Computer and Information Science; Vol. 8, No. 1; 2015
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Accelerating the Detection of Spectral Bands by ANN-ED on a GPU

Yassine El Hafid[1,2], Abdessamad Elrharras[1,3], Karim Guennoun[1], Abdelkader Amri[2] & Mohammed Wahbi[1]

[1] Laboratoire du Génie des Systèmes, SIRC/LaGeS-EHTP Casablanca, Morocco

[2] Laboratoire de Physique des Hautes Energies et d'Informatique Scientifique, Faculté des sciences Ain Chok Université Hassan II, Casablanca, Morocco

[3] Laboratoire de Traitement de l'Information Faculté des Sciences Ben Msik Université Hassan II Casablanca, Morocco

Correspondence: Yassine El Hafid, Ecole Hassania des Travaux Publics, Km 7, Route d'El Jadida, BP 8108 Oasis, Casablanca, Morocco. Tel: 212-663-176-499. E-mail: el_hafid_yassine@yahoo.fr

Received: December 30, 2014 Accepted: January 6, 2015 Online Published: January 28, 2015

doi:10.5539/cis.v8n1p95 URL: http://dx.doi.org/10.5539/cis.v8n1p95

Abstract

Spectrum sensing is the most important technique used to implement cognitive radio; this approach allows opportunistic and dynamic allocation of spectral bands. Among the methods used for detection, there are Artificial Neural Networks (ANN) and Energy Detection (ED); those exploit the signals coming from a Fast Fourier Transformed block (FFT). In this work, we focus on improving the performance of these three blocks by performing parallel computing, and considering the fusion of the two detectors ANN and ED. In this context, we implement three algorithms on GPU, which consist on exploiting the large number of cores to perform parallel calculation. The experimental results are compared with those obtained for CPU implementations. Our study presents how calculations distribution on GPU cores influences the global performance, and how to reduce execution time by optimizing data transfer. Furthermore, by exploiting the fine-grained parallel processing, and using a suitable choice of parameters, we find a considerable advantage of GPUs compared to CPUs, specifically for high data volumes.

Keywords: cognitive radio, spectrum sensing, FFT, energy detection, artificial neural networks, CPU, GPU, CUDA

1. Introduction

The use of the frequency spectrum increases continuously, due to the growing needs of wireless technologies and their various services (Kolodzy & Avoidance, 2002). The current management of spectrum bands is based on a static allocation that cannot support the growing demand. In this context, J. Mitola (Mitola, 1999) introduced the approach of cognitive radio, which allows efficient spectral band exploitation, by an opportunistic using of frequency bands. It means that spectrum management must move from classical way, to dynamic usage. The evolution given to this management is based on separating users into two categories:

- The Primary User (PU), which has a license to use a defined band.

- Secondary User (SU), that does not have a license but he can use a free band left by primary user.

There are several methods in the literature to perform this detection, including the matched filter (Fuchs, 2009), the cyclostationary detection (Renard, Verlant-Chenet, Dricot, De Doncker, & Horlin, 2010), the energy detection (Plata & Reátiga, 2012), and artificial neural networks (Lee & Koo, 2010).

In spectrum sensing, we shall take into consideration several aspects, such as the noise uncertainty, channel fading and potential obstacles around the secondary users (Hossain, Abdullah, & Hossain, 2012). Reducing effects of the factors above improve detection accuracy, particularly with cooperative detection, wherein a SU use several detection techniques. These techniques have to submit their decisions to a fusion center, which makes the final one. In this article, we depict two methods: the energy detection; and neural networks.

The two methods above require significant resources calculations, to achieve the treatment within an acceptable duration. A parallel approach seems a priori advantageous to reduce the computation time, and to increase the detection reliability.

In recent years, 3D games have exploited the graphics card processors to perform massive parallel computations. Indeed, in comparison to conventional processors, consumer graphics processors have benefited from significant development with at least a factor of 400 in terms of cores numbers and a factor of 6.5 in terms of computing performance. For example, the Figure 1 presents a performance comparison of several generations of CPU processors from Intel and Nvidia GPU graphics processor. The comparison is based on performance calculation of floating-point numbers in single and double precision.

It is clear that the numerical simulations of cognitive radio are a promising application for this platforms type. The GPU could handle a very large number of interacting transmitters / receivers that require storage space and a high-speed operation. Therefore, improving the spectrum sensing rapidity is a first step in simulating the behavior of a large number of smart radios terminals.

In the second section, we present different models and methods for spectrum sensing. In the third, we introduce the architecture of Nvidia Cuda parallel programming. The fourth one, we present detailed implementations of three blocks: FFT, ANN and ED, parallel on GPU and sequential on CPU. In the fifth part, we show the experimental results and a discussion of their interpretation. Finally, we conclude the paper and we propose future prospects of our work.

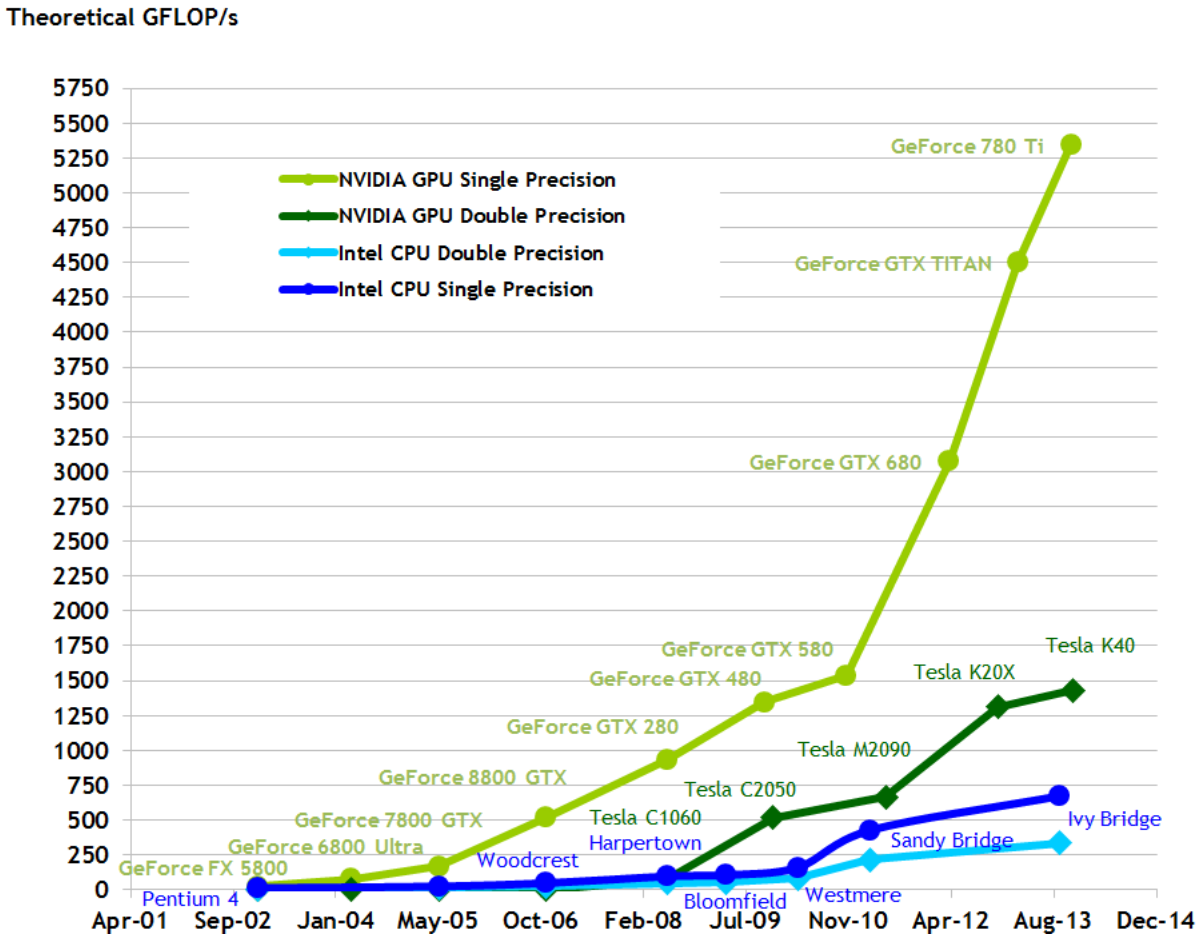

Figure 1. Single and double precision processing time of float numbers on several kind of GPUs and CPUs processors (NVIDIA, August 2014a)

2. Methods and Models for Detecting Free Bands

Detecting free bands is the most important step in intelligent radio equipment. In this section, we formulate mathematically the identification of the PU existence, by browsing the different techniques used for this purpose.

2.1 Mathematical Model of Detecting Free Bands

The problem of detecting free bands, is based on detecting the PU presence, this later emits a signal in a specific frequency channel. We can formulate the spectrum analysis problem as follows:

$$y(t)=\begin{cases} n(t) & : H_0 \\ \epsilon * x(t)+n(t) & : H_1 \end{cases} \text{ with } \epsilon \in \{0,1\} \quad (1)$$

y(t) : the received signal.

x(t) : the signal being deterministic or random but unknown.

n(t) : the noise present in the channel.

The hypothesis presents the case where the received signal contains the noise alone. While the hypothesis, refers to the ability of detecting the PU presence.

2.2 Matched Filter Detection

Detection by matched filter is the most optimal technique, in the sense of efficiency. This method has consisted in a linear filter, designed to maximize its output, which presents the signal to noise ratio (SNR) for a given input signal (Fuchs, 2009). With this system, secondary users (SU) must have complete knowledge about signal transmitted by the PU. This information includes the order and the type of modulation, the carrier frequency, pulse shape, and the packet format. The adapted approach for matched filtering is equivalent to a correlation scheme Figure 2; wherein a signal is convolved with a filter, which impulse response is defined by equation (2).

$$h(t) = \begin{cases} s(T-t); & si\ 0 \le t \le T \\ 0 \quad ; & elsewhere \end{cases} \quad (2)$$

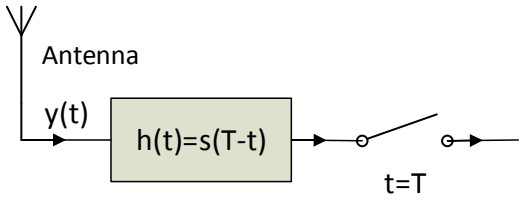

Figure 2. Linear filter based on correlation

2.3 Cyclostationary Detection

Generally, wireless transmissions have cyclostationarity characteristics based on their: data rate, modulation type, and the carrier frequency. In opposition to noise, which is random phenomenon, most of communication signals can be modeled as cyclostationary, since they present hidden periodicities in their signal structures (Ning, Sohn, & Kim, 2009); such periodicities reveal the PU presence.

Compared to others spectrum-sensing methods, the exploitation of these hidden periodicities in the primary signal, makes this method immune to the high noise. However, this method requires a powerful analog to digital converter (ADC) which results in a high-energy consumption.

2.4 Detection Method by Detecting Energy

In the context of energy detection method (Figure 3), we measure the energy (3) of the received signal, in a specific moment and at a predetermined frequency band. The primary signal detection is made by comparing the detector output with a threshold, which exhibits noise energy.

$$E = \frac{1}{N}\sum_{i=1}^{N} |Y(f)|^2 \quad (3)$$

First, the ADC digitizes the detector input, and then a pass band filter selects the desired channel. After that, the filtered signal Y (k) is transformed into the frequency dimension Y (f) through the FFT block, next it is squared and integrated over the observation interval. Comparing integrator output and threshold λ determines whether the PU is present or absent.

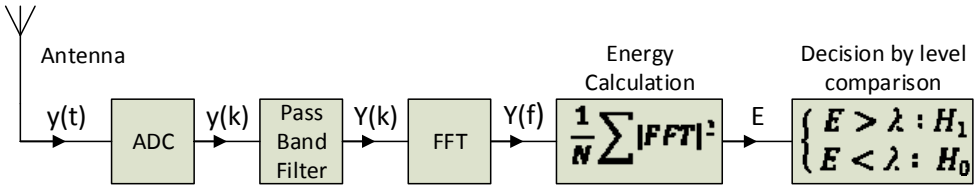

Figure 3. Energy detection diagram

2.5 Artificial Neural Networks

Noise and fading channel influence the final decision result. These parameters are coupled, have non-linear effects, and are unstable over time. Therefore, it is very difficult to identify the PU presence. As cited in (A Elrharras, Saadane, Wahbi, & Hamdoun, 2014), using ANN has improves the quality of detection. An analytical study of ANN model is detailed in this section.

2.5.1 The Formal Neuron

ANN imitates the structures of biological neurons. An artificial neuron is a mathematical representation of the biological one, which usually has multiple inputs and a single output Fig.4. The actions of excitatory synapses are represented by the numerical coefficients (synaptic weight) associated with the inputs. The numerical values of these coefficients are adjusted in a learning phase. In its simplest version, an artificial neuron calculates the weighted sum of the received inputs plus the bias, and then an activation function is applied to calculate output value.

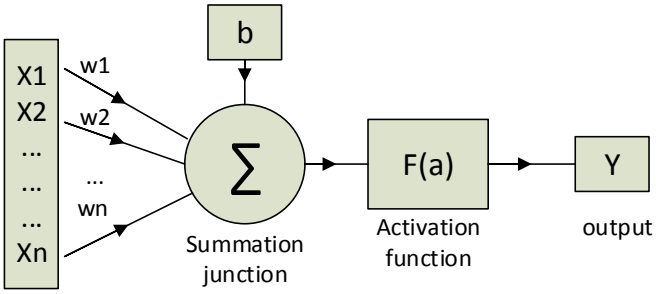

Figure 4. Formal Neuron

The neuron shown in Figure 4 has n inputs, designated as (X1, X2... Xn). Lines connecting these inputs to the summing junction are assigned with weights noted (W1, W2... Wn}. The activation function F (a) of McCulloch and Pitts neuron model (McCulloch & Pitts, 1943) is a threshold function. However, linear and sigmoid functions are also used in different situations. The output signal "Y" of the neuron is given by formula (4).

$$y = f(\sum W_i * X_i + b) \quad (4)$$

2.5.2 Multi-Layer Perceptrons

The Multi-Layer Perceptron (MLP) (Tang, Zhang, & Lin, 2010) is a set of neurons in layers Figure 5. The neurons output signals of a layer are the inputs signals for the next layer neurons. The general architecture of the MLP is represented by the neurons in consecutive layers, the first is the input layer, the last is the output layer and the intermediate layers are called hidden layers.

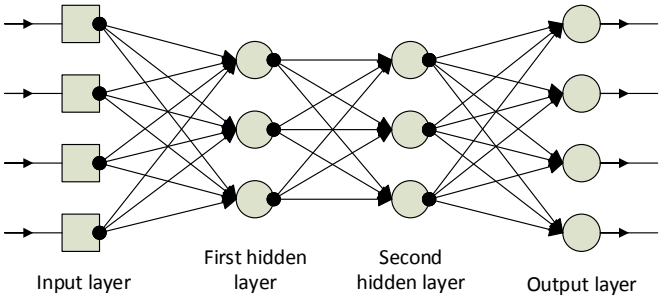

Figure 5. Multi-layer perceptrons (PMC)

In this standardized architecture, the neural layers are completely interconnected. It is to say that each neuron of a layer is connected to all neurons of the further layer.

2.5.3 A Model of Detection by a Neural Network

In the context of free bands detection by artificial neural networks (Tang et al., 2010), we receive a signal that will be classified into two classes.

First, the detector digitized input signal y (t) and select the desired band by a pass band filter. Then, the filtered signal is transformed to the frequency domain Y (f) through the FFT block, and then it is injected into ANN for making the decision Figure 6.

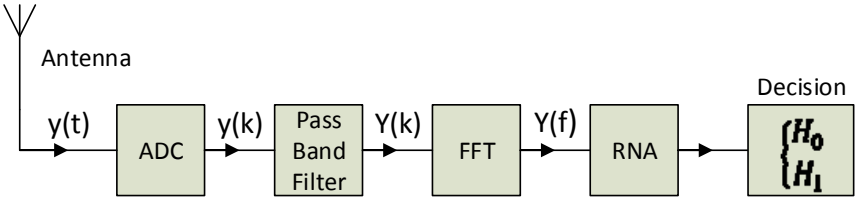

Figure 6. ANN detection diagram

2.6 The Architecture of NVIDIA CUDA Platform

GPU programming can be done in a conventional workstation or laptop computer with a graphics card. There are usually main processors called Host or CPU having a clock frequency in the range of 1.8 to 3.2 GHz. Concerning the graphics processor called Device, it is clocked from 700Mhz to 900Mhz, views more according versions. In a recent midrange machine, the number of computing cores from GPU is a multiple of 100 compared to a CPU. The PCI bus provides communication between the Host and Device, where we can connect several cards on the same bus.

At the architecture level, the RAM of the GPU, a GDDR5 has a higher bandwidth than the Host does usually GDDR3, fostering computation speed at the GPU. Each processor cannot access to the other resources, which requires us to make copies of the host RAM data to Device RAM and vice versa. The transfer operation consumes a lot of time during calculations, and limits the size of the data.

However, the version 6.5 of the Framework Cuda published in August 2014 (NVIDIA, August 2014a), allows Host and Device to see both RAMs as a unified memory, solving the problems mentioned above. In addition to the RAM GPU, each Cuda SM (Streaming Multiprocessor) has a smaller cache memory of around a hundred kilobytes, but has a higher bandwidth of about 1.7TB / s.

2.7 Running Parallel Tasks on CUDA

CUDA platform uses the Host and Device to do computations. Indeed, each CUDA program is divided into several parts, sequential portions executed by the host and parallel portions executed by the device. The host uses the device as an auxiliary processor and utilizes a generic function called Kernel. Images of this function (threads) are executed by the GPU CUDA cores, with different parameters or input data.

However, the tasks of the CUDA platform are distributed over the cores in the form of a grid of blocks. Each block executes in parallel a specific number of threads, which has a shared hidden memory GPU, only visible to threads in the same block. However, in most applications, thread synchronization is necessary. Thus, CUDA Version 5 allows this synchronization, only for the threads in the same block. Regarding the synchronization between blocks, it is still possible using the kernels division into several parts, or using synchronization flags. The maximum numbers of threads and blocks depends on the generation of NVIDIA GPUs used, and is still valid for the execution of some functions in the recent CUDA 6.5 library.

3. Implementation

In this section, we implement ANN, ED and FFT algorithms, on an ordinary machine of video games, costing less than 2000$. The calculation unit used in the experimental part has as a main processor the Intel Quad Core clocked at 2.6GHz, 4GB DDR3 RAM, with a graphic gamer card GTX690; including dual GK107 GPU 3072 CUDA cores and 4GB DDR5 dedicated memory. The system runs on windows7, and the code is written in C using the Nvidia CUDA Framework 6.5 that provides the necessary libraries to access the GPU cores.

3.1 FFT Implementation

For the FFT algorithm implementation, we use the predefined functions cuFFT (NVIDIA, August 2014b) of NVIDIA CUDA; while for the sequential version, it is based on the Cooley and Tukey algorithm (Cooley & Tukey, 1965). Comparing the performance of the two algorithms is based on measuring execution time for several sizes of complex type inputs vectors. Furthermore, the library cuFFT provides several types of functions according to the requested application. We keep C2C function (NVIDIA, August 2014b) and we define the size by assigning discrete values in multiples of power of 2 (size = 2 ^ m) as recommended in (Cooley & Tukey, 1965).

3.2 Implementation of Neural Network

First, we implement two versions of MLP neural network programs. These networks are implemented with three layers each. The first is executed only on CPU, and the second on both CPU and GPU. The objective is to measure the calculation time depending on the intermediate layer size, ranging from 200 to 2000 neurons by step of 200. As for the input vectors and the primary layer, they have both size of 1000 each, the output layer contains one neuron to indicate one of the two hypotheses.

In both versions of the program, each neuron of a particular layer is associated with a bias and a weight vector. To simplify the modeling, we bring together all vectors associated with the neurons of a layer in a matrix. The product of input vector and weight matrix yields a vector that is added to bias vector, and then sigmoid activation function is applied on each element. Once the layer calculation is completed, the same computations are used with the result data as input for the next layer, the operations are repeated until reaching the final layer.

Parallelization of the computation on the Device is done mainly in the matrix product (Liu & Vinter, 2014) and in the calculation of the sigmoid function. We have defined a fixed number of threads per block and the number of blocks is calculated from the size of the layer divided by the number of threads. Furthermore, other functions are used for making the allocation, releasing memory device and transferring data from and to the Host. Some of these operations may penalize computing time. Therefore, to reduce the transfer time, we used the Pinned Memory mode that allows direct data transfer between the CPU and GPU RAMs as showing in Figure 7.

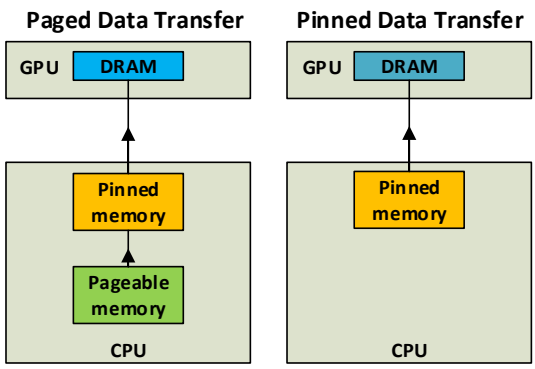

Figure 7. Pinned and paged memory data transfer

3.3 Energy Detection Implementation

The energy detection implementation is made in two parts: First, the square calculation product of each vector elements; Second, the square average calculation. A judicious choice of threads and blocks numbers has been specifically adopted for the product squared. The number of threads per block is set at 250; however, the blocks number ranges from 10 to 32000. The input vector size is defined by multiplying the threads and blocks numbers. Thus, an optimized kernel calculates the square of the product vector, harnessing the shared memory and minimizing data transfer rates to and from the GPU RAM. The calculation of average was implemented sequentially as a first step and could actually be optimized for best performance.

3.4 Implementation of ANN and ED Methods Fusion

The method of fusion was implemented as the main kernel. Represented by the Figure 8, runs at first the FFT program, and thereafter launches the two secondary kernels ANN and ED in parallel. The data comes from a file representing signals records as given vector size. GPU allocates memory one time outside the kernel; however, at each main kernel function calling, it executes data transfer between GPU CPU RAMs. At the end of ANN and ED functions, a logical operation makes the final decision.

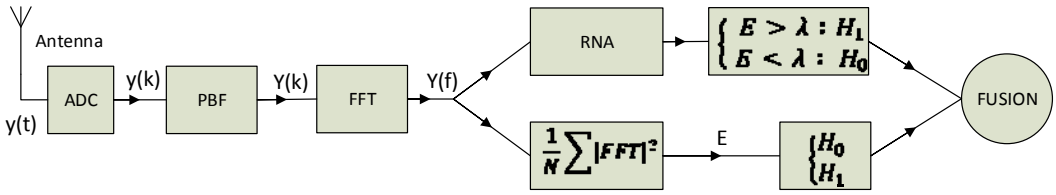

Figure 8. ED and ANN methods fusion

4. Result and Discussion

The experimental results show a correlation between the increases in cores numbers and computing performance improvement. However, this correlation is not observed for certain configurations. Thus, parameterization and optimization effort is indispensable for best results.

4.1 Analysis of FFT Performance Implementation

Figure 9 shows the FFT performance calculations on the GPU and CPU. It illustrates two intersecting curves: the x-axis represents the size of the vector expressed by the exponent power 2, and the y-axis represents the calculation time in microseconds. It shows that the two curves cross at a value m = 8, then for vector size less than 256, CPU gives better results than GPU and inversely for others sizes values. Moreover, for a typical detecting application, the vector size commonly used is 1024, and then for this valor we have demonstrated on Figure 10, 80% of FFT calculation time reduction by using the GPU.

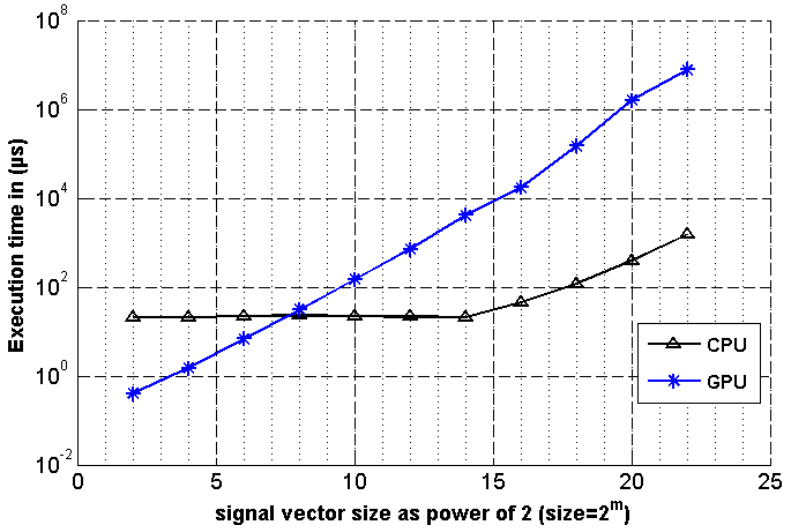

Figure 9. FFT calculations times on CPU and GPU according to input vector size

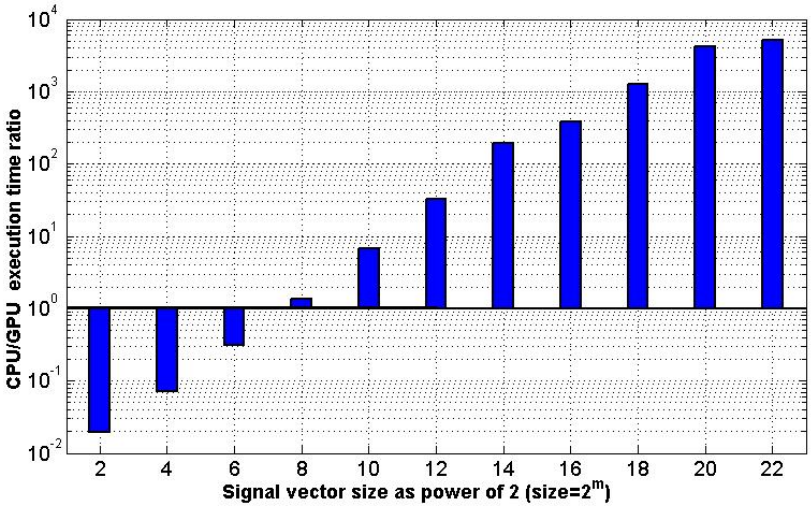

Figure 10. CPU/GPU FFT execution time ratio

4.2 Analysis of Implementation Performance ANN

Figure 11 is showing three superposed curves of ANN execution time on a logarithmic scale in microseconds. Both GPU curves represent the same algorithm with two memory allocation methods. It reveals that it is a significant GPU computing time reduction for Pinned type memory allocation, in comparing to those observed for CPU and GPU with conventional allocation method. During the experiments, the measured values for CPU and GPU with paged pool memory adopt random execution times, so we kept the minimum ones observed. We can explain this by loading and unloading memory pages from the hard drive. In addition, the CPU threads interactions of the operating system adopt a competitive behavior to memory access. Using the Pinned memory avoids the problems cited above, and allows better use of communication bus bandwidth between GPU and CPU memory via DMA (Direct Memory Access), which explains the improvement of execution time uniformity.

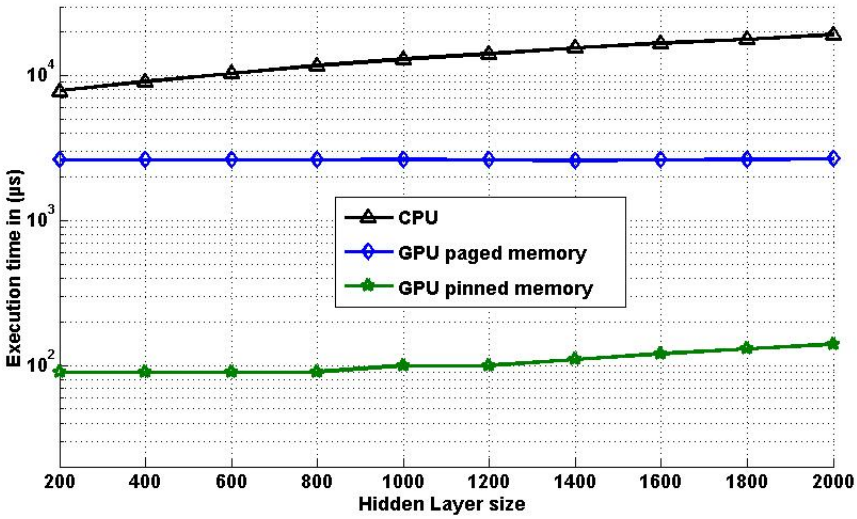

Figure 11. ANN executions times on CPU and GPU according to neurons numbers of hidden layer

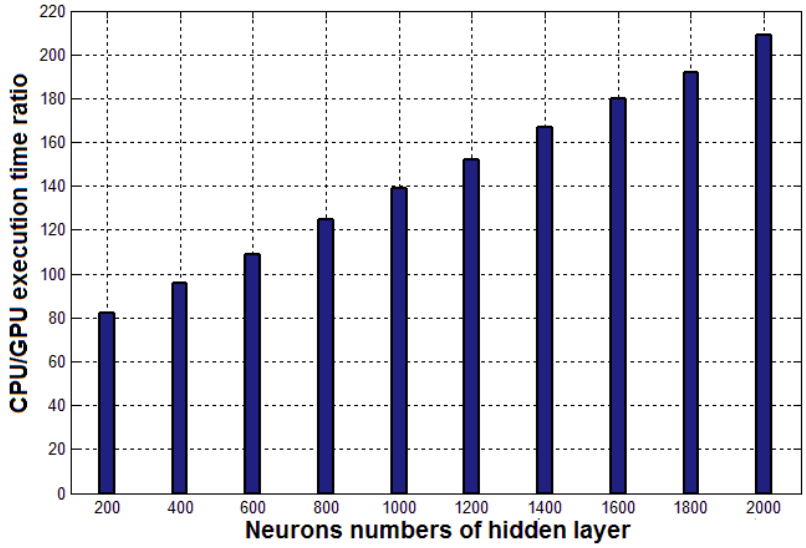

Figure 12. CPU/GPU Pinned ANN execution time ratio

Figure 12 is demonstrating the improved performance depending on the hidden layer size. As an example for a size of 2000, the performance reaches 210 times. However, we except reduction of their performance, due to the limited number of cores calculations.

4.3 Analysis and Comparisons of Performances ED Implementation

Fig. 13 is showing the energy calculation time of the signal, in function of the input vector size. Wherein we fix the threads number at 1000, while for blocks numbers, they have ranged according to equation (5).

$$vector\ size = N_{threads} * N_{Blocks} \quad (5)$$

For this specific value, close to 1024 which represent the maximum threads per block size. The GPU performance was reduced comparing to CPU, due to two reasons: the first, the small blocks number involves reduced cores number and limited using of GPU; the second is the CUDA cores clock speed is less than CPU one by a factor of 3 to 4.

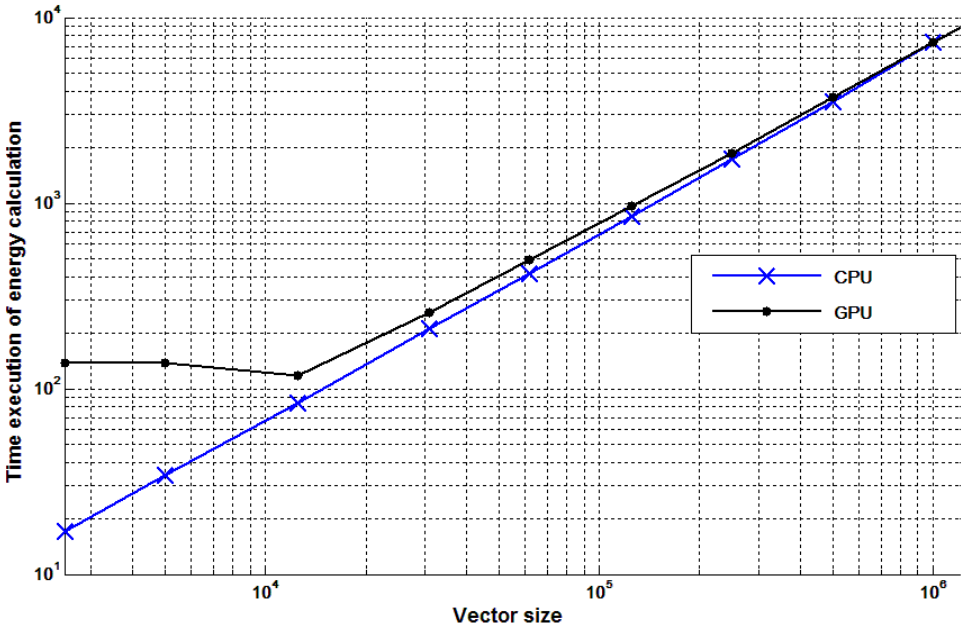

Figure 13. Time execution of energy calculation of the signal on CPU and GPU (1000 threads/ block) according the vector size

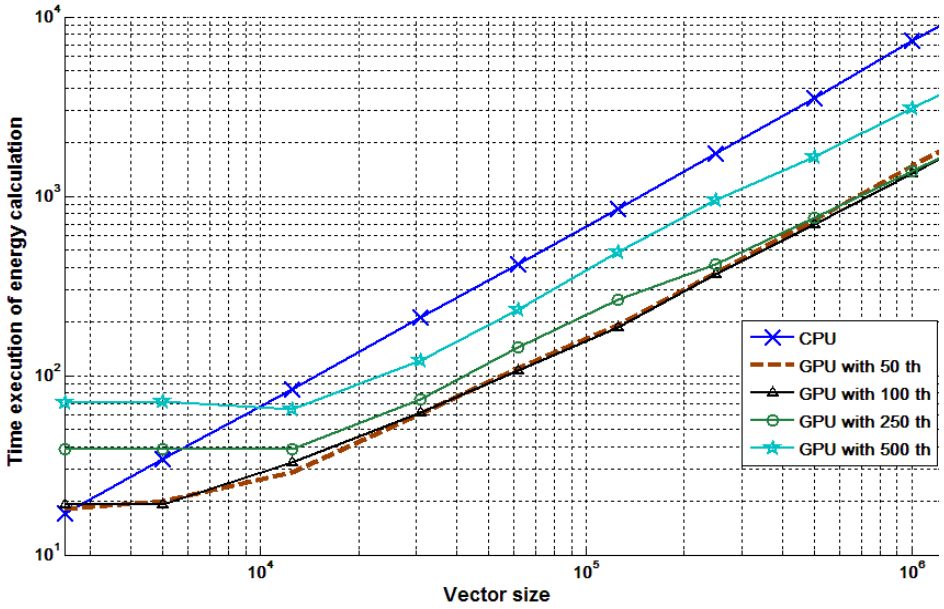

Figure 14. Signal energy calculation Time on CPU and GPU, according to the input vector size and threads numbers per blocs for the values 50, 100, 250 and 500

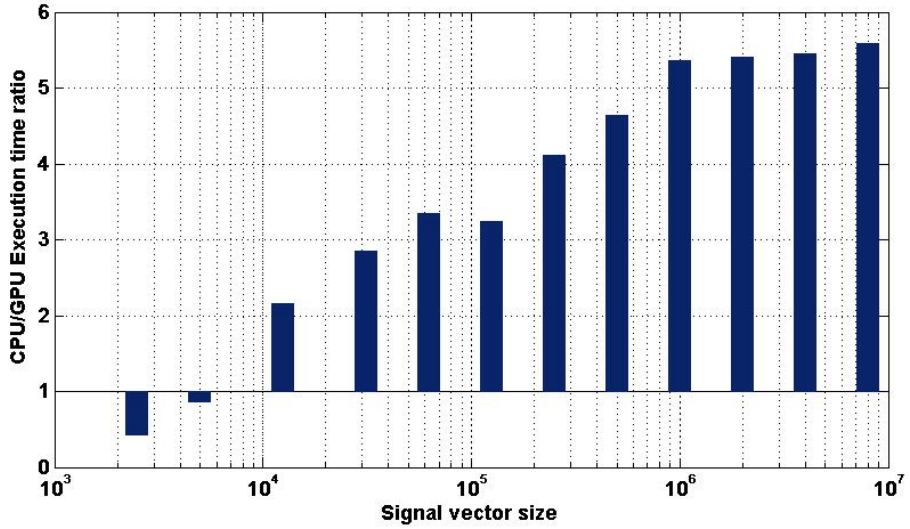

Figure 15. CPU/GPU calculation time ratio for threads number =250

In Figure 14, we tested different numbers of threads 50, 100, 250 and 500 and we observe a significant improvement. As example for $Nthreads = 250$ both CPU and GPU curves cross at $vector\_size = 6000$ beyond this value, for a $vector\_size = 250000$, we notice in Figure 15 a GPU performance factor is four times more than CPU.

However, we observe an anomaly for low threads number per block. Indeed, the performances for 50, 100 and 250 are almost identical. Nevertheless, for small vectors sizes, 50 and 100 are better than 250 threads per block. However, the maximum processed vector size is reduced, due to the limitation of maximum blocks number on calculation Grid, according to equation (6)

$$vector\_size\_max = N_{threads} * Nmax_{\ Blocks} \quad (6)$$

Furthermore, according to the Neyman Pearson criterion (Atapattu, Tellambura, & Jiang, 2010), maximizing the captured samples number improves the energy detector performance. According to the simulation sample proposed by (A. Elrharras, Saadane, El Aroussi, Wahbi, & Hamdoun, 2014), and in order to ensure a P_d (Probability of Detection) close to 98%, and for a $SNR = -12dB$, the input vector size must be greater than 2^{10}. Moreover, to keep the same P_d for $SNR = -15dB$, the input size must be up to 2^{12}. In this value range, GPU computing performance against the CPU can reach a factor of two.

4.4 The Result of the Fusion Implementation

Fusion methods on GPU are much more advantageous compared to CPU. The computation time is summative in the CPU case as shown in equations (7) and (8). As an example, for an input vector size of 1024 and an ANN with one hidden layer of 200 neurons, the gain in computation time measured is 71 as showed in Figure 16. Adding more fusion methods might enhance this benefit, and would probably allow increasing the reliability of the detector.

$$Tcpu = T_{FFT}cpu + T_{ANN}cpu + T_{ED}cpu \quad (7)$$

$$Tgpu = T_{data\ trasfert} + T_{FFT}gpu + \max\left(T_{ANN}cpu\ , T_{ED}cpu\right) \quad (8)$$

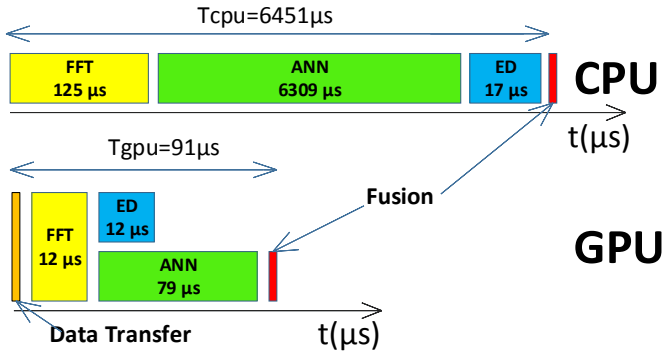

Figure 16. ANN and ED fusion execution time on CPU and GPU

There is an advantage of the GPU for the FFT, ANN and ED functions, due to the use of cores according to the size of the data to be processed. The exploited cores number does not exceed 40% of a single GPU. However, we can explain the observed calculation time increased by the data transferred between GPU and CPU, which for some cases, duration may exceed functions calculation time. Reducing transfer times is possible through direct transfer feature of data between CPU devices and the GPU via the PCI bus. In addition, the Association functions in a multi detector method reducing both the number of transfers and the allocated GPU memory. Finally, an expected benefit for streaming data treatment represents the continuous measurements of the radio signals. In this case, each layer process follows the previous layer data, and then significantly reduces time execution on GPU as shown in Figure 17.

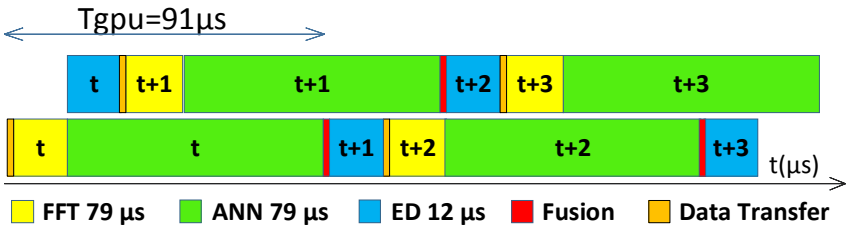

Figure 17. In chain signals data process by GPU

5. Conclusion

In this paper, we demonstrated that the implementation of the FFT, ANN and ED on a GPU computing platform presents a major gain for cognitive radio. With the gains in performance achieved, we could develop multi sensors simulations in order to study the competitive behavior of mobile users. The implementation of alternative free bands detection methods and their fusions will also be an important part to study for their aspects of rapidity and reliability. Another study area will be devoted to the expected performance gains with GPUDirect option, available on Tesla and Kepler GPU generations. This feature allows them directly access to the third peripheral resources connected to the internal bus of the computer or on the LAN (Local Area Network). Furthermore, Nvidia Company put on the market a supercomputer kit for embedded systems, equipped with the Tegra k1 processor and promising architecture for standalone applications of free bands detection. However, issues related to interference and energy consumption will certainly arise for a real intelligent radio detector implementation.

References

Atapattu, S., Tellambura, C., & Jiang, H. (2010). Analysis of area under the ROC curve of energy detection. *Wireless Communications, IEEE Transactions on, 9*(3), 1216-1225. http://dx.doi.org/10.1109/TWC.2010.03.091085

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation, 19*(90), 297-301. http://dx.doi.org/10.1090/S0025-5718-1965-0178586-1

Elrharras, A., Saadane, R., El Aroussi, M., Wahbi, M., & Hamdoun, A. (2014, 14-16 April 2014). *Spectrum sensing with an improved Energy detection.* Paper presented at the Multimedia Computing and Systems

(ICMCS), 2014 International Conference on. http://dx.doi.org/10.1109/ICMCS.2014.6911386

Elrharras, A., Saadane, R., Wahbi, M., & Hamdoun, A. (2014). Signal Detection and Automatic Modulation Classification Based Spectrum Sensing Using PCA-ANN with Real Word Signals. *Applied Mathematical Sciences, 8*(160), 7959-7977. http://dx.doi.org/10.12988/ams.2014.49736

Fuchs, J. J. (2009). *Identification of real sinusoids in noise, the Global Matched Filter approach.* Paper presented at the 15th Ifac-Ifors symposium on Identification and system Parameter Estimation. http://dx.doi.org/10.3182/20090706-3-FR-2004.00187

Hossain, M. S., Abdullah, M. I., & Hossain, M. A. (2012). Energy detection performance of spectrum sensing in cognitive radio. *International Journal of Information Technology and Computer Science (IJITCS), 4*(11), 11. http://dx.doi.org/10.5815/ijitcs.2012.11.02

Kolodzy, P., & Avoidance, I. (2002). *Spectrum policy task force*. (ET Docket No. 02- 135). Washington, DC, Rep: Federal Communications Commission. Retrieved from https://apps.fcc.gov/edocs_public/attachmatch/DOC-228542A1.pdf

Lee, Y., & Koo, I. (2010). A Neural Network-Based Cooperative Spectrum Sensing Scheme for Cognitive Radio Systems. *Advanced Intelligent Computing Theories and Applications, 93*, 364-371. http://dx.doi.org/10.1007/978-3-642-14831-6_49

Liu, W., & Vinter, B. (2014). *An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data.* Paper presented at the Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. http://dx.doi.org/10.1109/IPDPS.2014.47

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics, 5*(4), 115-133. http://dx.doi.org/10.1007/BF02478259

Mitola, J. (1999). *Cognitive radio for flexible mobile multimedia communications.* Paper presented at the Mobile Multimedia Communications, 1999.(MoMuC'99) 1999 IEEE International Workshop on. http://dx.doi.org/10.1109/MOMUC.1999.819467

Ning, H., Sohn, S. H., & Kim, J. M. (2009). A blind OFDM detection and identification method based on cyclostationarity for cognitive radio application. *IEICE transactions on communications, 92*(6), 2235-2238. http://dx.doi.org/10.1587/transcom.E92.B.2235

NVIDIA. (August 2014a). *CUDA RUNTIME API V6.5.* Retrieved from http://docs.nvidia.com/cuda/pdf/CUDA_Runtime_API.pdf

NVIDIA. (August 2014b). *CUFFT LIBRARY USER'S GUIDE DU-06707-001_v6.5.* Retrieved from http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf

Plata, D. M. M., & Reátiga, Á. G. A. (2012). Evaluation of energy detection for spectrum sensing based on the dynamic selection of detection-threshold. *Procedia Engineering, 35*(0), 135-143. http://dx.doi.org/http://dx.doi.org/10.1016/j.proeng.2012.04.174

Renard, J., Verlant-Chenet, J., Dricot, J. M., De Doncker, P., & Horlin, F. (2010). Higher-order cyclostationarity detection for spectrum sensing. *EURASIP Journal on Wireless Communications and Networking, 2010*, 3. http://dx.doi.org/10.1155/2010/721695

Tang, Y. J., Zhang, Q. Y., & Lin, W. (2010). *Artificial neural network based spectrum sensing method for cognitive radio.* Paper presented at the Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on. http://dx.doi.org/10.1109/WICOM.2010.5601105