. Scalability issues in an hmm-based audio fingerprinting. In *IEEE International Conference on Multimedia and Expo (ICME 2004)*, pages 735 – 738, July 2004.
3. I. Buck. Gpu computing with nvidia cuda. *SIGGRAPH '07: ACM SIGGRAPH 2007 courses ACM,*, 2007. New York, NY, USA.
4. Antonio Camarena-Ibarrola and Edgar Chavez. Real time tracking of musical performances. In *MICAI*, volume To appear, 2010.
5. Pedro Cano, E. Battle, T. Kalker, and J. Haitsma. A review of algorithms for audio fingerprinting. *Multimedia Signal Processing,IEEE Workshop on*, pages 169–167, December 2002.
6. W. Chen and H. Hang. H.264/avc motion estimation implementation on compute unified device architecture (cuda). In IEEE, editor, *IEEE International Conference on Multimedia and Expo 2008*, page 697:700, April 2008.

7. J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19:297301, 1965.
8. Zwicker E. Subdivision of the audible frequency range into critical bands. *The Journal of the Acoustical Society of America*, (33), 1961.
9. J. Haitsma and T. Kalker. A highly robust audio fingerprinting system. In *International Symposium on Music Information Retrieval ISMIR*, 2002.
10. R. P. Hellman. Asymmetry of masking between noise and tone. *Perception and Psychophysics*, 11:241–246, 1972.
11. 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.
12. Jurgen Herre, Eric Allamanche, and Oliver Hellmuth. Robust matching of audio signals using spectral flatness features. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 127–130, 2001.
13. Antonio C. Ibarrola and Edgar Chavez. A robust entropy-based audio-fingerprint. In *IEEE International Conference on Multimedia and Expo (ICME2006)*, pages 1729–1732, July 2006.
14. 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)*, page 48:55, July 2008.
15. Frank. Kurth and Roman Scherzer. A unified approach to content-based and fault tolerant music recognition. In *114th AES Convention, Amsterdam, NL.*, 2003.
16. M.D. 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*, page 1111:1120, April 2008.
17. Hui Lin, Zhijian Ou, and Xi Xiao. Generalized time-series active search with kullbak-leibler distance for audio fingerprinting. *IEEE Signal Processing Letters*, 13(8):465 – 468, August 2006.
18. D. Lloyd, C. Boyd, and N. Govindaraju. Fast computation of general fourier transforms on gpus. In *IEEE International Conference on Multimedia and Expo*, page 5:8, April 2008.
19. 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*, page 836:838, May 2008.
20. D. Luebke and G. Humphreys. How gpus work computer. *EEE Computer*, 40(2):96:100, Feb 2007.
21. Steffen Pauws. Musical key extraction from audio. In *International Symposium on Music Information Retrieval ISMIR*, 2004.
22. Arunan Ramalingam and Sridhar Krishnan. Gaussian Mixture Modeling using short time fourier transform features for audio fingerprinting. In *IEEE International Conference on Multimedia and Expo (ICME2005)*, pages 1146 – 1149, July 2005.
23. S. Ryoo, C.I. Rodrigues, S.S. Baghsorkhi, S.S. Stone, D.B. Kirk, and W.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*, page 73:82, USA, 2008.

24. Jin S. Seo, Minho Jin, Sunil Lee, Dalwon Jang, Seungjae Lee, and Chang D. Yoo. Audio bingeprinting based on normalized spectral subband centroids. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2005.
25. Seungwon Shin, Oanjin Kim, Jongweon Kim, and Jonguk Choil. A robust audio watermarking algorithm using pitch scaling. In *14th International Conference on Digital Signal Processing*, volume 2, pages 701 – 704, 2002.
26. Sigurdur Sigurdsson, Kaare Brandt Petersen, and Tue Lehn-Schioler. Mel frequency cepstral coefficients: An evaluation of robustness of MP3 encoded music. In *International Symposium on Music Information Retrieval (ISMIR)*, 2006.
27. 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.
28. S. Sukittanon and E. Atlas. Modulation frequency features for audio fingerprinting. In *IEEE, International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages 1773–1776, University of Washington USA, 2002.
29. S. Sukittanon, Atlas L.E., J.W. Pitton, and K. Filali. Improved modulation spectrum through multi-scale modulation frequency decomposition. In *IEEE, International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 4, pages 517–520, University of Washington USA, 2005.
30. 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.
31. Eberhard Zwicker and Hugo Fastl. *Psycho-Acoustics. Facts and Models*. Springer, 1990.