GPU-based acceleration of ECG characterization using high-order Hermite Polynomials

A. Gil, D. G. Márquez, G. Caffarena, A. Iriarte, A. Otero

Version: accepted article

GPU-based acceleration of ECG characterization using high-order Hermite Polynomials

A. Gil[1], D. G. Márquez[2], G. Caffarena[1] , A. Iriarte[1], A. Otero[1]

[1]Department of Information Technologies, University CEU San Pablo, Spain

[2]Centro Singular de Investigación en Tecnoloxías da Información (CITIUS), University of Santiago de Compostela, Spain

Abstract: In this paper we address the acceleration of the Hermite function characterization of the heartbeat by means of massively parallel Graphics Processors Units. This characterization can be used to develop tools to help the cardiologist to study and diagnose heart disease. However, obtaining this characterization, especially when a large number of functions is used to achieve a high accuracy in heartbeat representation, is very resource intensive. This paper addresses off-line and on-line heartbeat characterization, assessing the acceleration capabilities of Graphics Processors Units for these tasks. Polynomials up to the 30th order are used in the study. The results yield that the off-line processing of long electrocardiogram recordings with a GPU can be computed up to 186× faster than a standard CPU, while real-time processing can be up to 110× faster.

INTRODUCTION

Cardiovascular disease is the leading cause of death by natural causes in the world (WHO 2010)[1]. The electrocardiogram (ECG), a recording of the electrical activity of the heart, is a simple and inexpensive diagnostic tool that provides a great deal of information about the physiological processes that concur in the alterations of the electrical impulse in the myocardium. The use of the ECG as a diagnostic tool is increasing significantly worldwide. Its use has been common in hospitals from developed countries for decades, but now it is spreading fast in large emerging countries such as China and India (Reddy C 1998)[2](Gaziano CPC 2010)[3]. The increase in the quality of life that the inhabitants of these countries are enjoying, and the adoption of Western habits, bring an associated significant increase in cardiovascular disease. Furthermore, the quantified self movement, which promotes the idea of spreading ECG use beyond the clinic so it can be part of everyday life (Swan SAT 2012)[4] (Min MNA 2011)[5], is enabling the application of ECG technology not only to study pathologies which have developed to the point of presenting symptoms, but also in the early identification of diseases that are not (yet) symptomatic.

In the clinical routine, ECG analysis is still performed by visual inspection of an expert cardiologist. This is a very tedious task, especially in the case of the long term ECG, such as Holter recordings, which usually last form 24 to 72 hours and can be comprised of up to 12 leads. Each of these leads contains about 100,000 beats per 24 hours of monitoring. An additional disadvantage of the visual inspection of ECG recordings is that its outcome relies heavily on the cardiologist who performs it (Willems NEJM 1991)[6].

In this context, it is undoubted the interest in developing efficient computational methods of ECG analysis capable of processing massive amounts of data in an objective way. Among these methods, those located at an early stage of identification and morphological detection of the beat stand out for their importance. The information they provide is used for the characterization of the cardiac cycle on which the interpretation of the underlying disease process is subsequently performed. Errors that occur in this phase usually invalidate the outcome of the interpretation. The identification of the beat is a problem solved with acceptable accuracy in the literature (around 99.7% in reference databases). The morphological beat identification and classification by its origin in the myocardium, more commonly known as arrhythmia detection, is still an open problem (de Chazal ITB 2004)[7], particularly in the analysis of long term recordings (Kiranyaz ESA 2011)[8].

In the literature, different approaches for the detection and classification of arrhythmias can be found. One approach is to segment the waves that make up the beat (P, Q, R, S and T waves) and characterize these waves by parameters such as height, width, or their relative temporal distance (Hamilton ITB 1986)[9]. While this approach has the advantage of solving the task using

the same features that the cardiologist uses, the noise in the ECG, especially in extra-ambulatory ECG such as Holter recordings, makes it difficult to obtain a robust and reliable segmentation of the waves. Other authors use the sampled ECG data directly (Sörnmo 2005)[10]. This often leads to a very high dimensional feature space, hindering the machine learning techniques required during the subsequent arrhythmia identification and classification stages.

An increasingly popular solution is using a function space. The ECG samples are fitted with a linear combination of basis functions, and the coefficients of this linear combination are used as features for representing the beat. Although this was originally carried out with Laguerre orthogonal polynomials (Young ITBE 1963)[11], Hermite polynomials have lately become the most popular representation (Osowski ITBE 2004)[12] (Haraldsson AIM 2004)[13] (de Chazal ITB 2004)[7]. Firstly, the shape of these polynomials resembles the shape of the beats, which permits obtaining relatively accurate representations using a small number of polynomials (see Fig. (1)). Secondly, this representation has also proven robust in the presence of noise. And finally, the fact that Hermite polynomials make an ortho-normal basis implies that the information provided by different coefficients is not redundant. These advantages have made the Hermite representation very popular in the literature. A query in Google Scholar with the terms "hermite representation electrocardiogram" returns over 2100 scientific papers, of which over 900 have been published since 2010.

One disadvantage of the Hermite representation is that it is computationally demanding. This is especially true when polynomials with high degrees are used to obtain a more precise representation of the beat, thus enabling a better classification of arrhythmias in a subsequent stage. Parallel computing systems are used to solve complex scientific problems that require the processing of extremely large amounts of data, since they enable producing results in bounded computation times. Among the different available approaches, the use of Graphics Processing Units (GPU) has become a very attractive option for the acceleration of time-consuming computing techniques. These devices are multi-threaded systems composed of several hundreds of processors with an optimized memory hierarchy. They are relatively cheap, easy to install and, in many cases, they are as powerful in terms of computation as hundreds of microprocessors working in parallel, while providing a low power consumption. A single commodity PC with one or several GPU devices attached is able to replace a whole cluster with tens and even hundreds of computers. Their application to the biomedical field is patent: MRI reconstruction (Kirk 2010)[14] (Nicholls IM 2010)[15], cardiac tissue simulation (Garcia-Molla CBM 2014)[16], biomolecular dynamics (Zhang IWB 2013)[17] (Nicholls IM 2010)[15], bioinformatics (Nicholls IM 2010)[15], etc.

In this paper, we assess the suitability of GPU parallelization for the fast characterization of ECG recordings by means of Hermite functions. The MIT-BIH arrhythmia database (Moody IEM 2001)[18] is used as a benchmark, and two different scenarios are selected: off-line processing and on-line processing. The former aims at the processing of long recordings such as the ones obtained by a Holter device. The latter aims at real-time processing of the ECG. In (Márquez BIO 2013)[19] it was proved that 6 Hermite functions led to good enough results in terms of the Mean Squared Error (MSE) with respect to the original heartbeat. In (Gil IWB 2014)[20] preliminary results on the GPU-parallelization of the Hermite characterization for orders up to 9 were presented. However, the findings from (Márquez BIO 2013)[19] do not relate MSE with arrhythmia classification performance. The impact of Hermite polynomial order on arrhythmia classification is studied in (Márquez BSPC 2015)[21] concluding that to achieve optimal results it is necessary to reach an order in the range of 28-30.

In this work the fast implementation of Hermite heartbeat characterization for both off-line and on-line processing, covering high-order polynomials, is presented. We extend and improve the GPU implementations and results from (Gil IWB 2014)[20]. First, Hermite function calculation is optimized in order to reduce computation time, and also to increase mathematical precision so that high-order polynomials can be considered. The range of polynomial order is extended up to 30 to increase the accuracy of the representation (Márquez BSPC 2015)[21]. And finally, an in-depth timing analysis is provided to shed some light on the way that parallelization is handled by the GPU device.

The paper is organized as follows: Section 2 introduces the characterization of beats based on Hermite functions. Section 3 briefly introduces the GPU programming model. Section 4 explains the parallelization of the characterization algorithms for the off-line and on-line scenarios. The results are presented in Section 5 and, finally, the conclusions are drawn in Section 6.

QRS APPROXIMATION BASED ON HERMITE FUNCTIONS

The aim of using Hermite functions to represent beats is to obtain an accurate representation of the beat that is robust in the presence of noise and that permits addressing later stages of the ECG analysis, such as the classification of arrhythmias, with reasonable guarantees of success. The benchmarks used in this work are extracted from the MIT-BIH arrhythmia database (Moody IEM 2001)[18], conforming a total of 48 ECG recordings whose beats have been manually

annotated by at least two cardiologists. Each file from the database contains 2 ECG channels, sampled at a frequency of 360 Hz and with duration of approximately 2000 beats (half an hour).

The ECG data is processed in order to extract information about the morphology of the beat (i.e. the QRS complex), as well as information about the distance between each pair of consecutive beats (i.e., the RR intervals). The QRS of the beats permits the identification of most arrhythmia types. The T wave provides very little additional information for arrhythmia detection once the QRS morphology has been considered (Sörnmo 2005)[10]. The P wave does provide relevant information for certain types of arrhythmias. However, its signal-to-noise ratio is very low, making it impossible to identify it reliably, especially in ambulatory ECG such as Holter recordings (Swan SAT 2012)[4]. Hence the preferred solution in the literature is trying to extract similar information to the one provided by the P wave from features derived from the distance between consecutive beats (de Chazal ITB 2004)[7] (Lagerholm ITB 2000)[22].

Before performing the Hermite characterization, the ECG recordings are filtered to remove baseline drift and high frequency noise. Then, for each beat, a 200-ms window centered on the beat annotation is extracted. This window size permits the extraction of the complete QRS complex of normal beats, which have a length of 70-100-ms, and leaves out the P and T waves. The Hermite basis functions always converge to zero in $\pm\infty$. To ensure the convergence to zero at the edges of the signal window, the 200-ms window extracted from the ECG is extended to 400-ms by adding two 100-ms sequences of zeros at each side of the window. Eventually, the extracted 400-ms QRS data is stored in a 144-sample vector $x=\{x(t)\}$, and this vector can be approximated through a linear combination of N Hermite basis functions $\phi_n(t,\sigma)$ using the coefficients $c_n(\sigma)$, as follows

$$\hat{x}(t)=\sum_{n=0}^{N-1} c_n(\sigma)\phi_n(t,\sigma) \qquad (1)$$

The Hermite functions are defined as:

$$\phi_n(t,\sigma)=\frac{1}{\sqrt{\sigma 2^n n!\sqrt{\pi}}}e^{-t^2/2\sigma^2}H_n(t/\sigma) \qquad (2)$$

$H_n(t/\sigma)$ are the Hermite polynomials.

These polynomials can be computed recursively according to:

$$H_0(x)=1$$
$$H_1(x)=2x$$
$$H_n(x)=2xH_{n-1}(x)-2(n-1)H_{n-2}(x).$$

Note, that the parameter σ varies the width of the polynomials. The maximum value of this parameter, σ_{MAX}, for a given order n is computed in (Lagerholm ITB 2000)[22] and it holds that the bigger n is, the smaller σ_{MAX} becomes.

Given the orthogonality between the basis functions, it is possible to find the optimal coefficients that minimize the mean squared error (MSE) of the estimation for a given σ.

$$c_n(\sigma)=\sum_t x(t)\phi_n(t,\sigma) \qquad (3)$$

Thus, for each beat, the combination of σ and vector $\boldsymbol{c}=\{c_n(\sigma)\}$ ($n\in[0,N-1]$) that minimizes the MSE can be found and the values of these $N+1$ parameters can be used to reconstruct the QRS complex. The MSE is defined as in eqn. (4)

$$MSE=\sum\left(x_i(t)-\hat{x}_i(t)\right)^2 \qquad (4)$$

Fig. (1) shows how the Hermite polynomials model the heartbeat shape and how the accuracy of the representation can be improved by increasing the number of polynomials used. To quantify the improvement in accuracy, we have calculated the MSE (eqn. (4)) for all the beats of the MIT-BIH arrhythmia database. Table (1) shows the mean MSE values and their standard deviation when the beat is represented using 6, 10, 20 and 30 polynomials. It can be appreciated how when the degree of the polynomial increases the representation error decreases.

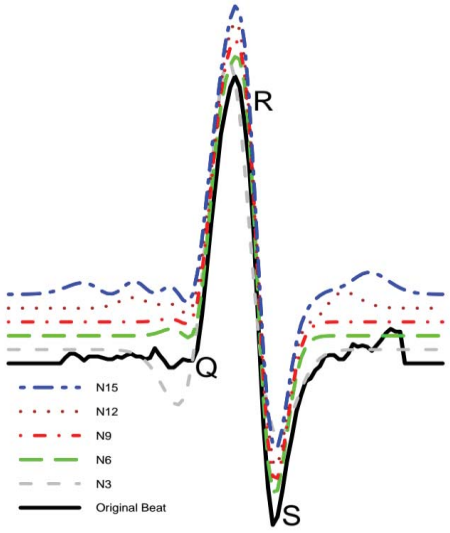


Fig. (1). Hermite characterization of a beat with 3, 6, 9, 12 and 15 polynomials.

GPU-BASED ACCELERATION

GPU devices are composed of hundreds of processor cores that work in parallel, executing the same task (i.e. kernel). Thus, GPUs enable the massive parallelization of algorithms, reaching speedups ranging from 10× to 300× (Nicholls IM 2010)[15] while keeping a low-power consumption (Brodtkorb ATD 2010)[23] in comparison with supercomputers or computer clusters.

Current devices are connected to the PC by means of a PCIe connection, thus, opening the door to high-performance computing to low-cost systems. GPUs have been welcomed by the scientific community due to their low cost, their arguably programming simplicity and their suitability for floating-point computations, which is the standard adopted in scientific computation. In the last two decades, they have been applied to a wide range of disciplines, including many biomedical research projects (Kirk 2010)[14] (Nicholls IM 2010)[15] (Garcia-Molla CBM 2014)[16] (Zhang IWB 2013)[17] (Gil IWB 2014)[20].

N	MSE
6	9.79 $\pm$ 10.0
10	5.32 $\pm$ 6.97
20	2.48 $\pm$ 4.92
30	1.97 $\pm$ 3.95

Table (1). Average mean squared error when representing beats with 6, 10, 20 and 30 polynomials.

The GPU is designed for executing the same task using a huge volume of data. Moving apart from this scenario (high data dependency, conditional flows, etc.), results in non-perceptible performance gains. C-like programming languages, such as CUDA (Compute Unified Device Architecture) (Kirk 2010)[14], can be used to program GPUs. These languages enable parallel programming and they provide fast compilation and an easy integration with traditional programs executed by the CPU (i.e. host side). It must be highlighted that CUDA was selected as the target programming language for this project, since it is tuned for the GPU devices used for this research (Nvidia GPUs).

The programming model of CUDA is intended for encapsulating the inner hardware details of the GPU to the programmer, in order to ease the development process, as well as to facilitate portability to different GPU devices (from Nvidia). The GPU holds several streaming processors (SP) which are formed of several cores that can work in parallel. The GPU executes the same piece of code (kernel) in parallel using different data sets. A thread is a particular execution of the kernel. Each SP handles in parallel a set of threads grouped together in the so-called warps. The execution of the threads in a warp is parallel as long as there are no conditional branches. If there are different execution paths, the SP executes in parallel all threads that point at the same instruction. Thus, each SP falls into the Single-Instruction Multiple-Data paradigm (SIMD), which, in this context, is renamed as Single-Threat Multiple-Data (STMD) due to the link between instructions and threads.

The SP must first cluster all threads in a warp that are in the same execution point, and then, it executes sequentially each cluster. Thus, the presence of conditional branches can deteriorate performance considerably.

The programmer has some control on the way that threads work in parallel by means of bearing in mind the STMD subtleties while coding, and by grouping the threads. Threads can be grouped in blocks with a 1D, 2D, or 3D configuration, and a big number of blocks can be instantiated. Thus, each thread has a 3-dimension identifier (ID). During scheduling, each block is assigned to an SP, and the SP starts the execution of all of its threads (by means of warps). In a similar fashion, blocks are distributed in a 1D/2D mesh, called a grid. Thus, the block also has an identifier, and this identifier is visible to its threads. Each thread can use the block's and thread's IDs to generate the memory locations of the data sets that it has to access.

Regarding memory, all threads can access global memory (DRAM), all threads within a block access shared memory (SRAM), and each individual thread accesses a set of local registers. The key point here is that global memory has a high capacity (i.e. 1-6 GB) but it is slow, while shared memory has a small capacity (i.e. 16-48 KB) but it is fast (a couple of orders of magnitude faster than global memory). Global memory must be accessed in a coalesced fashion, since the read and write operations work with several consecutive bytes (32, 64, 128, etc.). A 2-level cache system is present and, as expected, it optimizes DRAM usage, although it is recommended to do a thorough planning of the way that data is going to be accessed to avoid an inefficient use of global memory leading to prohibitive delays. Shared memory can be accessed randomly, leading to very fast data access if some constraints are considered.

Considering all these remarks, an algorithm that is suitable for parallelization can be efficiently executed on a GPU if, a wise selection of the block and grid shapes and sizes is carried out, and memory access is carefully devised.

BASELINE IMPLEMENTATION

In this section, the algorithm used to characterize the QRS complex is presented. This algorithm is implemented on a PC and it is used as the baseline to compute the speedup obtained by a CUDA-based implementation.

Algorithm (1) shows the computations involved in the characterization of the ECG beats. The inputs to the algorithm are the ECG data and the maximum polynomial order N to be used in the beat characterization. The output is the set of N+1 parameters used to characterize each of the QRS. The algorithm shows three main parts: i) the extraction of

the QRS complexes; ii) the generation of the Hermite functions (labeled as *Loop1*); and, iii) the Hermite characterization of the complexes (*Loop2*).
The QRS complexes are extracted from the ECG recording, outputting a 144-sample signal $x_i(t)$ for each beat as explained in the second section of the paper.

Input: ECG data, maximum polynomial order N
Output: Best set of parameters for each beat
$(\sigma, \mathbf{c}(\sigma))$

1: *Extract QRS complexes from the ECG file (*$x_i(t)$*)*
2: *# Loop 1*
3: *for all* σ *and* n *do*
4: *Compute* $\phi_n(t,\sigma)$ *# eqn. (2)*
5: *end for*
6: *# Loop 2*
7: $err_{min} = \infty$
8: *for all* $x_i(t)$ *do*
9: *for all* σ *do*
10: *for all* n *do*
11: *Compute* $c_n(\sigma)$ *# eqn. (3)*
12: *end for*
13: *Compute* $\hat{x}_i(t)$ *# eqn. (1)*
14: *Compute* $err = MSE(x_i(t), \hat{x}_i(t))$ *# eqn. (4)*
15: *if* $err_{min} > err$ *then*
16: $\sigma_{BEST} = \sigma$
$\mathbf{c}_{BEST} = \{c_n(\sigma)\}$
$err_{min} = err$
17: *end if*
18: *end for*
19: *end for*

Algorithm (1). QRS characterization

In the first loop (*Loop1*, lines 2-5), the values of the $\phi_n(t,\sigma)$ functions are precomputed according to eqn. (2). This precomputation is performed since the Hermite functions are used repeatedly during the second loop. The benefit of this precomputation was tested by comparing the computation time of *Loop2* under two scenarios: computing the Hermite functions inside the loop or using the precomputed values. The achieved speedup for the second option was 105× using an Intel i7 microprocessor.
As for the second loop (*Loop2*, lines 6-19), it is devoted to finding the optimal set of parameters (σ and coefficient vector $\mathbf{c}$) for each beat. It is composed of two nested loops: the outer one traverses all x_i and the inner one looks for the optimal coefficients (eqn. (3)) for different values of σ.
A total of S values of sigma are tried from the set $\sigma = \{\sigma_0 \dots \sigma_{MAX}\}$. The value of σ_{MAX} is a function of N (Lagerholm ITB 2000)[22] and S is around 100. Thus, for each beat and for different values of σ, the optimal coefficients ($\mathbf{c}$) are found and the combination of σ and $\mathbf{c}$ that minimizes the MSE (eqn. (4)) between estimation $\hat{x}_i$ and the actual QRS complex x_i is selected to characterize the beat.

An analysis of Algorithm (1) draws that both *Loop1* and *Loop2* are suitable for parallelization.

PARALLEL IMPLEMENTATION

The parallelization approach taken is shown in Algorithm (2) where loops *Loop1* and *Loop2* from Algorithm (1) were parallelized by means of CUDA kernels *kernel_*ϕ and *kernel_Hermite*.
Algorithm (2) shows that the parallel version starts by extracting the QRS complexes x_i (line 1) and by allocating memory in the GPU device (line 2). After that, the Hermite functions are generated (*kernel_*ϕ) and stored in the GPU global memory (line 3). While this kernel is being executed, the beats are sent to the GPU and stored in the GPU global memory (line 3). Once the first kernel has finished (line 4), the next kernel (*kernel_Hermite*) has access to all the required input data for the Hermite characterization in the GPU global memory. The characterization is then performed (line 5) and the results are sent from the GPU to the host memory (line 7). Before sending the results, it is necessary to synchronize with the GPU execution (line 6) to avoid reading inconsistent data.
Following, these kernels are explained as well as the way to optimize the data transfers for real-time processing, and for the processing of very long ECG recordings.

Precomputation of Hermite functions

The parallelization of *Loop1* in algorithm (1) is straightforward. The Hermite functions $\phi_n(t,\sigma)$ are composed of 144 samples with disregard of the values of n and σ. Arranging blocks in an $S \times N$ grid, and using 144 threads per block, leads to full parallelization. The idea is that each block deals with the computation of all the samples of a function ϕ for a concrete (n, σ) couple. The 2-dimensional block ID has as a first component the index of σ, and as a second one the maximum weight of the Hermite polynomials, n. Each thread within a block is devoted to the evaluation of eqn. (2) at a different time step t, which is directly the thread ID. This scheme results in the simultaneous computation of as many ϕ functions as SPs are in the GPU.
Fig. (2) displays the distribution of threads and blocks.

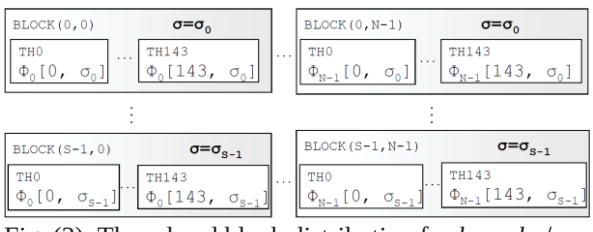


Fig. (2). Thread and block distribution for *kernel_*ϕ

Hermite characterization

The core of the Hermite characterization is carried out in *Loop2* (Algorithm (1)) and its parallelization requires a thorough analysis in comparison to *Loop1*. Each block will handle a different QRS complex $x_i(t)$, so there are as many blocks as beats in the ECG recording (i.e. approximately 2000 for MIT-BIH files). Each block holds 144 threads, since most of the time all the threads are able to work in parallel. A thread can use the block ID to select the beat to work on and the thread ID to know the index of the sample of the beat that is using for the computations.

Algorithm (3) shows the pseudocode for *kernel_Hermite*. It must be borne in mind that the kernel is executed by all the threads in a block. In lines 1-2, the block's and thread's IDs are used to obtain both the number of the beat (i) and the sample associated to the thread (t). Then, the beat data is copied onto shared memory (line 3) since it is going to be used many times during the kernel execution, so from now on, any reference to $x_i(t)$ implies a fast reading from shared memory. As in Algorithm 1, a loop traversing all values of σ is included (lines 5-20). The vector of coefficients $\boldsymbol{c}$ is computed for each value of σ (lines 6-11). First, the multiplications between the original signal samples ($x_i(t)$) and the Hermite functions ($\phi_n(t,\sigma)$) are computed in parallel, so that each thread performs a multiplication on its own (lines 6-8). Then, a reduction technique is applied to carry out the summation (Kirk 2010)[14] (lines 9-11) to avoid a fully sequential execution. The computation of the MSE requires having available the estimation of the beat for the current coefficients. Lines 12-14 show how each thread iterates through the different polynomial orders, computing the summation of the multiplication of the coefficients by the original samples of the QRS complex for each sample of the estimation in parallel. The MSE is computed then in two steps. The squared error between $x_i(t)$ and $\hat{x}_i(t)$ is computed in parallel and then, the summation is performed by reduction (lines 15-16). Finally, thread 0 updates the best solution if the new MSE computed is the minimum so far (lines 17-19).

Input: ECG data, polynomial order N
Output: Best set of parameters for each beat (σ, $\mathbf{c}(\sigma)$)

1: Allocate GPU memory
2: Extract QRS complexes from the ECG file ($x_i(t)$*)*
3: Call kernel_ϕ
  Send all $x_i(t)$ *to GPU #Write onto GPU Global memory*
4: Wait for GPU to finish processing
5: Call kernel_Hermite
6: Wait for GPU to finish processing
7: Read $\{(\sigma_i, \mathbf{c}_i(\sigma))\}$
  # (Write onto Host memory

Algorithm (2). Host-side code

Input: ECG data, polynomial order *N*
Output: Best set of parameters for each beat (σ, $\mathbf{c}(\sigma)$)

1: i = *block.ID # beat index*
2: t = *thread.ID # sample index*
3: Copy $x_i(t)$ *to shared memory* *# fully parallel*
4: err = ∞
5: for all σ *do*
6: *for all* n *do*
7: *Compute* $sum_t = x_i(t)\phi_n(t,\sigma)$ *# eqn. (3)*
 # fully parallel
8: *end for*
9: *for all* n *do*
10: *Compute* $c_n(\sigma) = \sum_t sum_t$ *# eqn. (3)*
 # reduction technique
11: end for
12: $\hat{x}_i(t) = 0$
12: *for all* n *do*
13: *Compute* $\hat{x}_i(t) += c_n(\sigma)\phi_n(t,\sigma)$ *# eqn. (1)*
 # fully parallel
14: *end for*
15: Compute $err_{tmp} += (x_i(t) - \hat{x}_i(t))^2$ *# eqn. (4)*
 # fully parallel
16: *Compute* $MSE = \sum_t err_{tmp}$ *# eqn. (4)*
 # reduction technique
17: # This code is only for thread 0
if t = 0 *and* err > MSE *then {*
18: $\sigma_{BEST} = \sigma$
$\mathbf{c}_{BEST} = \{c_n(\sigma)\}$
err = MSE
19: } end
20: end for

Algorithm (3). Pseudocode for *kernel_Hermite*

Data transfer optimization

The dataflow presented in Algorithm (2) can be modified and optimized in order to take into account the capabilities of GPUs to perform the simultaneous execution of a kernel and a data transfer (e.g. host to GPU, or GPU to host). This is of special interest, since we are dealing with the analysis of very long ECG records (i.e. Holter device) with an amount of data that will not fit the GPU memory, thus it will be necessary to process the data in subsets. Also, if real-time processing is targeted, it is mandatory that data is processed in small subsets to avoid long delays in the presentation of the results. Fig. (3) shows how it is possible to maximize performance by overlapping data transfer with GPU computation. The first task is the execution of *kernel_ϕ* and then, the first subset of beats (subset 0) is sent to the GPU. While *kernel_Hermite* is characterizing subset 0, subset 1 is being transferred. After the execution of this first call to *kernel_Hermite* the results can be transferred to the host memory. During the second call to *kernel_Hermite*, subset 1 is being characterized and subset 2 is being transferred to

the GPU, and again, the results corresponding to subset 1 are transferred to the host when the kernel computation is over. The process continues for the rest of subsets. Thus, it is possible to compute in a pipeline fashion (i.e. streaming processing) and performance is optimized. Note that the durations of the tasks (data transfer and kernel execution) are not the real ones. They have been set to easily show the behavior of the streaming processing. For instance, the execution time of *kernel_Hermite* is actually longer than the data transfer to the GPU, but the ratio between the displayed durations is not the real one.

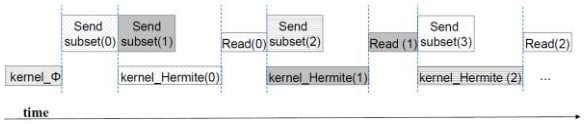


Fig. (3). Optimization of execution time via streaming processing

RESULTS

In this section we present the comparison between several parallel implementations based on CUDA and a single-thread reference programmed using C language. The baseline implementation was coded following Algorithm (1), while the parallel implementations considered Algorithms (2) and (3) and the optimizations for streaming processing. The test platform was a PC with an Intel-i7 (1,6 GHz and 4 GB of RAM) and a graphics processor Nvidia TESLA C2050 (448 cores, 4 GB of RAM). Three different tests were performed

A. Off-line processing of short recordings.
It intends to assess the processing of short ECG recordings. The number of beats per channel used for the test was $M \in \{10, 10^2, 10^3, M_{\max}(N)\}$, where $M_{max}(N)$ is the maximum number allowed by the GPU for a given maximum polynomial order N, which is close to 5000 beats. Also, the range of polynomial order was $N \in \{6,10,20,30\}$. A total of 16 experiments were performed.

B. Off-line processing, long recordings.
It intends to simulate the off-line processing of long ECG recordings, such as Holter recordings. A total number of beats per channel of around 10^5 was used. The data was processed in chunks close to 5000 beats, always trying to process the maximum number of beats that the GPU can handle for a given value of N. The range of polynomial order was {6,10,20,30}. A total of 4 experiments were performed.

C. On-line processing.
It intends to simulate the online processing of ECG recordings, such as the real-time processing of the ECG of a patient admitted to a critical care unit. It uses streaming with blocks of short duration (i.e. 1, 5, 10 and 100 beats per channel). The number of blocks for each experiment was selected in such a way that the total number of beats per channel was 10^5. The range of polynomial order was {6,10,20,30}. A total of 16 experiments were performed.

N	M	CPU (ms)	GPU (ms)	Speedup
6	10	26	66	0.39×
	100	173	65	2.65×
	1000	1,637	79	20.66×
	5000	10,626	128	83.02×
10	10	59	67	0.87×
	100	282	71	3.95×
	1000	2,569	82	31.32×
	4800	15,534	155	100.15×
20	10	177	75	2.38×
	100	604	80	7.52×
	1000	4,868	122	39.86×
	4600	27,287	288	94.90×
30	10	372	75	4.93×
	100	995	89	11.24×
	1000	7,216	180	39.99×
	4400	37,454	521.12	71.87×

Table (2). Performance results for Test A

Test A: Off-line, short recordings

Table (2) shows the computation time and speedup for Test A. The first column indicates the number of Hermite polynomials used and the second column the number of beats per channel processed, *M*. The third and fourth columns hold the computation time in ms of the baseline (CPU) and the parallel implementation (GPU), respectively. The last column shows the speedup. Note that the number of beats (second column) is constrained by the maximum data size that the GPU can handle due to memory restrictions.

Figure (4) displays the computation time for both the CPU and GPU implementations, and might help the reader to get an overall idea of the impact of the polynomial order *N* and the number of beats processed on the performance.

The results yield that the performance of the GPU improves as long as the maximum order of the Hermite polynomials increases, the number of beats increases, or both simultaneously. Note that for very short recordings, the performance of the GPU could be even worse than that of the CPU ($N \in \{6,9\}$ and 10 beats). The rest of the combinations present acceleration, although they are not significant for recording with less than 1000 beats. The maximum speedups happen for a number of beats around 5000 and they are in the range [72×, 100×].

We have performed a detailed analysis of the distribution of the computation time devoted to the different tasks performed in the parallel implementation (see Algorithm (2)). Fig. (5) displays this distribution, showing the percentage of the total time due to:

1. The memory allocation of variables in the PC memory (Malloc CPU)
2. The memory allocation of variables in the GPU memory (Malloc GPU)
3. The execution of *kernel_ϕ*
4. The transfer of the beats to the GPU (First transfer)
5. The execution of *kernel_Hermite*
6. The transfer of the results to the host (Read)

Fig. (5).a. shows the distribution for 6 Hermite polynomials according to different recording sizes. The black areas correspond to "*Malloc GPU"* and the light grey areas to "*kernel_Hermite"*. This shows that for small numbers of beats the time it takes to set the GPU up before processing data is longer that the GPU processing time. Since the baseline requires a much faster setup before starting processing, the GPU does not provide any advantage. When the number of beats increases the GPU processing time takes over the memory allocation time and, since the CPU requires notably more time to compute the Hermite coefficients, a significant gain is obtained from using a GPU.

Fig. (5).b. holds similar results for 30 polynomials. In this case the GPU computation time percentage dominates for long ECG recordings, hence the high speedups obtained. Note also that in this case data transfer to the GPU is not negligible for short recordings.

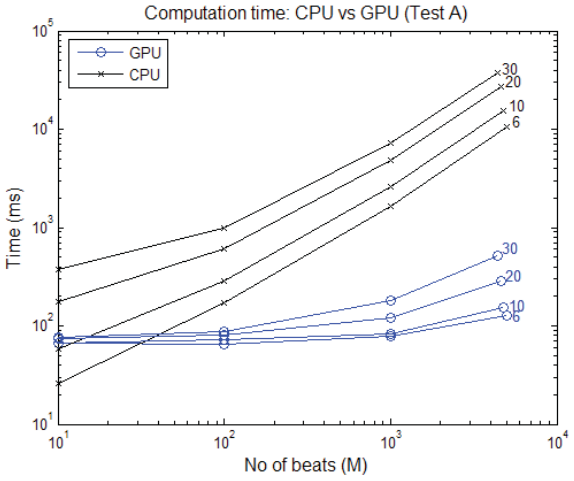
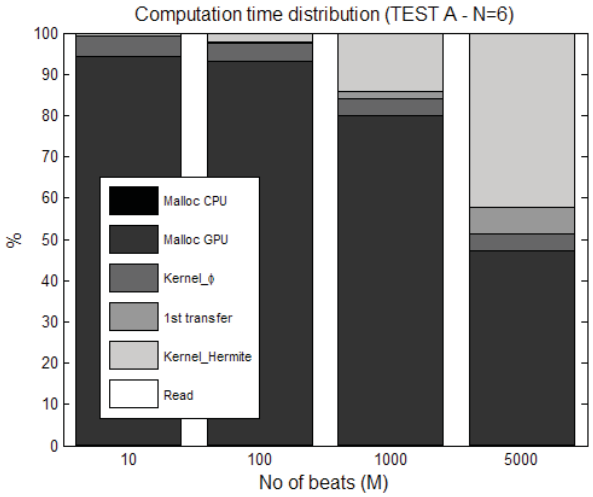
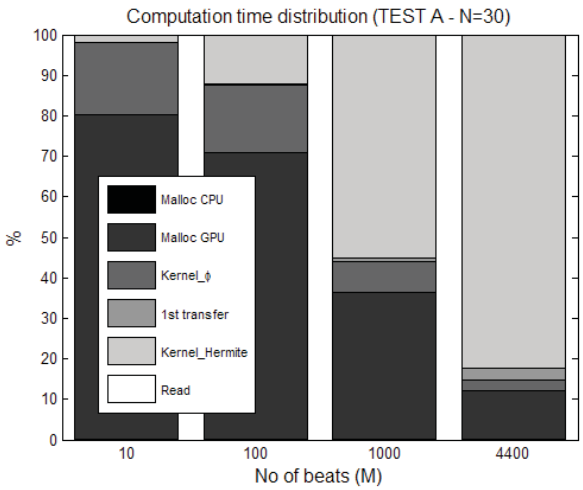


Fig. (4). Execution time of CPU and GPU implementations of **Test A** for different polynomial orders.



a)



b)

Fig. (5) Computation time distribution of the different tasks involved in the GPU implementation of **Test A** for: a) 6 Hermite polynomials, and, b) 30 polynomials.

It must be highlighted that the performance does not increase monotonically. This is due to the fact that at some point, for values of $N \geq 20$, the number of local variables in the kernel exceeds the number of registers in the SP. Therefore, it is necessary to resort to global memory, hindering acceleration – the so-called *register spilling*.

Test A does not correspond to a realistic real world scenario, but it provides useful insights on the behavior of the GPU. The information related to long recordings (M≈5000) will be used in Test B to optimize the processing of long recordings. Likewise, the analysis of the processing of short recordings will provide clues to implement Test C.

Test B: Off-line, long recordings

Table (3) shows the computation time and speedup for Test B. The first column indicates the number of Hermite polynomials used. The second column contains the number of blocks processed (M_1) and the number of beats per block (M_2). The total number of beats processed ($M=M_1M_2$) is approximately 10^5. The third and fourth columns hold the computation time in ms of the baseline (CPU) and the parallel implementation (GPU), respectively. The last column shows the speedup.

N	M_1 M_2	CPU (ms)	GPU (ms)	Speedup
6	20 5000	211,559	1,230	172×
10	21 4762	322,829	1,735	186×
20	22 4546	599,249	4,612	130×
30	23 4348	845,513	9,882	86×

Table (3). Performance results for Test B

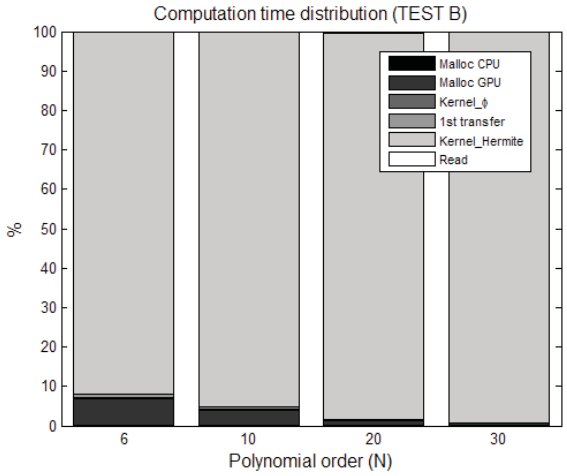


Fig. (6) Computation time distribution of the different tasks involved in the GPU implementation of **Test B** for $N \in \{6,10,20,30\}$.

The results show that the performance of the GPU is significantly boosted with respect to Test A. This is due to the fact that the execution time of *kernel_Hermite* now dominates more than 90% of the total computation time, as it can be seen in Fig. (6), where the time distribution for different polynomial orders is displayed. Again, for N=30 there is a lower speedup, but still significant, due to GPU register spilling. Note that the time required to transfer the beats to the GPU is not considered in the graph, since it happens simultaneously with the computation of *kernel_Hermite* and it requires a shorter time.

The speedups range from 86× to 172×, thus, proving the benefits of using a GPU for long recording processing. The processing time has been reduced from minutes (CPU) to seconds (GPU), leaving plenty of room to include complex classification techniques for the detection of arrhythmia.

We can extrapolate the results to a real world scenario where a Holter device with 12 channels is used to monitor a patient during three days. In this case, the time that a CPU requires to characterize the beats using 30th order polynomials is estimated as more than 3 hours and a half, while a GPU would require approximately 2 minutes a half.

Test C: On-line processing

Table (4) holds the computation time and speedup for Test C. The first column indicates the number of Hermite polynomials used. The second column contains the number of beats per block (M_2). The higher M_2, the longer the latency. Assuming a heart frequency of 1 Hz, the processing latency is equal to M_2 in seconds. The value of M_1 is set so the total number of beats is $M=M_1M_2=10^5$. The third and fourth columns hold the computation time in ms of the baseline (CPU) and the parallel implementation (GPU), respectively. The computation time of the CPU is constant for a given value of N since it does not depend on M_2. The last column shows the speedup.

N	M_2	CPU (ms)	GPU (ms)	Speedup
6	1	160,652	46,882	3.43×
	5		9,315	17.25×
	10		5,404	29.73×
	100		1,494	107.53×
10	1	250,691	61,864	4.05×
	5		12,815	19.56×
	10		6,613	37.90×
	100		2,267	110.56×
20	1	472,305	101,674	4.65×
	5		20,626	22.90×
	10		10,402	45.40×
	100		4,965	95.12×
30	1	689,779	157,276	4.39×
	5		29,978	23.01×
	10		15,653	44.07×
	100		11,359	60.72×

Table (4). Performance results for Test C

The results show that performance gains are obtained even when results are provided in a single beat basis. In fact, the GPU always outperforms the CPU implementation (see Fig. (7)). However, the speedups obtained for $M_2=1$ are not high enough to justify the cost of a GPU, since a multithreaded implementation

using a standard CPU could match the GPU performance. Moreover, the CPU is able to process in real-time since the execution time in column 2 (Table (4)) is smaller than the time it takes for the heart to beat M_2 times. The benefit of using a GPU for real-time processing comes from the fact that being faster than the CPU it saves time to apply more powerful on-line classification algorithms.

The speedups range from approximately 4× to 110×. This time, an increase in N or M_2, always leads to an improvement in the speedup, since the sizes of M_2 are small enough to avoid register spilling. Fig. (8) shows the time distribution for Test C when the polynomial order is 6. Most of the time now is devoted to the execution of *kernel_Hermite*. The distributions for higher values of N are similar.

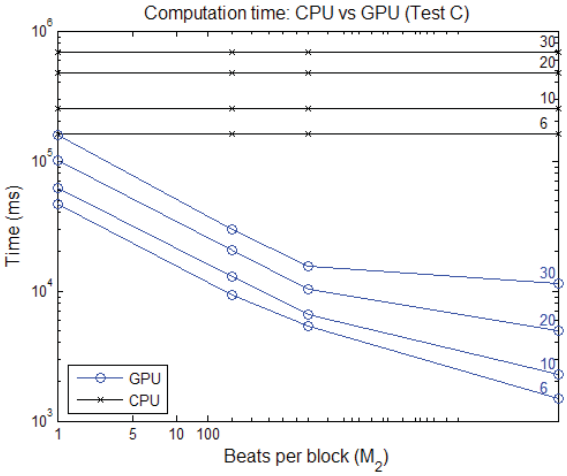


Fig. (7). Execution time of CPU and GPU implementations of **Test C** for different polynomial orders.

There is a clear tradeoff between latency and performance gain, since, the higher the performance, the higher the latency. For instance, it seems reasonable to wait 10 seconds ($M_2=10$) to obtain the results, and the speedup in this scenario is significantly faster than a multithreaded CPU-based implementation (around 20×). However, it is not so clear that waiting 100 seconds ($M_2=100$) to get the results complies with the definition of real-time, regardless of the performance gain (around 100× for some cases).

In summary, these results show the interest of using GPUs to perform real-time beat characterization, provided that we can group them in sets of at least 5.

Regarding the previous implementations from (Gil IWB 2014)[20], the implementations used in this work take them as a starting point, though some major improvements have been applied. The idea of precomputing the Hermite functions (eqn. (2)) to avoid redundancy in the computation process, was also applied to the implementation of eqn. (2) itself. The continuous computation of common mathematical operations, such as the factorial function, was optimized. This led to improving the results of Test A, so now both the CPU and GPU implementations run faster and, still, there is a significant gain in the obtained speedups, which are doubled with respect to the former results. Also, eqn. (2) was reformulated to avoid numerical issues. The straightforward implementation in (Gil IWB 2014)[20] could not reach polynomial orders beyond 9 due to numerical problems. A numerical-stable implementation of the Hermite function has been applied to this work, so now it is possible to reach the high polynomial order required for the three tests presented.

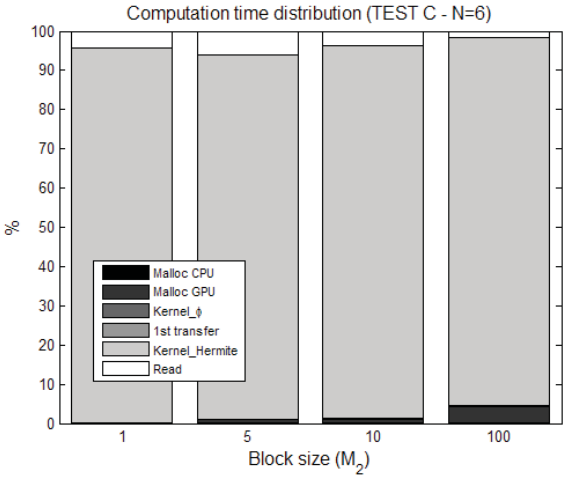


Fig. (8) Computation time distribution of the different tasks involved in the GPU implementation of **Test C** for $N=6$.

CONCLUSIONS

In this paper, an algorithm for the parallel characterization of beats by means of Hermite functions using GPUs was presented. Hermite polynomials of orders ranging from 6 to 30 were considered. The CUDA-based parallel code was explained in detail and performance results were presented. A detail timing analysis was performed for the different implementations presented.

Speedups up to 186× were obtained for off-line processing and up to 110× for on-line processing. The GPU off-line processing of long ECG recordings enables reducing computation time of a Holter recording from several hours to a couple of minutes. As for on-line processing, both the CPU and GPU are able to work in real-time, although the GPU outperforms the former. The benefits of using a much faster GPU-based implementation in the real-time characterization of beats is that this leaves plenty of time to implement a sophisticated real time arrhythmia classification algorithm and, thus, to improve the accuracy of arrhythmia detection.

The authors propose as future research lines the addition of a classification stage for both off-line and

on-line processing (Lagerholm ITB 2000)[22] (Barbakh IJN 2008)[24].

ACKNOWLEDGEMENTS

This work was supported by the University San Pablo CEU under the grant PPC12/2014. D. G. Márquez is funded by an FPU grant from the Spanish Ministry of Education (MEC) (Ref. AP2012-5053).

DISCLOSURE

Part of this article has been previously published in International Work-Conference on Bioinformatics and Biomedical Engineering, Granada (Spain), pp. 527-538, 2014.

REFERENCES

[1] World Health Organization: Global Status Report on Noncommunicable Diseases 2010. http://whqlibdoc.who.int/publications/2011/9789240686458_eng.pdf?ua=1 (Accessed September 27, 2014)

[2] Reddy KS, Yusuf S. Emerging epidemic of cardiovascular disease in developing countries. Circulation 1998; 97(6): 596-601.

[3] Gaziano TA, Bitton A, Anand S, Abrahams-Gessel S, Murphy A. Growing epidemic of coronary heart disease in low-and middle-income countries. Curr Probl Cardiol 2010; 35(2): 72-115.

[4] Swan M. Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0. J Sensor Act Net 2012; 1(3): 217-253.

[5] Min C, Gonzalez S, Vasilakos A, Cao H, Leung V C. Body area networks: A survey. J Mob Net Applic 2011; 2: 171-193

[6] Willems J L et al. The diagnostic performance of computer programs for the interpretation of electrocardiograms. New Eng. J. Med 1991; 325: 1767–1773.

[7] de Chazal P, O'Dwyer M, Reilly RB. Automatic classification of heartbeats using ECG morphology and heartbeat interval features. IEEE Trans Biomed Eng 2004; 51(7): 1196-1206.

[8] Kiranyaz S, Ince T, Pulkkinen J, Gabbouj M. Personalized long-term ECG classification: A systematic approach. Exp Syst App 2011; 38: 3220-3226.

[9] Hamilton PS, Tompkins WJ. Quantitative Investigation of QRS Detection Rules Using the MIT/BIH Arrhythmia Database. IEEE Trans Biomed. Eng 1986; 33(12): 1157-1165.

[10] Sörnmo L, Laguna P. Bioelectrical signal processing in cardiac and neurological applications. Elsevier, 2005.

[11] Young TY, Huggins WH. On the representation of electrocardiograms. IEEE Trans on Biomed Eng 1963; 10:86-95.

[12] Osowski S, Linh TH, Markiewicz T. Support vector machine-based expert system for reliable heartbeat recognition, IEEE Trans Biomed. Eng 2004; 51(4): 582-589.

[13] Haraldsson H, Edenbrandt L, Ohlsson M. Detecting acute myocardial infarction in the 12-lead ECG using Hermite expansions and neural networks. Artificial Intelligence in Medicine 2004; 32(2): 127-136.

[14] Kirk DB, Hwu WmW. Programming Massively Parallel Processors: A Hands on Approach. 1st ed. Morgan Kaufmann Publishers Inc: San Francisco 2010.

[15] Nickolls J, Dally W. The GPU Computing Era. IEEE Micro 2010; 30(2): 56-69.

[16] Garcia-Molla V et al. Adaptive step ODE algorithms for the 3D simulation of electric heart activity with graphics processing units. Comp in Biology and Med 2014; 44: 15-26.

[17] Zhang Q, García JM, Wang J, Hou T, Sánchez HEP. A GPU based Conformational Entropy Calculation Method. Proc. of 1st Int Work-Conference on Bioinformatics and Biomed Eng; 2013 March 18-20; Granada: Copicentro Editorial 2013.

[18] Moody GB, Mark RG. The impact of the MIT-BIH arrhythmia database. IEEE Eng. in Medicine and Biology Mag. 2001; 20(3): 45-50.

[19] Márquez DG, Otero A, Félix P, García CA. On the Accuracy of Representing Heartbeats with Hermite Basis Functions, Proc. of BIOSIGNALS; 2013 Feb. 11-14; SciTePress 2013; pp. 338-341.

[20] Gil A., Caffarena G., Márquez DG, Otero A. Hermite Polynomial Characterization of Heartbeats with Graphics Processing Units. Proc. of 2nd Int Work-Conference on Bioinformatics and Biomed Eng; 2014 April 11-14; Granada: Copicentro Editorial 2014.

[21] Márquez DG, Otero A, Félix P, García CA, Presedo J. A study on the representation of QRS complexes with the optimum number of Hermite functions. Biomed Signal Proces. 2015. *In Press.*

[22] Lagerholm M, Peterson C, Braccini G, Edenbr L, Sörnmo L. Clustering ECG complexes using Hermite functions and self-organizing maps. IEEE Trans. Biomed. Eng 2000; 47: 838-848.

[23] Brodtkorb A, Dyken C, Hagen T, Hjelmervik J, Storaasli O. State-of-the-Art in heterogeneous computing. ACM Trans. Des. Autom. Electron. Syst. 2010; 18(1): 1-33.

[24] Barbakh W, Fyfe C. Online Clustering Algorithms. Int. J. Neural Syst. 2008; 18(3): 185-194.