Ricardo Daniel Domingos Chorão

Licenciatura em Ciências de Engenharia Biomédica

Parallel Programming in Biomedical Signal Processing

Dissertação para obtenção do Grau de Mestre em
Engenharia Biomédica

Orientador: Prof. Doutor Hugo Filipe Silveira Gamboa

Júri:

Presidente:	Prof. Doutor Mário António Basto Forjaz Secca
Arguentes:	Prof. Doutor José Luís Constantino Ferreira
Vogais:	Prof. Doutor Hugo Filipe Silveira Gamboa
	Prof. Doutor José Luís Constantino Ferreira

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outubro, 2012

Parallel Programming in Biomedical Signal Processing

Aos meus pais

Acknowledgements

In the end of one of the most prolific periods of my life, I would like to thank the people that made this rewarding learning experience possible.

First of all I would like to thank Professor Hugo Gamboa for welcoming at PLUX - Wireless Biosignals, S.A. and for agreeing to work with me during the last semester of a 5 year academic route. Your guidance and the discussions we had proved invaluable and have given me a new perspective when I needed it.

I am also grateful to all the staff at PLUX for the kind and humorous moments we shared in the last months and for making me feel welcome and part of a great team.

I want to thank Joana Sousa for her support and for pushing me further, helping me to achieve more.

I am also thankful to Doctor Mamede de Carvalho, for helping me acknowledge the potential of the developed work.

My sincere appreciation to Neuza Nunes for enlightening me when I most needed and for always helping me to learn with my mistakes.

A special thanks to my friends Nuno Costa, Rodolfo Abreu, Angela Pimentel and Diliana Santos. Your support helped keeping me motivated and it was a pleasure to have you around.

I am also grateful to my good friend Sara Costa, Sérgio Pereira, João Rafael and Ana Patrícia for their support during these last years of my life and for all the laughs we shared.

Finally, my deepest appreciation to my parents, Luís and Isabel, my brother Luís, and my dearest friend Cody for their unconditional support and for showing me what is most important in life.

Abstract

Patients with neuromuscular and cardiorespiratory diseases need to be monitored continuously. This constant monitoring gives rise to huge amounts of multivariate data which need to be processed as soon as possible, so that their most relevant features can be extracted.

The field of parallel processing, an area from the computational sciences, comes naturally as a way to provide an answer to this problem. For the parallel processing to succeed it is necessary to adapt the pre-existing signal processing algorithms to the modern architectures of computer systems with several processing units.

In this work parallel processing techniques are applied to biosignals, connecting the area of computer science to the biomedical domain. Several considerations are made on how to design parallel algorithms for signal processing, following the data parallel paradigm. The emphasis is given to algorithm design, rather than the computing systems that execute these algorithms. Nonetheless, shared memory systems and distributed memory systems are mentioned in the present work.

Two signal processing tools integrating some of the parallel programming concepts mentioned throughout this work were developed. These tools allow a fast and efficient analysis of long-term biosignals. The two kinds of analysis are focused on heart rate variability and breath frequency, and aim to the processing of electrocardiograms and respiratory signals, respectively.

The proposed tools make use of the several processing units that most of the actual computers include in their architecture, giving the clinician a fast tool without him having to set up a system specifically meant to run parallel programs.

Keywords: Parallel processing, parallel algorithms, biosignals, signal processing, heart rate variability, respiration.

Resumo

Os pacientes com doenças neuromusculares e cardiorespiratórias precisam de ser monitorizados continuamente. Esta monitorização origina grandes quantidades de informação multivariada que precisa de ser processada em tempo útil, de modo a que possa ser feita a extracção das caracteristícas mais importantes.

O processamento paralelo, inserido na área das ciências da computação, surge naturalmente como uma forma de dar resposta a este problema. Para que o processamento paralelo tenha sucesso é necessário que os algoritmos existentes de processamento de sinal sejam adaptados às arquitecturas modernas de sistemas computacionais com vários processadores.

Neste trabalho são aplicadas técnicas de processamento paralelo de biosinais, conjugando o domínio computacional e o domínio biomédico. São feitas várias considerações acerca de como desenhar algoritmos paralelos para processamento de sinal, seguindo sobretudo o paradigma de programação "data parallel". A ênfase é dada ao desenho de algoritmos e não aos sistemas computacionais que os executam, apesar de serem abordados os sistemas de memória partilhada e de memória distribuída.

Foram desenvolvidas duas ferramentas de processamento de sinal que integram alguns dos conceitos de paralelismo mencionados ao longo do trabalho. Estas ferramentas permitem uma análise rápida e eficiente de biosinais de longa-duração. Os dois tipos de análise feita incidem sobre a variabilidade da frequência cardíaca e sobre a frequência respiratória, partindo de sinais de electrocardiograma e de respiração, respectivamente.

As ferramentas fazem uso das várias unidades de processamento que a generalidade dos computadores actuais apresenta, proporcionando ao clínico uma ferramenta rápida, sem que este tenha a necessidade de montar um sistema dedicado para a execução de programas em paralelo.

Palavras-chave: Processamento paralelo, algoritmos paralelos, biosinais, processamento de sinal, variabilidade da frequência cardíaca, respiração.

Contents

List of Figures

List of Tables

Acronyms

AAL Ambient Assisted Living

ALS Amyotrophic Lateral Sclerosis

ANS Autonomic Nervous System

AWS Amazon Web Services

BVP Blood volume pressure

CPU Central Processing Unit

ECG Electrocardiogram

GUI Graphical User Interface

GPU Graphical Processing Unit

HDF5 Hierarchical Data Format 5

HRV Heart Rate Variability

PSD Power Spectral Density

RAM Random Access Memory

RMS Root Mean Square

SFTP Secure Shell File Transfer Protocol

SSH Secure Shell

1

Introduction

1.1 Motivation

The ever increasing development of clinical systems for continuous monitoring of patients' biosignals has become widely available, not only in clinical facilities but also at home in an ambient assisted living (AAL) environment.

The continuous monitoring of the patients vital signals gives rise to huge amounts of data which need to be processed as soon as possible. The collected information is most of the times multivariate data, composed by many types of biosignals, all of them acquired simultaneously. Examples of such biosignals are the electrocardiogram, respiration, electroencephalogram or electromyogram [2].

The detection of changes in the patients state must be fast and reliable in order to act in a preventive manner and avoid complications for the subject. This detection is achieved through the extraction of relevant features from the collected data.

Real-time analysis is a common practice nowadays. The most illustrative example is the real-time QRS detector and its well known beep noises, each one representing one heartbeat. There are other types of analysis that can only be done after the data acquisition process is finished. This long-term analysis may be very significant since it allows the assessment of the evolution of the patients' vital signals over a long period of time. Moreover, the analysis of long-term biosignals allows the detection of low-frequency phenomena, which can not be analysed otherwise.

In this work we attempt to contribute to the processing of long-term records using parallel programming techniques. Parallel programming techniques come naturally as a way to address this issue since they have the potential to reduce the execution time of an algorithm, provided that the algorithm was designed accordingly (it must be a parallel

algorithm).

One of our goals was also to make the developed parallel processing algorithms accessible to the clinician and the researcher, so that in order to apply them to long-term records one does not need to have programming skills.

We also introduce some guidelines to design parallel algorithms following the data parallel model, in a very similar way to the *MapReduce* programming model [3].

This work was developed in cooperation with PLUX, Wireless Biosignals, S.A. [4], which kindly provided the long-term biosignals that allowed to test the performance of the developed algorithms and signal processing tools. The biosignals were acquired by amyotrophic lateral sclerosis patients.

1.2 State of the art

The application of parallel programming techniques to biomedical signal processing is not entirely new. In the field of biomedical image processing (which is typically computationally intensive), parallel programming is used frequently with graphical processing units (GPUs) instead of the commonest central processing units (CPUs) to process the data [5, 6].

In the field of bioelectric signals, parallel processing techniques have been used with several purposes, namely filtering [7], independent component analysis [8] (a common electroencephalogram analysis technique) and ECG QRS detection [9].

In this work some of these techniques are used in the design of parallel algorithms. These algorithms are to be part of heart rate variability and respiratory analysis tools.

There are several heart rate variability analysis tools, such as "Biopac HRV Algorithm" [10], "HRVAS: HRV Analysis Software" (using MatLab) [11] and "HRV Toolkit" (Physionet). They all provide frequency and time domain analysis of ECG records. However they lack interactivity and do not allow one to deal directly with long duration ECGs, both in the visualization and in the processing parts. Moreover, the available tools do not work from the raw ECG. It is assumed that the QRS complexes were previously detected elsewhere. This dependency of another tool to process the ECG directly is not *user friendly*. HRV analysis tools such as the HRVAS include parallel processing in its HRV analysis. However the program crashes when attempting to analyse long-term records. Another problem with some of these tools is that they are based on proprietary software, and therefore are not *open source*, which was our intent.

With this in mind, we acknowledged the need to develop an HRV analysis tool that can surpass the existent tools in what portability and software availability are concerned. This tool should also allow an efficient HRV analysis from the raw ECG records (as they were acquired, without previous pre-processing) and be able to analyse long-term records based on parallel programming techniques. The tool should be user-friendly and allow a simultaneous visualization of processing results and ECG records, for a more precise analysis, which none of the aforementioned tools does.

Regarding the analysis of respiratory records, to our knowledge that kind of analysis is not incorporated in any signal processing tool. There are many examples in the literature of the analysis of respiration, extracted from ECG records and not from respiratory records themselves [12, 13, 14]. The respiratory analysis from respiratory records rather than from ECG records is more accurate. To design and develop a tool that performs such an analysis (and especially over long-term records), would represent an innovation in the biosignal processing area.

1.3 Objectives

The main goal of this work is the application of parallel programming techniques in the field of biomedical signals processing. The application of such techniques arises from the need to process long-term multivariate data, obtained from continuously monitoring patients. For that, several parallel algorithms, following the data parallel programming model and a model similar to the *MapReduce* [15] programming model were designed, the most important being a ECG QRS detector and a respiratory cycles detector.

Another objective of this work was to make the developed parallel algorithms accessible to the clinician and the researcher, even if they do not have a background in programming. To address this particular goal, the algorithms were integrated in a biosignals processing tool, in two modules:

1. a heart rate variability (HRV) analysis tool;
2. a respiratory analysis tool.

The proposed tools are highly portable and user friendly and allow an efficient processing of long duration biosignals.

The developed tools are particularly suited for the analysis of long-term records, since they were integrated in a previously existing tool which aimed at the visualization of long-term biosignals [16]. This way, the user is able to thoroughly inspect the biosignals and the processing results simultaneously, in a synchronized way, making it much easier and faster to unravel fundamental analysis features.

A report generation feature was also a pre-requisite of the projected processing tools. The objective was to find a way of summarizing the most important analysis results present in each record and make them accessible in a portable format (apart from the processing tool itself) so that they could be carried and properly annotated by medical personnel.

2

Concepts

In this chapter, some concepts with relevance to the comprehension of this work are presented. They range from biosignals to parallel computing and parallel programming models, and its application in the analysis of biosignals.

2.1 Biosignals

Biosignal is a term that represents all kinds of signals that can be recorded from living beings. They are typically recorded as univariate or multivariate time series and contain information related to the physiological phenomena that originated them. This information can be extracted by signal processing techniques and may then be an important input for clinical research or medical diagnosis.

As it will be presented in the next section, there are several types of biosignals, its nature ranging from electrical to mechanical. Special emphasis will be given to the electrocardiogram (ECG) and to the respiratory signals since an important part of this work is focused on their analysis.

2.1.1 Biosignals types

There are many ways a biosignal can be generated by a living being, but the one thing they all have in common is that they somehow reflect physiological events. Recording a particular type of signal in order to understand the cause of a problem is a common practice in medicine. The advances in mathematics, computing science and medicine have given us the tools and the knowledge to correctly analyse and understand the significance of a particular feature extracted from a biosignal. Nowadays, digital signal

processing has made it possible to understand that, as the organ systems in our body are connected and do not behave independently, so it is with biosignals. This connection allows us to draw conclusions from biosignal analysis and feature extraction that would not seem related *a priori* to the nature of the biosignals.

Biosignals may have several physiological origins, according to which they are classified. In this work, the biosignal types used were [1, 2]:

- **Bioelectric signals**: these signals are generated by nerve or muscle cells. In this situation an *action potential* is measured with surface or intramuscular electrodes. The electrocardiogram, electromyogram and the electroencephalogram are examples of such signals;
- **Biomechanical signals**: these signals are produced by measurable mechanical functions of biological systems, such as force, tension, flow or pressure. Examples of such signals are the BVP (blood volume pressure), accelerometry (which measures acceleration) and the respiratory signal, which is acquired with a piezoelectric transducer, typically placed in the chest. An example of a respiratory signal is depicted in figure 2.1.

Other types of biosignals are biomagnetic signals, biochemical signals, bioacoustic signals and biooptic signals.

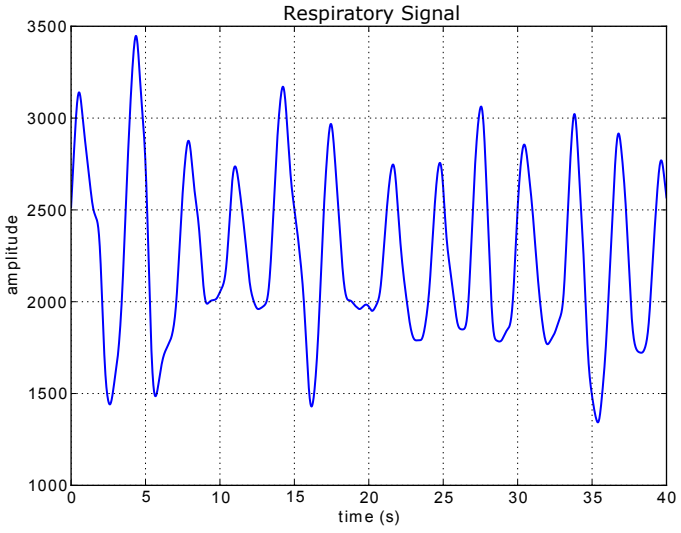


Figure 2.1: The respiratory signal is generated by the force exerted by the chest on a piezoelectric material. The deformation of the material is proportional to the expansion of the rib cage and is translated into the amplitude of the signal.

Electrocardiogram

The electrocardiogram is the signal recorded from the electrocardiography technique, a tool for evaluating the electrical activity of the heart. Figure 2.2 illustrates a typical representation of a normal ECG, which presents a very distinct waveform.

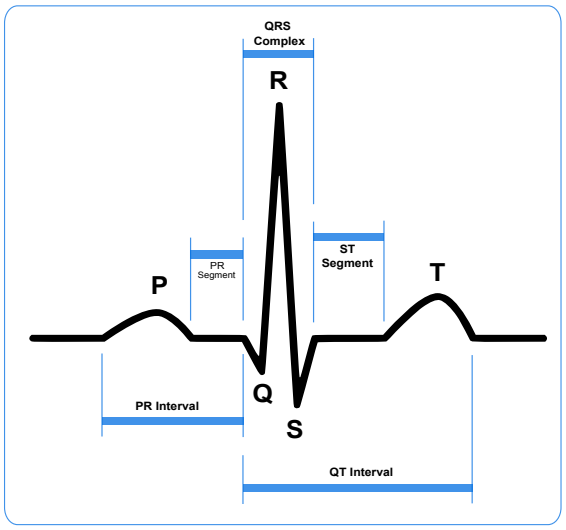


Figure 2.2: A typical ECG, with highlighted features.

Each feature highlighted in fig.2.2 corresponds to a different event in the cardiac cycle.

The first deflection, the P wave corresponds to current flows during atrial depolarization. It has a duration of 80-100 ms. The PR interval, with a typical duration of 120-200 ms, represents the onset between atrial and ventricular depolarization. A PR interval greater than 200 ms indicates an AV conduction block.

The QRS complex, the most prominent feature of the ECG, is the result of ventricular depolarization. Its normal duration is 60-100 ms. If the QRS complex duration is greater than 100 ms, there may be a block in the impulse path, which results in the impulses being conducted over slower pathways within the heart.

The T wave is the result of ventricular repolarization. Atrial repolarization does not usually show on an ECG record because it occurs simultaneously to the QRS complex [17].

The shapes and sizes of the P wave, QRS complex and T waves vary with the electrodes locations. Therefore, it is a typical clinical practice to use many combinations of recording locations on the limbs and chest (ECG leads) [18, 17].

When a muscle contracts, it generates an action potential. After that, there is a long *absolute refractory period*, which, for the cardiac muscle, lasts approximately 250 ms. During this period the cardiac muscle can not be re-excited, which results in an inability for heart contraction [18]. Therefore, a theoretical heart rate limit of about 4 beats per second, or 240 beats per minute can be considered.

Respiration

The respiratory signal has a far more simple waveform than the ECG. It is generated by expansion and contraction of the rib cage, which exerts a force over a respiratory band placed in the chest. From what can be observed in figure 2.1, we see that the signal

consists of oscillations, more or less periodical. The measurement units are amplitude, since what matters is the displacement of every respiratory cycle, relative to each other. In chapter 4 the amplitudes are normalized so that the observable amplitude values are not dependent on the gain factor of the acquiring device. The spacing between every breath (also referred in this work as a respiratory cycle) allows the calculation of the breath frequency. In the respiration tool described in chapter 4 the maximum breath frequency is considered to be 40-60 cycles/minute [19]. This limit is important from an algorithmic standpoint because it allows the rejection of respiratory cycles which are detected less than 1s apart (detection errors).

2.1.2 Heart rate variability

Heart rate variability (HRV) is a method of physiologic assessment which uses the oscillations between consecutive heart beats to evaluate the modulation of the heart rate by the autonomic nervous system (ANS) [20]. HRV is also referred to as RR variability because of the way these oscillations are typically extracted from an ECG record - by measuring the RR intervals, as depicted in fig.2.3.

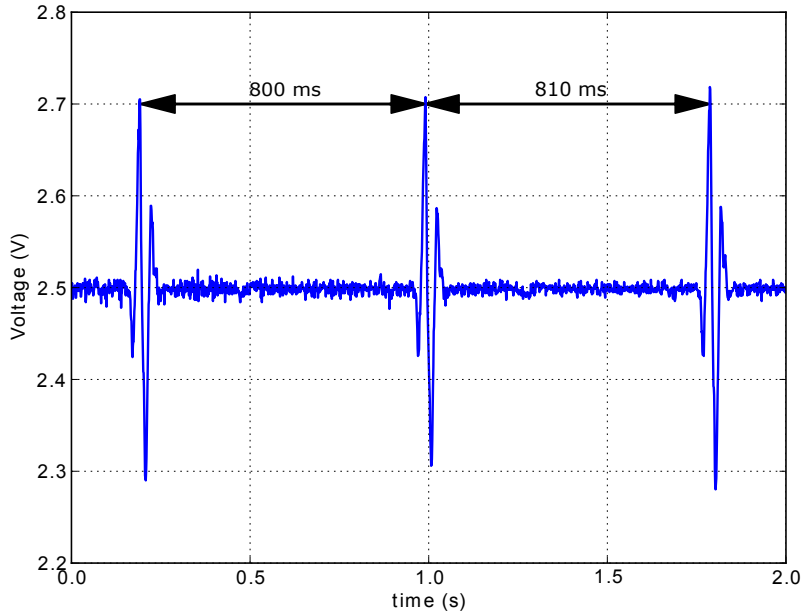


Figure 2.3: Two RR measurements between three consecutive heart beats.

There are many HRV analysis techniques, the most important being frequency domain analysis and time domain analysis [21].

Time domain analysis of HRV allows calculation of the heart rate, statistical parameters, tachogram (evolution of the RR interval duration in time) and instant heart rate representations. It also allows histogram representations which represent the RR intervals durations distribution.

Frequency domain analysis of HRV is based on power spectral density (PSD) analysis of RR intervals and provides basic information about how power distributes as a function of frequency. The power related to different bands relates to different branches of the ANS, which makes the power distribution a very important tool to identify ANS related problems.

HRV analysis has many clinical applications, posing as an indicator of certain diseases. Several studies state the reduced HRV in patients with amyotrophic lateral sclerosis (ALS) [22, 23]. In normal subjects, large changes in successive RR intervals (> 50 ms) occur frequently, but irregularly. In patients with diabetes, reduced HRV may help detecting early cardiac parasympathetic damage [24].

2.1.3 Respiratory analysis

The analysis of respiratory signals has applications in several fields, namely:

- In diagnosing sleep disorders, such as sleep apnea, characterized by pauses in breathing or abnormally low breathing during sleep [25];
- Assessment of mental load and stress and detection of emotional changes [26];
- Respiratory monitoring in athletes to determine ventilation levels and study the relation with their performance [27].

2.1.4 Biosignals acquisition

In biosignals acquisition, the sensor is the part of the instrument that is sensible to variations in the physical parameter to be measured. The sensor must be specific to the nature of the biosignal to be measured. It provides an interface between the biological medium and electrical recording instruments, by converting the physical measurand into an electric output.

High-precision, low-noise equipment is often needed, because the signal (which is normally small in amplitude) typically contains unwanted interferences. Such interferences may be the 50 Hz noise from the electronic equipment, caused by the lighting system or intrinsic interferences like the contamination of an ECG record by the electric activity of adjacent muscles.

The typical steps followed throughout data acquisition are depicted in figure 2.4

Because the signals will most likely be used for medical diagnostic purposes, it follows that the information contained in them must not be affected or distorted by amplification, analog filtering or analog-to-digital conversion.

The analog-to-digital conversion follows analog filtering and signal amplification. The analog-to-digital converter (A/D converter) is used to transform analog signals into digital signals. It measures the input analog signal and gives a numerical representation to the signal as its output. The discrete signal has then a numerical representation and can be easily stored in a computer. The digital signal is not an exact representation of the

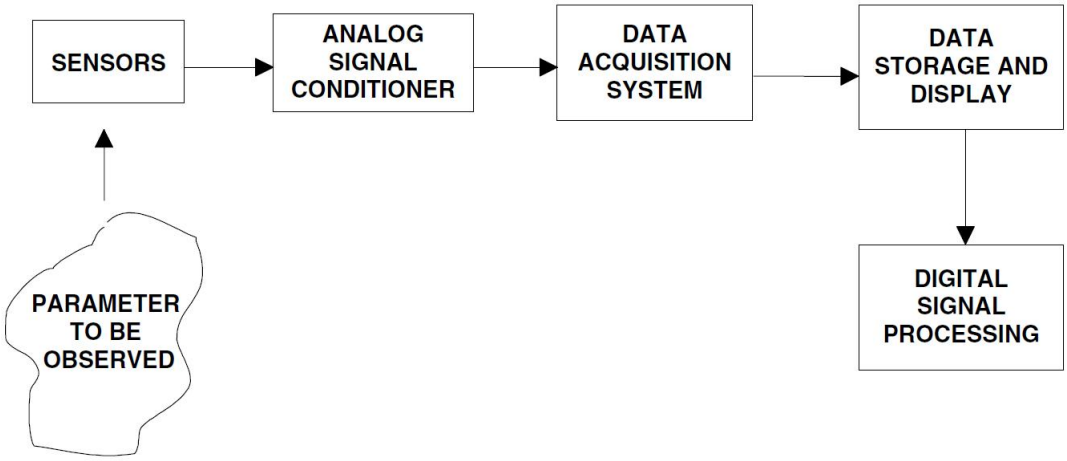


Figure 2.4: From [1]. The data acquisition system converts the analog signal into a digital signal that can be stored. Signal processing techniques may then be applied to reduce the noise (signal enhancement).

original signal, it is only an approximation. It is generated by repeatedly sampling the amplitude of the analog signal at fixed time intervals. The inverse of that time interval is called the sampling frequency of the digital signal. Figure 2.5 illustrates this sampling process.

Besides the sampling frequency, there is another process which affects the degree of similarity between the original analog signal and the digital one called quantization. As mentioned, a numerical value is assigned to a particular amplitude in the A/D converter. Quantization reduces the infinite number of amplitudes (a continuous interval) into a limited set of values. The A/D converter resolution determines the number of bits of the signal values. The typical number of bits of the discrete samples is 8, 12 or 16, for most A/D converters. An insufficient number of bits can lead to information loss during the conversion process, which in turn may lead to errors in the interpretation of the data.

The sampling frequency, if too low, can lead to the same problems as low A/D converter resolution. If too high, however, it may give rise to the generation of huge data collections, especially when we are dealing with long duration records, for example whole nights. To process that data in a reasonable amount of time, we may need parallel programming techniques.

2.1.5 Biosignals processing

Biomedical signals processing is an intermediate step between acquiring the signal and drawing conclusions from the data.

Most of the times, the relevant information to be extracted from a record is concealed, not directly seen through raw data observation. The aim of digital signal processing is revealing those "hidden" signal features, with relevance for direct analysis [2].

Sometimes, before the processing techniques can be applied, the data must undergo

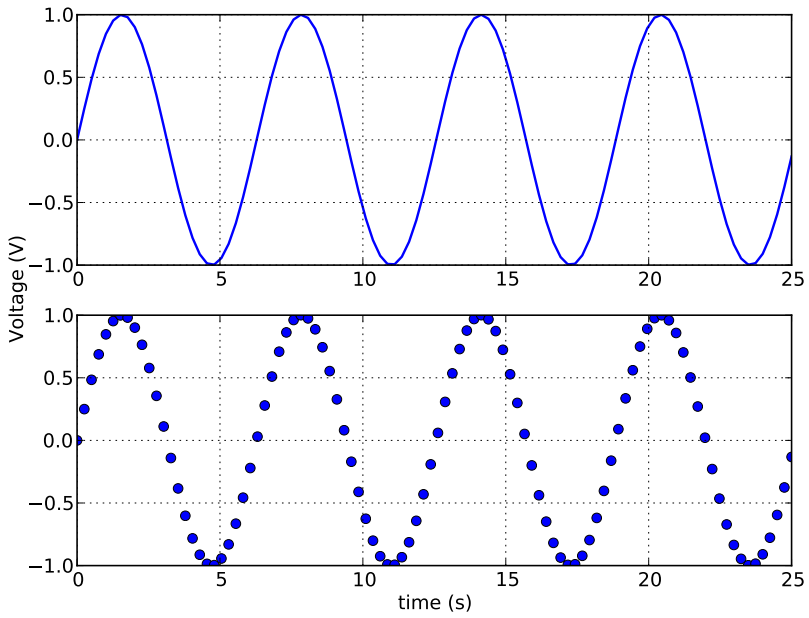


Figure 2.5: The analog signal and the corresponding digital signal.

a pre-processing phase, in order to make the data suitable for further analysis. A typical pre-processing technique, noise removal, is illustrated in figure 2.6.

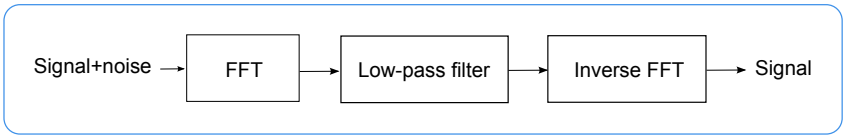


Figure 2.6: A pre-processing technique. Here, the noise is removed from the signal by application of a low-pass filter in the frequency domain.

After the pre-processing phase, the signal is ready for features extraction and interpretation.

The algorithms applied make possible not only an enhanced signal visualization, but also the signals statistical analysis and the extraction of certain parameters, specific of the nature of the biosignals. One example of a such algorithm is an algorithm that detects the R times in an ECG record.

2.2 The Hierarchical Data Format 5

The *Hierarchical Data Format 5* (HDF5) is a file format and a software, specifically designed to store and access large datasets. HDF5 files do not have a theoretical limit in size. These files allow storage of very complex structures and very fast access to random portions of

the data, as well as writing data to the files at very high speeds [28].

By continuously monitoring patients, remotely or in a medically controlled environment, huge amounts of data are generated. In healthcare, a fast response to evidence of a medical condition or disease is of paramount importance and can save lives. Hence the pressing need for fast data processing and analysis. Because of the above, *.hdf5 (or *.h5) files are suitable for storage of long duration biosignals and subsequent analysis.

H5 files have a hierarchical structure composed by two types of objects:

- Groups - a group is a container structure, similar to a folder. A group may contain an arbitrary number of other groups and datasets;
- Datasets - a dataset is the structure that contains the data. The data in a dataset must be of homogeneous type, but in the same file several datasets may hold data of different types.

This file format also supports the introduction of metadata called attributes. Each group can have its own attributes and the file, as a whole structure can have attributes associated too. When the files contain medical information - such as recorded biosignals -, the attributes are very useful to store key information about the subject, the type of the record or the data. Such attributes may be the acquisition date and time length of the record, the sampling frequency or the patient's name.

To manage files in this format, there is a *Python*[29] library called *h5py* [30]. This library allows file writing and data manipulation in a very convenient way, in a manner which is similar to another scientific *Python* library, the *numpy* [31].

2.3 Central processing unit

A resource is any component with limited availability within a computer system . This section provides a description of the most important system resource to account for when dealing with parallel programming - the CPU. Examples of other important resources are the, random access memory (RAM) hard disk space or a network throughput.

The CPU is the most important system resource concerning parallel programming. The whole point of dividing a problem into several tasks and run them concurrently is to be able to use more computational power at the same time.

Every modern computer can perform several tasks at the same time. While running a user program, a computer can also be printing text to a screen at the same time. The CPU switches from program to program, running each for several milliseconds. Strictly speaking, at a given instant the CPU is only running one program, but in a second, it has switched back and forth, giving the user the illusion of parallelism. This pseudo-parallelism is managed by the operating system's scheduler. The kernel is a part of an operating system which is responsible for resource management. The scheduler (or CPU

scheduler) is the part of the kernel that determines when and for how long a process is allowed to run [32].

A parallel job scheduler allocates nodes for parallel jobs and coordinates the order in which jobs are run. While several jobs are executed simultaneously, others are enqueued and wait for nodes to become available. Two very important goals of a scheduler are the optimization of the number of jobs completed per unit of time and keeping the utilization of computer resources high [33].

In computers with multiple processors, true parallelism can be achieved because different programs can be run at exactly the same instant in different processors. In each processor, the scheduling of the different processes takes place.

When a sequential program is run in a system with multiple processors, the program runs in a single processor. For the program to be executed by multiple processors, some specification must be given by the user, identifying the parts that can be run in parallel.

Figure 2.7 illustrates this concept. It consists of two screenshots from a dual core Ubuntu machine, during the execution of a program. Initially the system is idle and there is low CPU usage in both cores. There are small oscillations in the CPU usage related to background processes.

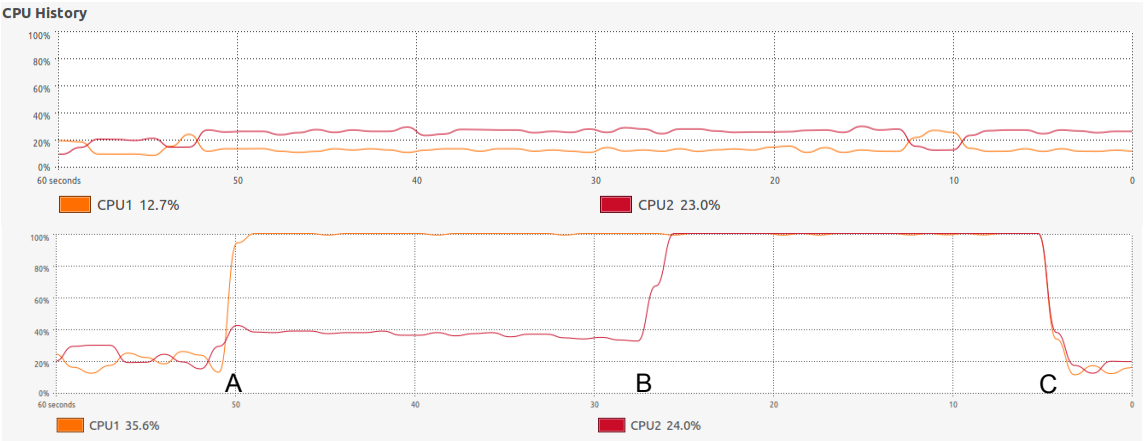


Figure 2.7: These screenshots show the CPU usage over time in a system with 2 cores.

At instant A, a single task is executed. It is automatically assigned to one of the CPUs by the operating system, and it can be seen that there is only one CPU with approximately 100% usage. At instant B, another task is executed, and it is assigned to the other CPU. In this instant, the two CPUs are at approximately 100% - there is true parallelism. At instant C, the execution is terminated and the immediate decrease of CPU usage is quite clear.

2.4 Parallel programming

2.4.1 Overview

Nowadays, computers with multiple processors are widely available. The total computational power of such computers is inexpensive, if we were to compare it with a single central processing unit (CPU) with the equivalent computational power.

According to Moore's law [34], the number of transistors that can be placed on a chip doubles every two years. This trend drove the computing industry for 30 years, leading to increased computer performance, while decreasing the size of chips and increasing the number of transistors they contained . However, transistors can't shrink forever, namely because of heat generation. In 2004, a point was reached where performance increases slowed to about 20 % a year. In response, manufacturers are building chips with more cores instead of one increasingly powerful core [35].

A single core processor runs multiple programs "simultaneously" by assigning time slices to each program, which may cause conflicts, errors and slowdowns . Multicore processors increase overall performance by being able to handle more work in parallel. The different cores can share on-chip resources, which reduces the cost of communications, as opposed to systems with multiple chips [36].

To take advantage of this new architecture, a new way of thinking must be employed, and so has the way how software and algorithms are designed. This way of designing algorithms is called parallel programming.

In parallel programming, a single problem is divided into several tasks that can be solved concurrently by different processing units. This approach allows faster problem solving by increasing the overall computational power and makes it possible to tackle problems that require huge amounts of memory and time to be solved and that could not be solved otherwise. The total CPU time required to solve a problem is the same or even higher than the CPU time it would take for the problem to be solved using a single processor. However, the human time it takes for the problem to be solved is much lower.

A schematic of a distributed computation, taking place in different computing units can be seen in fig.2.8.

A number of complex problems in science and engineering are called "grand challenges" and fall into several categories, such as [37]:

- Quantum chemistry, statistical mechanics and relativistic physics
- Weather forecast
- Medicine, and modelling of human bones and organs
- Biology, pharmacology, genome sequencing, genetic engineering

These problems can not be solved in a reasonable amount of time without the use of parallel computing platforms and appropriate parallel algorithms.

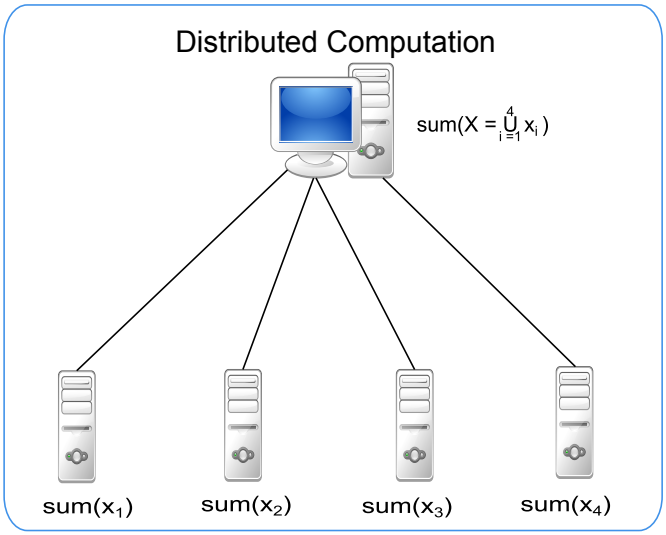


Figure 2.8: An example of a distributed computation. The problem consists of computing the sum of a large set of numbers. The data is partitioned so that each computing node sums each partition. Finally, the master computer only has to sum the results.

The computing platform may be a multicore computer or several independent computers connected in some way, i.e., a computer cluster.

The main idea is that n computers can provide up to n times the computational speed of a single computer, with the expectation that the problem can be completed in $1/n^{th}$ of the time [38]. This is of course an ideal situation, which does not occur in practice, except for some embarrassingly parallel applications - applications that are parallel in nature -, such as Monte Carlo simulations.

Theoretically, the maximum speedup of a parallel program is limited by the fraction of the program that can not be divided into concurrent tasks - serial fraction of the problem. The maximum speedup of a parallel program is given by Amdahl's law [38, 37] in equation 2.1.

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1+(n-1)f} \tag{2.1}$$

Where S is the speedup, n is the number of processors, f is the fraction of the computation that is computed sequentially and t_s is the time the serial execution would take. If the sequential fraction of the program is 0, an ideal situation, the speedup is n, that is, the execution time is reduced by $1/n$. With f different than 0, as occurs in practice, the maximum speedup is given by:

$$\lim_{n\to\infty} S(n) = \frac{1}{f} \tag{2.2}$$

If half of the program is sequential ($f = 0.5$), the best scenario is to reduce the execution time by half (which happens when the other half of the program is executed instantly

in parallel).

The efficiency and speedup factor, two performance related concepts are only defined in chapter 5 where it is most appropriate due to the nature of the chapter.

The holy grail of parallel computing is the automatic parallelization of a sequential problem by a compiler. The compiler would on its own identify the existing parallelism within the program and assure its concurrent execution. Such a compiler does not exist yet, and automatic parallelization only had a limited success so far [39]. There are some programming languages such as Chapel [40], Parallel Haskell [41] and SISAL [42], but in all of them the user must identify with some language construct the parts of the program that can be run in parallel.

2.4.2 Programming models

Regarding problem decomposition, there are two main programming models: data parallelism and control parallelism. A brief introduction to data parallelism and to pipeline parallelism, a special case of control parallelism, is given below:

2.4.2.1 Data parallelism

In data parallelism, multiple processing units apply the same operation concurrently to different elements of a data set. This is the case in the example of figure 2.8. The main feature of data parallel algorithms is their scalability ,i.e., a k-fold increase in the number of processing units leads to a k-fold increase in the throughput (number of results processed per unit of time) of the system, provided there is no overhead associated with the increase in parallelism [37]. In biosignals processing the data parallel approach is quite attractive, especially if we are dealing with huge datasets, which cannot all be processed at once. In the following section, a type of data parallel algorithms, called Map Reduce algorithms is presented.

2.4.2.2 Pipelined parallelism

Pipelining is a technique applicable to a range of problems that are sequential in nature. The problem is divided into a series of tasks, each one to be completed after the other. Each stage contributes to the overall problem and passes on information required by the next step as illustrated in figure 2.9. If all the segments work at the same speed, once the pipe is full the work rate of the pipeline is equal to the sum of the work rates of the segments. A pipeline can be seen as an assembly line, with a simple flow of results and precedence constraints. Unlike data parallelism, this approach is not scalable because the level of parallelism is usually a constant, regardless of the problem size [37, 38]

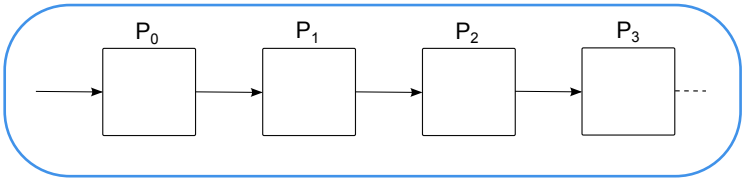


Figure 2.9: A pipelined computation. There are precedence constraints between the different pipelined processes.

2.4.3 *MapReduce* algorithms

MapReduce is a style of computing - programming model and associated implementation -, originally implemented by *Google*. The programming model was presented by the first time by Dean and Ghemawat in 2004 [15]. A schematic of the computation can be seen in fig.2.10.

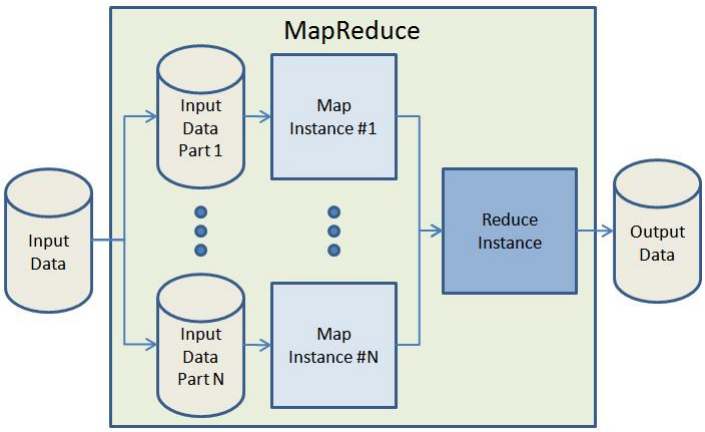


Figure 2.10: Schematic of a *MapReduce* computation

All the user needs to do is write two functions called *Map* and *Reduce*. The *Map* function processes a key/value pair and generates a set of intermediate key/value pairs; the *Reduce* function merges all the intermediate values associated to the same intermediate key.

Programs written in this way are easily parallelized and executed on a large cluster of commodity machines. This implementation is therefore highly scalable.

The system manages the parallel execution in a way that is tolerant to hardware failures, while coordinating the task distribution between the compute nodes. *MapReduce* can be viewed as a library which is imported in the beginning of the problem and handles all the details concerning parallelization, hiding them from the programmer.

A typical *MapReduce* example is the "word count" problem. In this problem the return value is the number of occurrences for each word in a collection of documents.

The computation executes as follows [3]:

1. A set of Map tasks convert input elements to key/value pairs. Here, each Map task

may read a whole document and break it into its sequence of words $w_1, w_2, \ldots, w_n$. The output of the Map task is a sequence of key/value pairs:

$$(w_1, 1), (w_2, 1), \ldots, (w_n, 1)$$

2. The key/value pairs resulting from each Map task (each Map task could be related to a different file) are sorted by key. The sorted key/value pairs are then aggregated. The obtained sequence is of the form:

$$(w_1, [1, 1, 1]), (w2, [1, 1]), \ldots, (w_n, [1, 1, 1, 1])$$

This step is sometimes called "Grouping and Aggregation".

3. Finally, the Reduce task takes input pairs consisting of a key and its list of associated values, and combines those values in a way that was previously specified. The output of the Reduce task is a sequence of key/value pairs consisting of the input key paired with a value, constructed from the list of values received along with that key. In the "word count" example the Reduce task simply has to sum the values in the lists associated to each key. The final result would then be:

$$(w_1, 3), (w2, 2), \ldots, (w_n, 4)$$

A programming model very similar to *MapReduce* is described in chapter 3, along with examples of parallel algorithms that follow this model. Because the programming model is not exactly the same as *MapReduce*, the algorithms are called in this work "*MapReduce* derived algorithms" and are applied in the area of biomedical signal processing. The main similarity with *MapReduce* is the need to define two functions to tackle a problem (a map and a reduce function). The main differences are:

- The map function does not necessarily process key/value pairs - the input data structures may be more complex, and always include a part of the signal to process;
- The sorting and aggregation is not a necessary step.

3

MapReduce Derived Algorithms

In this work, a set of algorithms following the *MapReduce* programming model were developed. They were called *MapReduce* derived algorithms because of the existing similarities between the thought process which originated them and the *MapReduce* programming model itself. In this chapter the underlying programming strategy is described and some examples are presented.

3.1 Computing architecture

The developed algorithms are the parallel versions of the algorithms of an HRV analysis tool, adapted to the analysis of long-term records. The adopted programming model offers high scalability, since the algorithms are meant to be run in a distributed system. The idea of analysing clinical data using a programming model similar to MapReduce is recent and not widespread [43, 44]. The architecture that better suits the correct operation of these algorithms is illustrated in figure 3.1.

The master-slave architecture (also called master-worker model) is characterized by the existence of a master computer which controls the execution of the program [45]. In this model, the master processor executes the main function of the program and distributes work from the central node to worker processors [33], hence being responsible for worker coordination and load-balancing. According to this model, there is no communication between the workers, whatsoever.

The family of the developed parallel algorithms does not rely on a distributed shared memory (where all the physically separated local memories can be addressed as one), although they could be run in such architectures. As shown in figure 3.1, the algorithms' design allows them to be run in a distributed memory environment, which gives them

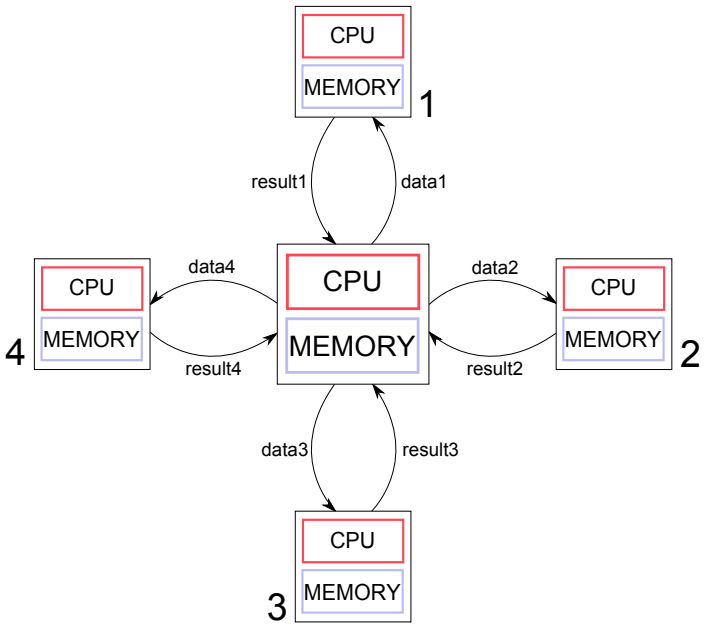


Figure 3.1: Master-slave distributed parallel computing architecture best suited to run the developed *MapReduce* derived algorithms.

more flexibility.

The data parallel model is applied to achieve higher scalability, by partitioning the data. The implementation of this model is very common in scientific computing [46]. By partitioning the data further and further the number of computer nodes (and the overall computing power) is increased, and the original problem may be solved faster. This partitioning process can not be done indefinitely, i.e. there are a few factors that might limit the extent to which the data can be divided. They are:

- The nature of the algorithm: the algorithm may require a minimum amount of signal samples to perform well, and therefore dividing the signals past that limit is counterproductive;
- The task(s) assigned to each computing node should contain enough computations so that the task execution time is large compared to the scheduling and mapping time required to bring the task to execution [45]. The size of a task is described by its granularity. In coarse granularity, each task takes a substantial time to execute and is composed by a large number of sequential instructions [38]; in fine granularity the scheduling and mapping overhead is significant when compared to the task execution time. Therefore, the partitioning level of the data must find a compromise between their granularity and execution time [47].

In figure 3.1 there are 4 computing nodes. Each node has its own local memory and

communicated directly with the master computer. Because the algorithms do not depend on any global variables and the processes on each node do not communicate with each other there is no need of a message passing interface.

It is important to note, however, that not all algorithms have a high degree of data parallelism [48] and to ensure the best possible results one may have to implement other strategies instead (or additionally).

3.2 Detailed features

In order to design the *MapReduce* derived algorithms to solve a specific problem, the problem at hand must be expressed in terms of two functions:

1. a *map* function that processes parts of the signal (passed as input value to the function);
2. a *reduce* function that merges all the outputs of the *map* functions to compute the desired result.

The *map* functions are meant to execute remotely, on different compute nodes, according to the architecture of figure 3.1. The *reduce* function is the last step of the computation and is typically executed in the master computer, after all the workers have finished their computations and communicated the results to the master computer. Because the *reduce* function is run by a single computer, it is very important that its complexity is kept to a minimum. The *reduce* function can in theory be run in parallel as well, although this approach has not been explored in this work. This would be a key process to overcome situations in which the original problem did not exhibit a high degree of data parallelism, and the mapping results would still require a complex treatment in the master computer, during the *reduce* part. When designing such algorithms it is extremely important that the worker nodes are loaded as much as possible, so that the remaining work, taking place in the master computer is as little as possible.

Follows a brief mathematical description of the algorithms designing method:

Let the original problem be represented by function f,

$$y = f(X, t) \tag{3.1}$$

where y is the answer to the problem and X is the signal to process (f can also be viewed as the sequential algorithm and y its result). Note that f is a function of time, since X is a time series with an underlying notion of order.

Before applying the data parallel model it is necessary to divide the signal. This process can be represented by

$$X = \{x_1, x_2, \ldots, x_n\} \tag{3.2}$$

The set represented by equation 3.2 can be:

- a partition, in which case,

$$x_i \cap x_j = \varnothing, i = 1, \ldots, n; j = 1, \ldots, n; i \neq j \tag{3.3}$$

and

$$\bigcup_{i=1}^{n} x_i = X \tag{3.4}$$

In this situation, each computing node would be passed a slice of the signal, all the slices being independent. Note that in this situation, since each node does not have access to the order in which each slice appears in the original signal, the notion of order is not relevant at all. The computation of some statistical parameters, only concerned with the values of the signal (and not its order) can be computed following this method of division. Examples of parameters that can be computed this way are the minimum and maximum of the signal, its mean, standard deviation and histogram.

- a partition, including overlapping regions, in which case equation 3.3 does not hold. Instead,

$$x_i \cap x_j \neq \varnothing, i = 1, \ldots, n; j = 1, \ldots, n; i \neq j \tag{3.5}$$

The overlapping region may be fixed in size, or it may depend on some parameter. The division of X using overlapping regions is typically done when an algorithm has a certain memory, and therefore does not perform well in the edges of the signal.

- a more complex data structure, that might include some information related to the whole signal. In this case, the division gives rise to data structures with more information than $x_i, i = 1, \ldots, n$.

$$X = \{x'_1, x'_2, \ldots, x'_n\}, x'_i = \{x_i, a, b, c, \ldots\}, \tag{3.6}$$

where $a, b, c, \ldots$ represent the additional information related to the whole signal (and/or possibly other individual parts).

Very often each node must know the order of the part to which it will apply the *map* function. This may be because the *map* function depends on the order or because the *reduce* function will need that information so that the mapping results can be sorted.

Other examples of additional information related to the whole signal are its length, its mean value or the number of parts the division of the original record originated.

There is simply no rule to what this additional information includes. It totally depends on the algorithm and must be thought of during the designing process. To identify what information the data structure must include, one must question himself what information is necessary during the *map* and *reduce* steps. Since the *map*

function will surely have a slice of the signal as input value, it must be found out what other information will be required for that computation that is not within the slice of signal. For the *reduce* step, it may also be required some additional information. Typically, the mapping results and the order in which the slices appeared in the original signal will suffice.

It is very interesting to note that some of the information which is included in the dividing data structures could be omitted, if the algorithms were run in a shared-memory environment. It is the case of all the information included in the data structure that is repeated between working nodes (redundant information). This information could be replaced by global variables in a shared memory, to be accessed by the working nodes at execution time.

After the division of the long-term record, the data-structures are assigned to the computing nodes, along with a file containing the code for the *map* and *reduce* functions. Each node will produce its own result:

$$\begin{aligned} y_1 &= map(x_1, t_1), \\ y_2 &= map(x_2, t_2), \\ &\vdots \\ y_n &= map(x_n, t_n) \end{aligned} \tag{3.7}$$

In the case of an embarrassingly parallel computation, the *map* function will be the same (or almost the same) as the sequential version of the algorithm. This is the case with event detection computations (where an events detection function is applied just the same over a slice of the signal as it were over the whole signal), the computation of a sum or the length of a long-term record.

After the mapping results $y_1, y_2, \ldots, y_n$ have been computed, they are transferred to the master computer and passed as arguments to the *reduce* function, which in turn will compute the answer of the original problem:

$$y = reduce(y_1, y_2, \ldots, y_n) \tag{3.8}$$

Globally, the processing undertaken can be summarized by the following equation:

$$f(X, t) = reduce(map(x_1), map(x_2), \ldots, map(x_n)) \tag{3.9}$$

A schematic of the computation can be seen in figure 3.2.

3.3 Examples

In this section a few examples illustrating the principles discussed above will be presented. The importance of the parallel computation of simple statistical parameters such as *mean* and *standard deviation* should not be dismissed. Although computationally light,

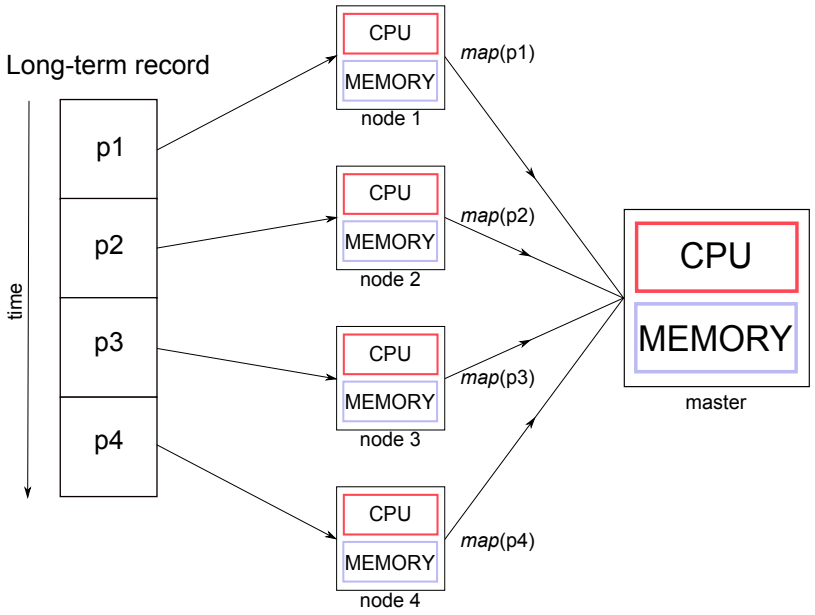


Figure 3.2: Schematic of the whole computation. In this situation the division of the long-term record gives rise to 4 data-structures, which are processed in 4 computing nodes, prior to the application of the *reduce* function by the master computer.

the fastest way to compute these parameters might be in a distributed manner, due to the size of the biosignal (as was mentioned in chapter 2, these parameters might also be computed by a single processor, sequentially, analysing each part at a time). The *standard deviation*, *histogram* and *events detection* MapReduce algorithms will be presented.

3.3.1 Histogram and pNN50

The histogram of a signal is a tool meant for the analysis of the amplitude distribution of the signal samples. Before the computation of the histogram, the number of bins is defined. The bin size will be equal to the amplitude of the signal divided by the number of bins. Sometimes it is only relevant to know the distribution of the signal values between a certain interval, dismissing the samples outside the interval. If so, the maximum and minimum values of the signal do not need to be known beforehand and the bin size will be equal to the interval size divided by the number of bins. In this section the histogram is only meant as the information necessary to compute the graphical representation of this analysis tool and not the figure itself. The *hist* function might be defined as follows:

$$hist(X, nBins) = ((i_1, i_2, \ldots, i_{nBins}), (c_1, c_2, \ldots, c_{nBins})), \tag{3.10}$$

where i_n represents the bin interval limits and c_n the bin count associated to the interval i_n.

If $X = x_1, x_2, \ldots, x_n$, the *map* function is simply:

$$histMap(x, nBins) = hist(x, nBins) \tag{3.11}$$

The associated *reduce* function is:

$$histReduce(m_1, m_2, \ldots, m_n) = \left(m_1[0], \sum_{i=1}^{n} m_i[1] \right), \tag{3.12}$$

where m_i are the mapping results, i.e, the outputs of the $histMap$ function.

Each node computes partial histograms and the reduce function simply adds up the counts from all the different parts.

The pNN50, a widely used HRV analysis parameter can be computed following this process, since the problem involves counting events. The NN50 is the number of interval differences of successive RR intervals greater than 50 ms. The pNN50 is obtained dividing NN50 by the total number of RR intervals. A detected regularity in large changes in successive RR intervals may indicate cardiac parasympathetic damage [24].

Because of the mathematical complexity of the pNN50 MapReduce algorithm, only a description of the steps taken by the *map* and *reduce* functions will be made.

For the computation of this parameter, the *map* function would need to return:

- The number of successive RR intervals spaced by more than 50 ms;
- The number of RR intervals computed in that signal part (equal to the total number of detected ECG peaks less 1);
- The order in which the signal slice appeared in the long-record (this has to be an argument of the map function, passed by the master computer when assigning tasks to the computing nodes);
- The first and last detected peak and the first and last RR interval.

The reduce function would first compute the NN50 parameter. For that, it would:

1. Sort all the mapping results according to the order of the slice they refer to in the original record;
2. Compute the lost RR intervals in the borders of the slices. For that it uses the first peak of a mapping result and the last peak of the previous mapping result and compares the resulting RR interval with the adjacent intervals evaluating if they are spaced by more than 50 ms;
3. Sum all the partial NN50 mapping results and the ones found out in 2., obtaining the NN50 parameter.

It would then divide NN50 by the total number of RR intervals, obtaining pNN50.

3.3.2 Standard deviation

When aiming to write the parallel version of a simple mathematical calculation it is best to fully understand the definition of the parameter. The standard deviation is computed as shown in the following equation:

$$std(X) = \sqrt{E[X^2] - (E[X])^2}, \tag{3.13}$$

where $E[X]$ is the mean value of X. Let $X = \{x_1, x_2, \ldots, x_n\}$. In this case the division of X gives rise to n partitions. The standard deviation *map* function applied to each partition is

$$stdMap(x) = (sum(x^2), sum(x), length(x)) \in \mathbb{R}^3 \tag{3.14}$$

Before presenting the associated *reduce* function it is important to appreciate the size reduction that happened during the mapping process. Initially the standard deviation could not be computed by a single processor, due to memory issues because of the size of the signal. After the mapping process, each signal part "is" reduced to a 3 element tuple. Provided that all the necessary information for the final computation is within that tuple, the master computer will have no trouble (memory-wise) computing the parameter.

The standard deviation *reduce* function will have a number of arguments equal to the number of partitions. Let $m_n = stdMap(x_n)$ and $s = sum(m_1, m_2, \ldots, m_n)$, i.e., s is the sum of all the mapping tuple results element-wise. Then, the standard deviation *reduce* function is:

$$stdReduce(m_1, m_2, \ldots, m_n) = \sqrt{\frac{s[0]}{s[2]} - \left(\frac{s[1]}{s[2]}\right)^2} \tag{3.15}$$

It is easy to verify that equation 3.15 is correct according to the definition (equation 3.13), since:

- $s[0] = \sum X^2$
- $s[1] = \sum X$
- $s[2] = length(X)$

3.3.3 Events detection

The events detection class of algorithms is very well suited for execution under the MapReduce programming model. This is because the problem has an embarrassingly parallel nature and also because, unlike the examples presented so far, the events detection algorithms can be computationally intensive. The embarrassingly parallel nature of the problem is explained by the fact that the events detection algorithms may perform just as well over slices of the signal as they perform over the whole signal. There are many applications of events detection algorithms in biomedical signals processing. A few common examples are:

- a ECG peak detection algorithm (or QRS detector as it is sometimes referred to);
- a respiratory cycles detection algorithm, which allow the computation of the breath frequency from respiratory signals;
- an algorithm for the detection of muscle activity from a EMG record.

These examples are just a sample of the applications of events detection algorithms. Thus the importance of the suitability of these kind of algorithms to run in parallel, which will drastically reduce their execution times. The division of the signal will typically be accompanied of an overlapping portion, since the events detection algorithms do not perform the same in the edges as in the middle of the signal.

In this work, two events detection algorithms were used (although more loosely one could also consider the computation of the pNN50 parameter and the histogram as events detection algorithms): a ECG peak detection algorithm, fundamental for the developed HRV analysis tool and a respiratory cycles detector, just as important, regarding a respiratory analysis tool (these tools will be described in the next chapter).

The *map* function utilized in this class of algorithms is basically the events detection algorithm itself. The computing nodes will search for events over the signal slices that were assigned to them and return the detected events and the order in which their signal slice appeared in the original signal.

The *reduce* function will only have to sort the mapping results (assuming the instant or time interval of the detection is important - it may only matter that an event was detected) and possibly remove some double detections. Double detections might occur due to the detection of the same event in an overlapping region and in the edge of the subsequent partition.

Executing this class of algorithms in a distributed environment has a huge potential, since the speedup factor is approximately proportional to the number of computing nodes. A biosignal of about 7 hours, acquired during the night can be partitioned in time slices that are smaller and smaller, as more and more computing nodes are assigned to the problem. If the algorithm performs well over slices of a few seconds, there is nothing preventing the problem to be solved in approximately that amount of time, provided that enough computing power is available.

Generally, for this type of problems the execution time is:

$$T = map\,time + reduce\,time \tag{3.16}$$

Assuming that the signal slices are equally sized,

$$map\,time \approx \frac{sequential\,time}{number\,of\,partitions} \tag{3.17}$$

The $\approx$ symbol is due to slight increases in the mapping execution time because of the overlapping regions (these increases are negligible compared to the overall execution

time).

The overall execution time becomes:

$$T \approx \frac{sequential\ time}{number\ of\ partitions} + reduce\ time \tag{3.18}$$

This is the reason why the *reduce* function must do as little work as possible and we must load the working nodes as much as possible: the *map* time can be reduced by further dividing the signal, but the speedup will always be limited by the execution time of the *reduce* function.

4

Signal processing tools

4.1 Overview

In this chapter, two signal processing tools developed during this work are described. These processing modules were integrated in a long-term biosignals visualization tool, a result of a previous work [16]. The resulting tool allows the clinician and the researcher to analyse long-term records without having to deal with all the programming and signal processing algorithms directly. An effort was made to analyse long-term records efficiently. In this context, parallel programming techniques came naturally as a way to solve possible bottlenecks that limit the speed at which the signals are processed.

In the next section, the acquisition system is described. Then, the programming architecture of the proposed application and two biosignal analysis tools - a heart rate variability analysis tool and a respiratory analysis tool - are presented, as well as some of their applications.

4.2 Acquisition system

To acquire long-term biosignals is no easy task. The acquisition system must be as comfortable as possible, so that the patients are not disturbed by its utilization. This particular study occurred under the project "wiCardioResp" [4]. This project is developing technology to remotely monitor patients with cardiorespiratory problems and neuromuscular diseases (such as amyotrophic lateral sclerosis) while the patients are comfortably at home. The acquisitions were carried out with the patients agreement, during the night, and last approximately 7 hours.

A patient-friendly way of acquiring the signals was attained with the aid of a chest

wrap, which integrates respiration, ECG and accelerometer sensors (see figure 4.1). The sensors were provided by PLUX, Wireless Biosignals, S.A..

The chest wrap makes it harder for the sensors to detach from the patient due to sleep movements, which would compromise the acquisition.

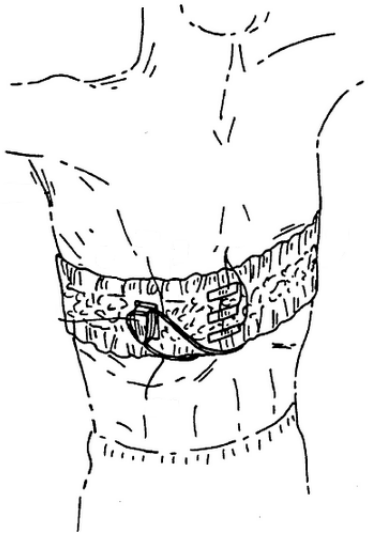


Figure 4.1: Chest wrap with integrated accelerometer, ECG and respiration sensors.

The applications of the analysis of heart rate variability and the analysis of respiratory signals were mentioned in chapter 2. The accelerometry signals make it possible to identify sleep periods in which the patient turned over in bed.

The signals were sampled at 1 kHz and with a 12 bit resolution. They were sent by Bluetooth to a mobile phone and recorded in *.txt format.

4.3 Application architecture

As mentioned, the signal processing tools developed were integrated in a pre-existing tool. A schematic of the architecture of the application is depicted in figure 4.2.

4.3.1 Previously developed work

The tool has a visualization module, developed in a previous work [16], which allows the visual inspection of very long signals. Signals approximately 7 hours long, and sampled at 1 kHz are composed by millions of samples.

First there is a conversion step, in which the *.txt recorded signals are converted to the hdf5 file format, with the advantages mentioned in chapter 2.

Trying to visualize the signal with a regular plotting tool is not possible, since having the whole signal in memory will lead to memory errors, as already discussed. Another limiting factor is the number of samples, which is simply too much to fit in a computer screen. The solution found in [16] was to provide a global view of the signal, which

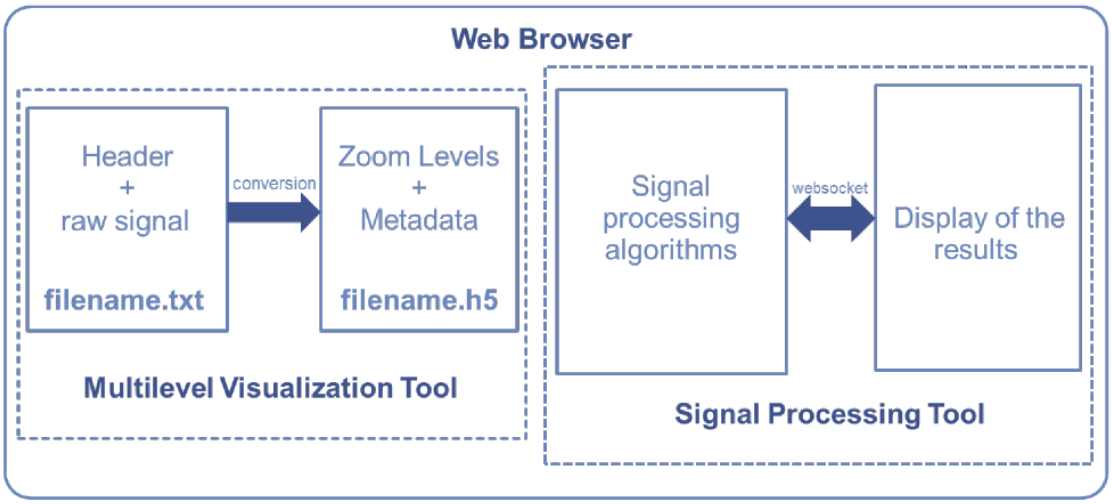


Figure 4.2: Programming architecture of the signal processing tool.

would vary in detail depending on the time length one would wish to visualize. The keys '+' and '-' allow zoom in and zoom out respectively. An example of a 7 hour long ECG record is show in figure 4.3.

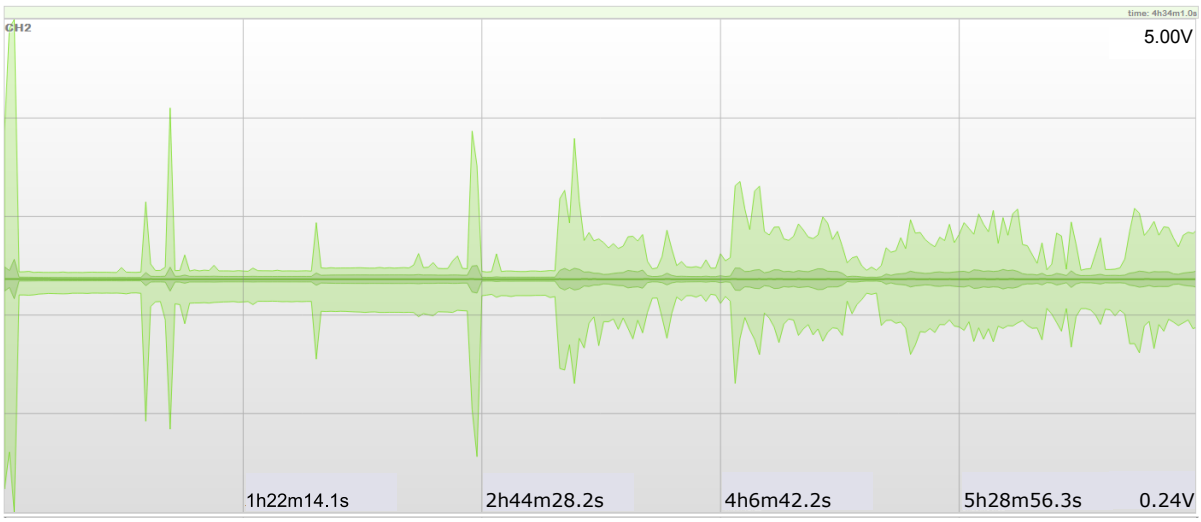


Figure 4.3: Multilevel visualization of a 7 hour long ECG

The tool runs in a web browser, rendering it extremely portable and giving it the potential to be accessed from anywhere with an internet connection. The signals can ultimately be stored in a server and accessed remotely from anywhere through the application and the signal processing can happen remotely.

4.3.2 Newly developed work

The signal processing modules were integrated in the application as depicted in figure 4.2. The user is able to interact with the displayed signal through a web-based interface,

originating websocket requests. These requests typically include:

- The name and directory of the signal file;
- The channel of the signal (the records contain several types of signals, each one from a different channel in the recording device);
- The initial and final time of the signal slice the user is currently viewing;
- The name of a function (a processing function), which is associated to the action the user triggered on the browser.

All this is hidden from the user and is already associated to interactive elements such as buttons or dragging elements. Any further information related to the signal, which might become necessary in the signal processing algorithms is obtained by accessing the file and reading the corresponding metadata (hdf5 attributes).

So, for example, there might be a button that "computed" the mean of a portion of the signal. The user would press that button, and automatically the information described above would be sent through the websocket and be the input of a *mean* algorithm (which might be implemented in parallel). The result would then be sent back through the websocket to be displayed to the user in the browser.

The processing result (such as plot or parameters) are displayed next to the visualization of the signal and allow a direct interaction with the signal, as will be explained next.

4.4 Heart rate variability analysis tool

All heart rate variability analysis algorithms are based on the RR intervals. Other than the spacing between all QRS complexes, the values the ECG assumes are completely meaningless in HRV analysis.

A signal lasting 7 hours and sampled at 1 kHz has 25.2 millions of samples. However, the number of RR intervals in the signal is relatively small, and independent from the sampling frequency. Considering a mean heart rate of 60 beats per minute, the number of RR intervals would be 25199, about 1000 times smaller than the length of the signal.

If the user selects the first three hours of the signal, and wishes to see the tachogram (RR evolution in time) those three hours of the signal would have to be processed, starting at the detection of the R peaks. Only then could the HRV processing algorithms be applied.

Processing the ECG this way, every time the user makes a request, is a tedious process (see chapter 5 for the detailed execution times) and not at all practical.

The alternative to this method of signal analysis was to define the peak detection as a preprocessing step which would always take place before the HRV analysis *per se*.

The ECG peak detection step is always the first step of the analysis and is applied over the whole signal, regardless of the time slice the user wishes to view. This process only has to be done once. The results are stored in a *.h5 file, and are accessed every time the user makes a request related to the associated ECG record. This introduces an interesting concept: storing meaningful processing results which are necessary for further analysis, to spare time in future analysis sessions. Now, if the user selects a 3 hour time slice, the *.h5 file containing the instants of the different R peaks is accessed and the peaks corresponding to the selected time slice are extracted right away to be used by HRV analysis algorithms. This process is illustrated in figure 4.4.

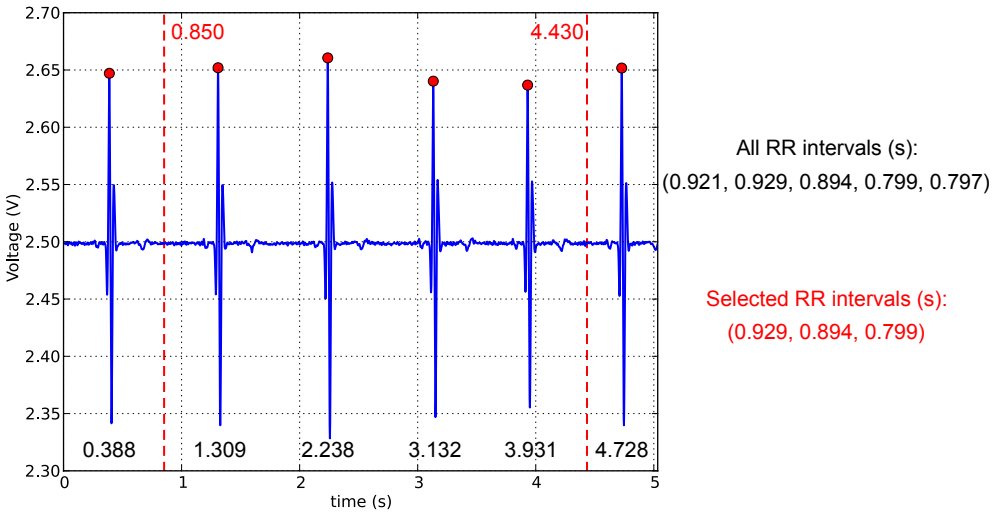


Figure 4.4: The user only wishes to analyse the portion of the signal between the dashed lines (0.85 s to 4.43 s). Of the 6 total peaks, only 4 of them are retrieved and 3 RR intervals are computed (out of the total 5).

The red dots in figure 4.4 are the detections of the ECG peak detection algorithm. For a matter of simplicity, only a small signal is shown, but the same principle applies to any other signal.

This preprocessing step is executed in parallel and its execution times are discussed in chapter 5. By waiting a few minutes for the preprocessing step to complete, a huge amount of time is saved in the future. The heart rate variability tool provides analysis results in just a few seconds (less than 10 seconds) and can in this sense be classified as a real-time analysis tool.

Initially the user is only shown a button associated to the peak detection algorithm. By pressing this button the user is asked for the ECG channel and the algorithm will run for a few minutes. When the execution is over, the screen is refreshed and an indication of the places where the signal processing results will appear is displayed.

In a future session, when the user loads the signal, the program looks for the processing results file and the peak detection is skipped.

An overall view of the integration of the processing module in the visualization tool is illustrated by figure 4.5

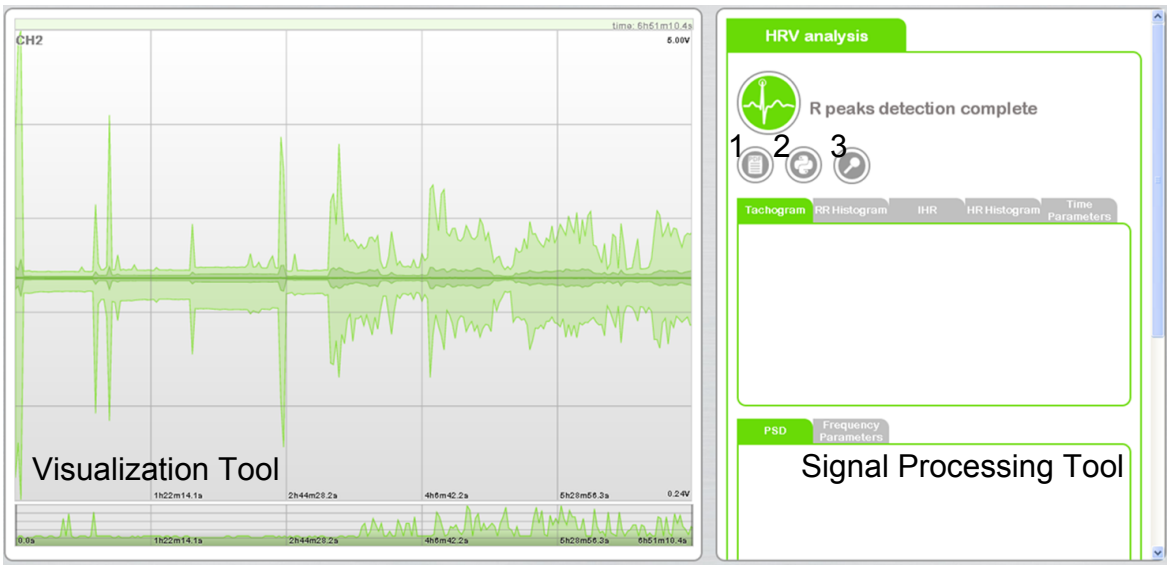


Figure 4.5: 1. Report Generation; 2. Programming mode analysis; 3. On-screen analysis.

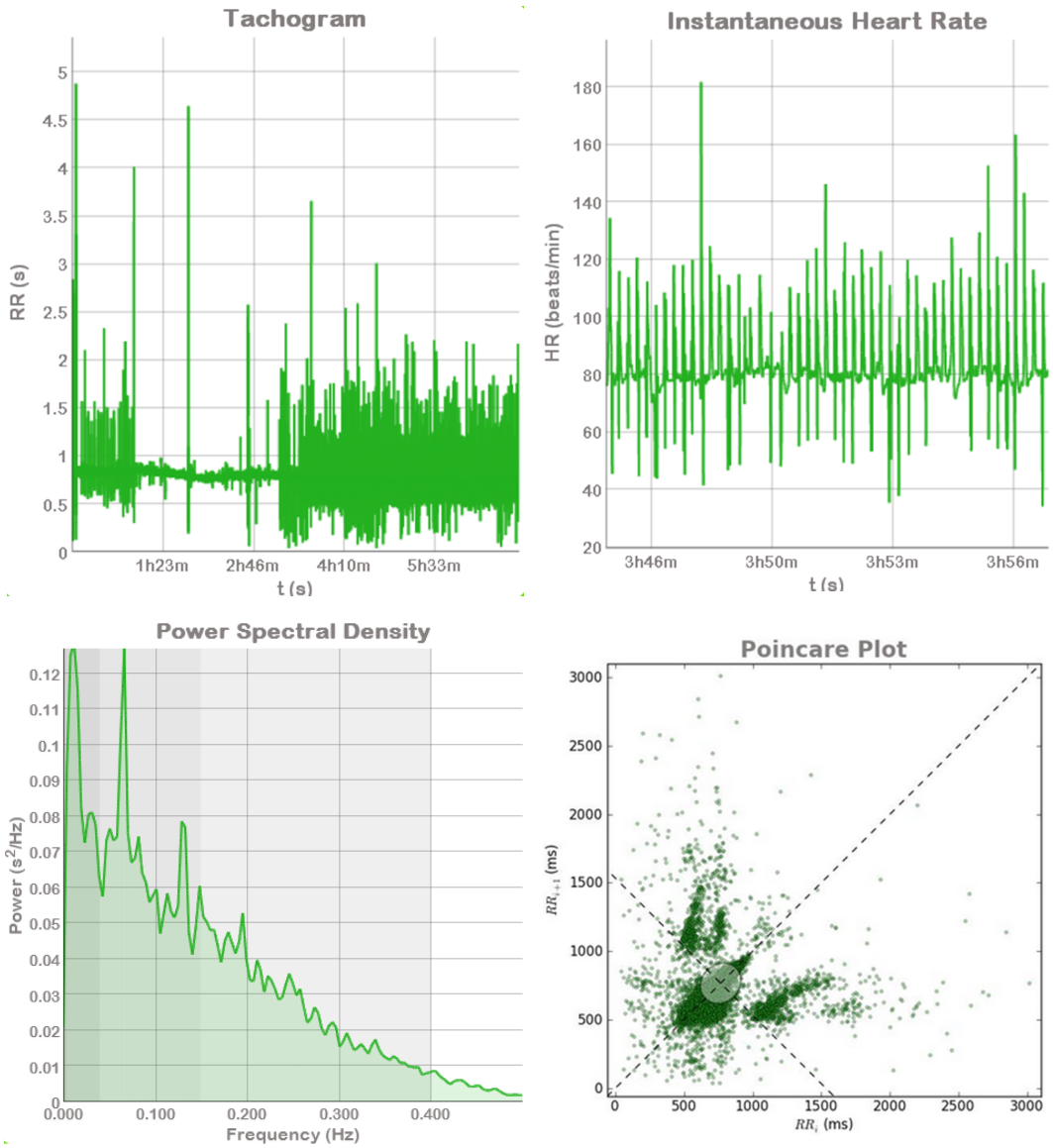
The signal processing tool has 3 main functionalities: the on-screen analysis, a report generation feature and a programming mode analysis.

4.4.1 On-screen analysis

The on-screen analysis, as the name suggests, is displayed in the browser. After all the HRV analysis algorithms execute, the results are sent through the websocket and properly displayed. The results may be visual representations (plots) or tables with parameters relevant to the analysis. The on-screen analysis was organized in 3 different groups: (linear) time domain analysis, frequency domain analysis and non-linear analysis. In figure 4.5 only the container elements of the time and frequency domain analysis are displayed. The grey tabs allow the user to switch between the different results easily.

The on-screen analysis results of a 7 hours long ECG are illustrated in figure 4.6. All the parameters and visual representations were computed according to [21].

An interesting functionality that allows a thorough inspection of the signals is the synchronization of some visual representations with the visualization tool. As mentioned, it is possible to zoom in and zoom out and to select specific time slices to visualize the signals. Representations such as the tachogram or the instantaneous heart rate, have a direct temporal relation to the signal, i.e, each point of these representations can be directly associated to a specific sample in the ECG record. With this in mind, it was made possible to drag and drop in these plots, and automatically the zooming feature of the



Statistical Parameters	
Mean RR (ms)	766.28
Min RR (ms)	39.00
Max RR (ms)	4882.00
SDNN (ms)	147.04
Mean HR (1/min)	81.43
SDHR (1/min)	26.92
Mean HR (1/s)	1.36
Mean HR (1/5s)	6.79
Mean HR (1/10s)	13.57
rMSSD (ms)	192.87
pRR50 (%)	16.71

Geometrical Parameters	
TINN (ms)	4843.00
HRV tringular index	16.52

	Peak (Hz)	Power (ms^2)	Power (%)	Power (n.u.)
ULF	0.003	171.68	0.99	
VLF	0.012	3410.23	19.69	
LF	0.066	7228.21	41.74	51.98
HF	0.150	6506.71	37.57	46.79

LF/HF	1.1109

SD1 (ms)	102.8468
SD2 (ms)	141.9197
SD1/SD2	0.7247

SD1 (ms)	136.3834
SD2 (ms)	156.9718
SD1/SD2	0.8688

Figure 4.6: The most important parameters and visual representations provided by the on-screen HRV analysis.

signal (from a previous work) would be updated. This means that the tachogram and instantaneous heart rate, when zoomed by the user, activate the signal zoom so that there is synchronization at all the times. If some strange value of instantaneous heart rate is observed (let us say, an instantaneous heart rate of 200 bpm), the user can quickly drag and drop around that instant and the signal will be zoomed in, displaying the corresponding waveform. Then the clinician might discard that value, if it is clear that it comes from an error in the peak detection algorithm, or he can confirm its pathological nature by observing the correct QRS spacing of that time interval. This synchronization is also extremely useful in quickly identifying regions of the signal with a lot of noise, since there is the knowledge of the physiologically admissible instantaneous heart rate values. Any value outside these boundaries is quickly identified, and by inspecting the signal it can be classified immediately as correct (pathological or not), algorithm error or noise in the signal record.

The HRV analysis algorithms for this part of the tool did not need to undergo any parallelization process because the whole execution only took a few seconds. Considering the duration of the signal, the execution time was very good.

For more information regarding the HRV analysis provided by this tool, see appendix B.

4.4.2 Report generation

The report generation feature has the functionality of generating a *.pdf file (a report) containing the analysis results of the signal. This way the clinician can carry the analysis results with him and make his own annotations and comments.

The report includes a global analysis, in which the parameters and visual representations of the previous section are presented. It also illustrates the evolution of those parameters in time. The signal is divided into 15 equally sized parts and the parameters appearing in figure 4.6 are calculated for each of those parts in parallel. This number of points was chosen arbitrarily and can be easily increased or decreased. More points will yield better looking representations, but each point will weigh less, since it is relative to a smaller time interval.

The report also includes hour-by-hour evolutions of the tachogram and the instantaneous heart rate.

It takes about 15 to 20 seconds for the report to be generated and opened automatically.

It is also important to mention that various versions of the report were developed regarding the number of pages presented. For signals that are too small, the global representation (2 pages) suffices. For long-term signals the report will be 6 pages long.

The first page of the report, the hour-by-hour evolution of the instantaneous heart rate and the parameters evolution representations from a night record are illustrated in figure 4.7. For further information see appendix B.

Figure 4.7: Four pages of a HRV analysis report from a night record.

4.4.3 Analysis in programming mode

The analysis in programming mode is a powerful and very useful feature, implemented specifically for the researcher or the programmer. As the application, it also runs in the web browser [49]. This way, additional processing can be made using scientific programming libraries [50]. Several important variables were loaded into the namespace, along with a "help text" describing the currently loaded variables. The loaded variables were: the sampling frequency of the signal, an array with the samples of all the R peaks, and the beginning and ending of the selected time slice, in samples. This information allows a personalized user analysis.

4.5 Respiratory analysis tool

The respiratory analysis tool is very similar in nature to the HRV analysis tool. The preprocessing step of detecting the respiratory cycles is much faster than the ECG peaks detections. This happens because of the simpler waveform of the respiratory signal, compared with the ECG. As will be seen in the next chapter, if the tool was executed in a machine with 7 or 8 processors, the execution would be so fast that the preprocessing step would be almost unnecessary. Moreover, the respiratory analysis algorithms used in this tool are simple statistic measures which are not computationally intensive and so the whole processing happens a lot faster than on the HRV example. The choice of not implementing a parallel version of the respiratory cycles detection algorithm would have been valid, taking into account the speed of the whole computation. However, by implementing the preprocessing in parallel, the tool is assured to also execute fast on machines with less powerful processors.

The cycles detection algorithm is a simple threshold-base algorithm which uses as threshold the root mean square (RMS) of the signal. The algorithm was modified to use an adaptive threshold, with the RMS of the signal being computed every 20 seconds when analysing long-term biosignals. Figure 4.8 illustrates the concept of adaptive threshold.

For the sake of simplicity of visualization, in the small signal of figure 4.8 the threshold is changed every 5 seconds. The threshold changing makes it very hard to miss detections because of baseline fluctuations, as would occur if the threshold did not intersect the respiratory cycle.

The features of the respiratory analysis tool are very similar to the HRV analysis tool. The on-screen analysis is extremely simple. The analysis is composed by (figure 4.9):

- A visual representation of the evolution of the instantaneous breath frequency through time. The breath frequency is computed in a similar way to the instantaneous heart rate: by dividing 60 by the derivative of the respiratory cycles array, in seconds;
- A visual representation of the evolution of the instantaneous breath amplitude through time. The amplitudes are given in normalized units, so that they do not

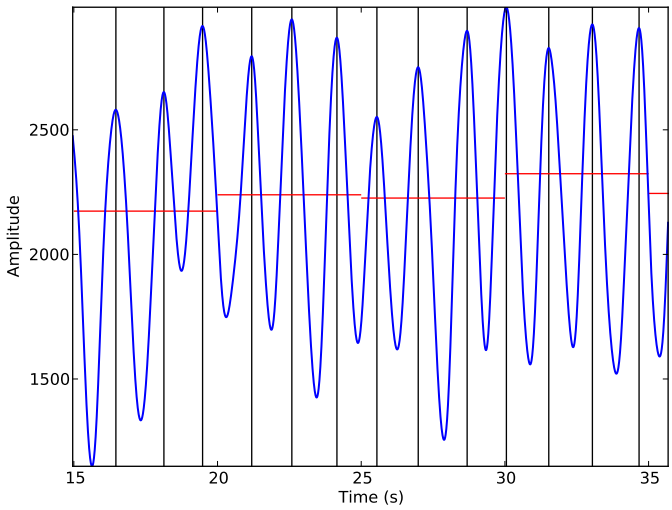


Figure 4.8: Detection of respiratory cycles using an adaptive threshold, which changes every 5 seconds

depend on the acquisition system and the results can be compared with the analysis of signals from other devices.

- A breath frequency histogram;
- A breath amplitude histogram;
- A table with the basic breath frequency and breath amplitude statistics: minimum, maximum, standard deviation and mean values.

In the example signal portion analysed in figure 4.9 the relevance of the breath amplitude histogram becomes quite clear. The signal has a notable respiratory cycle, with an amplitude that is much greater than the rest of the respiratory cycles. This indicates that the patient at that instant experienced a very deep breath.

The result is a high histogram dispersion, with that respiratory cycle corresponding to the single count associated to the maximum amplitude. The breath amplitude evolution plot also makes that outlier quite evident.

It is unclear whether the breath amplitude analysis will be relevant for comparison of respiratory events over long periods of time, since the baseline fluctuations have influence on the results.

Again, as with the HRV analysis tool, the user can simply drag and drop over the instantaneous breath frequency and instantaneous breath amplitude plots and almost instantaneously the plot and the visual representation of the respiratory signal will be in sync.

The *.pdf report is very similar to the HRV analysis report and will not be shown. It suffices to say that it includes global parameters and visual representations, hour-by-hour

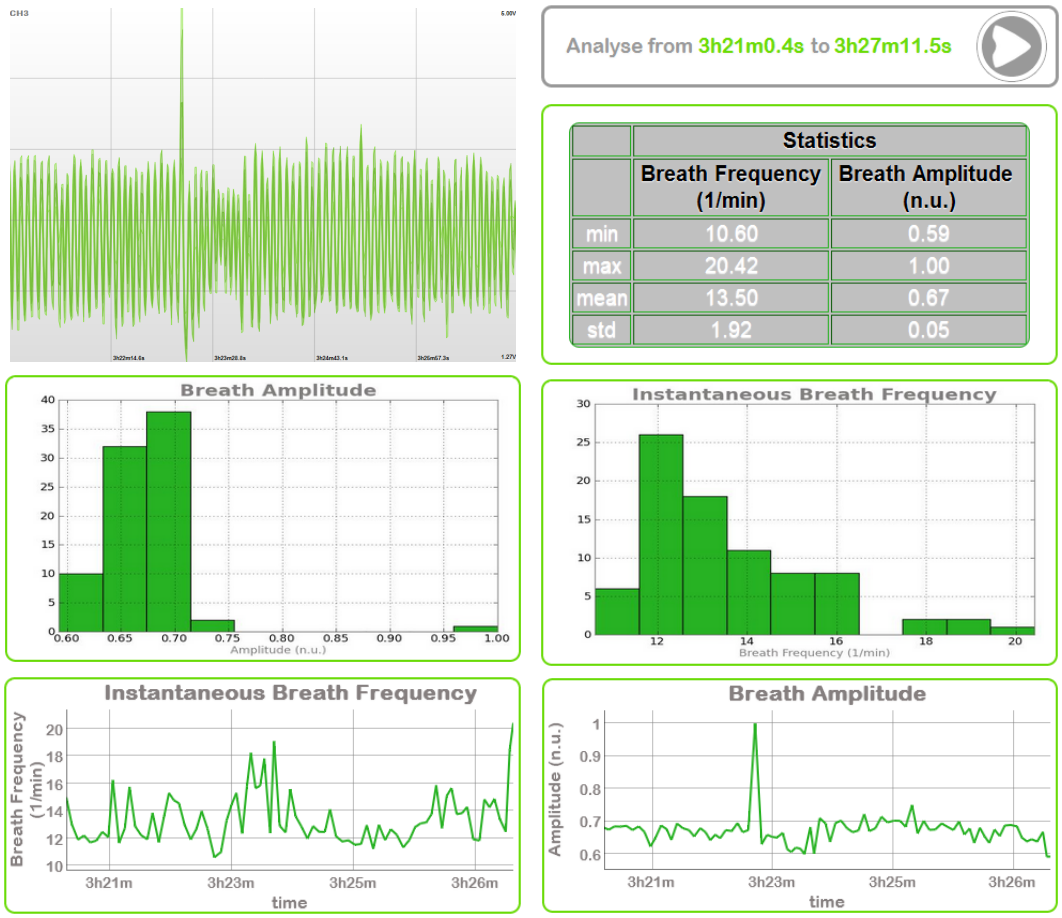


Figure 4.9: On-screen analysis of a portion of a respiratory signal.

evolution of the instantaneous breath frequency and instantaneous breath amplitude, and the evolution of the respiratory statistical parameters (15 points as in the previous section).

4.6 Concluding remarks

After the preprocessing step, every time the user intends to analyse a signal slice, the *.h5 file with the processing results is accessed. This access is extremely fast since the file format allows random access to portions of the stored datasets.

The fact that this access is so fast, even in *.h5 files with datasets containing millions of samples, allows the use of this file format in another interesting way. It is common (especially when dealing with long-term records) that the return value of an algorithm is small enough to fit in the memory without any problem, but there are algorithm which use too much memory and a memory error is raised before the execution finishes.

Very often the application of algorithms to process long-term signals involves the manipulation of large data structures with a huge number of entries (clustering algorithms,

for examples, typically manipulate huge matrices). If the algorithm does not finish executing when applied to long-term records because of memory errors, a good idea is to store the values in a *.h5 file. This file would then represent an extension of the RAM, in the same way the operating system uses what is called virtual memory [33] (when the system runs low on memory the operating system allocates disk space temporarily and uses it as memory). This virtual memory under the form of a *.h5 file may just be the solution to solve such problems.

Regarding the parallelization of the parameters evolution, used in both tools, there were 2 ways that could be followed in order to achieve it (in this problem, a task is considered to be the computation of all parameters for $1/15^{th}$ of the signal - there are 15 tasks):

- The data parallel approach, designing the algorithms in the way explained in chapter 3 and using all of the available processing units to compute a single task faster;
- A simple higher level task parallelism in which each task would be computed by a single processing unit.

The option adopted in the signal processing tools was the second one. It typically pays off to parallelize tasks at the highest possible level. This means that if the same problem has to be solved several times, it usually is not a good idea to parallelize the sequential algorithm.

Using the first option would involve to design parallel algorithms, and it would take more computations for a task to complete, and therefore more time.

The problem that we are trying to solve is composed of 15 tasks. Let us assume that each task takes a unit of time to be executed in a single processor.

If there were 15 identical processing units, all of them computing a single task at the same time, the problem would take exactly 1 unit of time to complete.

On the other hand, applying a lower level parallelization by means of designing parallel algorithms would yield inferior results in this situation. If every single task was tackled by all 15 processors, it would take more than $1/15$ units of time, because the parallel algorithm involves more computations than the serial one: the data has to be distributed among all the processing units and in the end the results have to be merged in some way (a typical *MapReduce* computation). The overhead of communications would also be bigger in the first case and would be a speedup limiting factor - the data would have to be distributed and retrieved from the processing units 15 times, whereas in the second case it would only happen once.

Note that if the number of available processing units is much greater than the number of tasks the lower level parallelization might be best. For the same problem, if there were 1000 processing units and still 15 tasks, the *MapReduce* approach would be undoubtedly better.

Finally, it is important to mention the memory usage during the process of displaying the results. The browser provides an option which shows the heap memory distribution

between the different elements of the processing tool. The heap is a section of the memory where all the variables that are created and initialized at runtime are stored [33]. This is the case with the analysis results, which are created at runtime.

Before any analysis takes place, the visualization tool was using 3.19 MB of RAM. Analysing the heart rate variability of a 7 hour long record increased the allocated memory to 14.30 MB; the respiratory analysis of a 7 hour long record increased the allocated memory to 6.74 MB.

As expected, the more lightweight nature of the respiratory analysis (it has less features) results in a lower memory consumption. The objects that were consuming the most memory were JavaScript arrays, which were associated with the visual representations that provided zooming capabilities. Elements which are not interactive such as histograms were simply included as images in the application and use fewer memory.

This method of visualizing the signal processing results proved to be efficient for the type of analysis made and the memory consumption was very small considering the amount of RAM of nowadays computers. This method was only possible because the analysis results of the huge sized long-term records were fairly small. New ways of displaying the analysis results would have to be thought of, if a different signal processing tool, whose results were a lot bigger in size, was to be implemented. In this situation if the memory errors associated with the signal processing algorithms were overcame, the memory errors associated with the proper display of the results in the browser would still have to be dealt with.

5

Performance Evaluation

In this chapter, the performance evaluation of two different algorithms related to the case studies previously mentioned will be made.

The algorithms are both peak detectors. In the case of the heart rate variability analysis tool the algorithm is a ECG peak detector and in the case of the respiratory analysis tool the algorithm is a respiratory cycles detector. The algorithms are therefore very similar in their functions and degree of parallelization. However, as it will be seen, their execution times are quite different, which leads to slightly different (yet illustrative) results regarding their run times in a computer with several processors.

5.1 Overview

The algorithms were executed remotely using a virtual machine from *Amazon Web Services* [51]. The specific service utilized for this purpose was the *Amazon Elastic Compute Cloud* (Amazon EC2), which allows the allocation of resizable computing resources. In Amazon EC2 there are several instance types, according to their storage capacity, memory, CPU and graphics processing unit (GPU). A *c1.xlarge* instance was utilized for performance evaluation purposes. This instance is classified as a "High-CPU Extra Large Instance" and was chosen because of its 8 virtual cores, a number of cores that is sufficient for the kind of evaluation made in this chapter. A more detailed view of the virtual machine features is depicted in table 5.1.

The operating system running on the virtual machine is a terminal-based Ubuntu. By not providing a GUI, a higher performance can be achieved because there are less processes running and there is a lower memory usage as well when the system is idle.

Instance type	c1.xlarge
CPU	8 cores with 2.3 GHz each
Model name	Intel(R) Xeon(R) E5506
Cache size	4096 kB
RAM	7 GB
Storage	1690 GB

Table 5.1: AWS virtual machine features

The connection to the machine was made via *secure shell* (SSH). Because SSH uses port 22, a firewall rule had to be added in the virtual machine: port 22 was opened for incoming connections.

Before being able to start the execution of the algorithms, the signals to analyse had to be transferred to the remote machine. The signal files and the code including the algorithms were transferred via *SSH File Transfer Protocol* (SFTP), using also port 22.

Both the respiratory and the ECG signal records last approximately 7 hours. The signals were acquired at 1000 Hz, having approximately 25 million samples, each.

The code was written such that each algorithm was run 8 times over each signal, using a different number of CPUs each time. This process was repeated 3 times for each algorithm, so that there could be 3 execution times for each algorithm, for each number of cores.

The total time of analysed record (in both cases) is more than the duration of the signal. This happens because the division of the signals is accompanied by overlapping regions, to make up for the poor algorithm performance in the edges of the records. Let S be the overall signal size analysed by the algorithms, $nsamples$ the number of samples of each record, $njobs$ the number of divisions made and ov the overlap size, in samples. Then, taking into account the overlap influence, the overall signal size can be computed according to equation 5.1.

$$S = nsamples + ov.(njobs - 1) \tag{5.1}$$

The overall size of the analysed signal is proportional to both the overlap size and the number of jobs to distribute among the workers (in this shared memory architecture, each core is a worker). The overlap is chosen according to the algorithm and it is assumed fixed in size. As each part grows in size, the overlap influence becomes less and less.

In a long-duration signal the overlap influence is typically negligible, since each part is much bigger than the overlap.

It is tempting to choose a number of jobs (or divisions) equal to the number of available workers. This is the best solution if there is enough RAM to tackle the problem this way. In a distributed memory architecture this means that each node must have enough RAM for being able to process $1/n^{th}$ of the signal at once if there are n computing nodes; in a shared memory environment, since the signal would all be processed at once, there should be enough memory to process the whole signal. This approach may be feasible

depending on the duration of the record and on the memory that the signal processing algorithms utilize.

Very soon in the course of this work the processing of long-duration records originated memory errors, and therefore the signal division was made "statically", i.e., each part has a fixed length, and the number of parts varies with the length of the signal. This is not the best way to divide the signals. Ultimately, the best division method must depend on the computing architecture: the number of workers, available RAM and the overhead of communications.

A tool for automatically determining the optimal division according to the computing architecture utilized (and the operating system running) might be a great performance tuning tool. However, that study was not done in this work.

Two very useful concepts to analyse the performance of a parallel algorithm are now defined: the speedup factor and efficiency.

The speedup factor is a measure of relative performance between a multiprocessor system and a single processor system and is defined as [38]:

$$S(p) = \frac{t_s}{t_p} \tag{5.2}$$

where t_s is the execution time on a single processor and t_p is the execution time using p processors. The speedup factor gives the increase in speed in using a multiprocessor.

Efficiency is the ratio of speedup to the number of processors used [37]. It is defined as [38]:

$$E = \frac{t_s}{p.t_p} \tag{5.3}$$

The efficiency is a number between 0 and 1 and represents the fraction of time that the processors are being used on the computation. If $E = 50\%$, the processors are being used only half the time, on average, on the actual computation [38].

5.2 ECG peak detector

The algorithm for ECG peak detection is a Pan and Tompkins derived algorithm [52] that was previously implemented. The data parallel implementation of this algorithm was made using 10 minute parts and overlapping regions of 400 ms. The overlap was determined empirically after dividing small signals (that could be analysed at once) and comparing the results obtained considering different overlap sizes. The long-duration record analysed lasted 7 hours and 17 minutes, originating 44 jobs (the last job was slightly shorter in length). The results of the signal processing on the remote virtual machine of table 5.1 are illustrated in figure 5.1

The code was executed three times, each one being represented by a different line in figure 5.1. The speedup is evident. Another interesting feature of figure 5.1 is the results

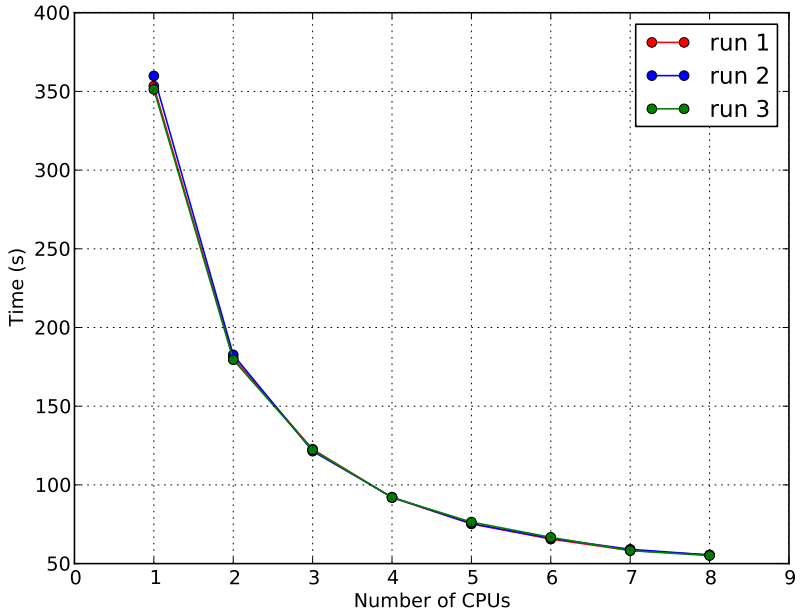


Figure 5.1: Execution times of the ECG peak detection algorithm in a 8 core remote virtual machine.

similarity over the 3 executions. Despite this similarity being expected, it is interesting to mention that such results were never obtained on multicore desktop computers in the course of this work - there were always significant execution time variations, which might be explained by variable resource consuming background processes and the GUI.

In the current situation, a multiple core shared memory environment, let p be the number of available processors and t_s be the execution time with a single processor. Then, the total execution time is simply:

$$t_p = \frac{t_s}{p} \tag{5.4}$$

Equation 5.4 only holds assuming a 100% degree of data parallelism and that each part has the same size (and no overhead). In this situation there is a linear speedup.

The expected execution time and the mean execution times of the 3 executions can be seen in figure 5.2. The red line was drawn simply by computing the mean of the three executions; the first point of the dashed line (1 CPU) is the mean of the three executions - the serial time -, and the rest of the expected values were computed varying the number of CPUs in equation 5.4.

The overall results depicted in figure 5.2 are quite satisfactory since there is only a slight separation between the mean execution time and the expected execution time. The separation between the two lines increases with the number of CPUs, i.e., as the number of CPUs increases the speedup factor, despite increasing, becomes farther and farther from the expected. The execution mean times of figure 5.2, the speedup factors and the

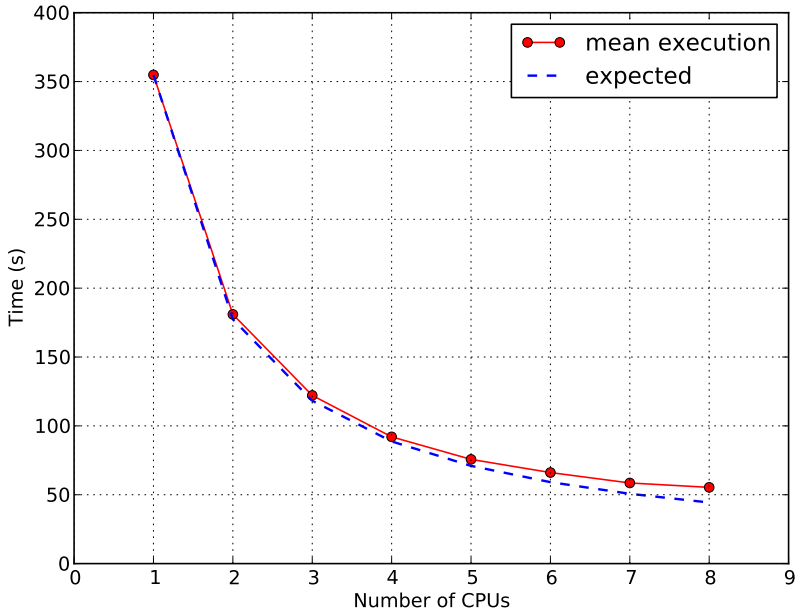


Figure 5.2: ECG peak detection algorithm mean execution time and the expected execution time.

efficiencies are shown in table 5.2.

In order to explain the reason why the efficiency is less than 1, several factors have to be accounted for:

- There are parts of the computation in which not all the processors are performing useful work (serial parts of the computation - see equation 2.1);
- The parts may not all take the same time to process, i.e., the number of computations varies according to the signal values (the tasks were all different);
- Despite the existing load balancing (see appendix A) each task does not take the same time to process (and the number of tasks is not even multiple of the number

CPUs	Mean execution time (s)	Expected speedup	Real speedup	Efficiency
1	354.83	1	1	1
2	180.96	2	1.96	0.98
3	122.12	3	2.91	0.97
4	92.03	4	3.86	0.97
5	75.68	5	4.69	0.94
6	66.11	6	5.37	0.90
7	58.66	7	6.05	0.86
8	55.36	8	6.41	0.80

Table 5.2: Performance analysis of the ECG peaks detection algorithm.

of processors in some situations) and it is inevitable that not all the processors have the same overall load.

5.3 Respiratory cycles detector

The algorithm for the detection of respiratory cycles is a simple threshold-based algorithm, using the root mean square (RMS) of the signal as threshold. Some adaptations were made, to account for likely baseline fluctuations in long-duration records. Namely, the RMS of the signal is computed frequently. In this sense, the respiratory cycles detection can be viewed as an adaptive threshold algorithm.

This algorithm is by far less computationally intensive than the one mentioned in the previous section. Therefore the execution times are much smaller than the ECG peak detection ones.

The algorithm was run 3 times over a respiratory signal of 6 hours and 51 minutes in a similar way to the previous section. The signal also had a sampling frequency of 1000 Hz.

The signal was divided into 20 minute parts (this algorithm requires far less memory than the ECG peak detector, which was run over 10 minute parts), with 400 ms overlaps.

The signal processing results in the remote machine for this algorithm are shown in figure 5.3.

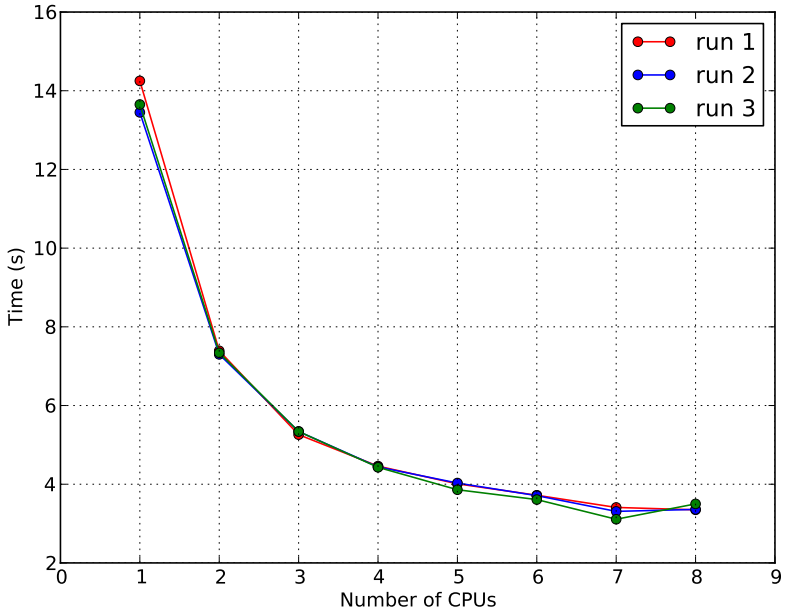


Figure 5.3: Execution times of the respiratory cycles detection algorithm in a 8 core remote virtual machine.

The total execution time is very small, and each task is processed in less than a second. In this situation there are differences, albeit small, between the 3 executions with each

processor.

The mean and expected execution times are illustrated by figure 5.4 and the speedup factor and efficiency measures can be seen in table 5.3.

The speedup factors and efficiency measures of table 5.3 are clarifying. The results are clearly inferior, performance-wise, to those in the previous section.

The execution time variability might be due to the task attribution by the parallel scheduler to the different processors. This kind of communication overhead happens when each task is so small that before distributing evenly all the tasks among the different processors, a processor might have finished its task already. Or, to put it simpler, the execution time of each task is comparable to the time it takes for the scheduler to distribute all the tasks. Because of this, there is a probability that a processor may be idle at a given time, even when there are more tasks to compute (there is not a real load-balancing as would occur if the tasks were bigger).

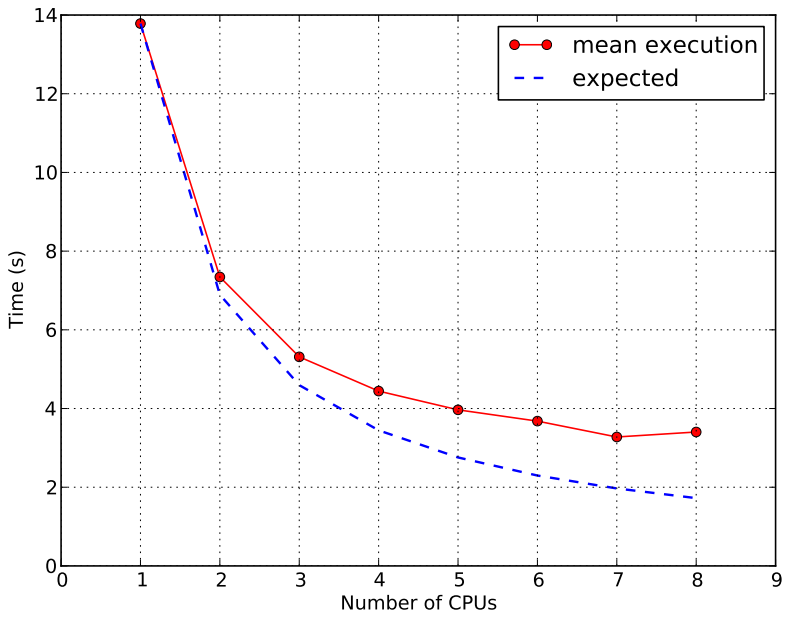


Figure 5.4: Mean execution times of the respiratory cycles detection algorithm.

An alternative explanation for the reduced efficiency and speedup factors is provided by Amdahl's law (equation 2.1). Since the total execution time is fairly small (comparing with the previous section), the serial parts of the computation may present a large fraction of the overall computing time, becoming a significant speedup limiting factor. In this situation, the results would evidence that the speedup limit has been reached. If so, the non-existent speedup increase when the program is executed with 8 processors, rather than 7 (the speedup actually decreases) could be explained by an unsteady CPU usage (see figure 2.7). Since the execution time is approaching its lower limit, any execution

CPUs	Mean execution time (s)	Expected speedup	Real speedup	Efficiency
1	13.78	1	1	1
2	7.34	2	1.88	0.94
3	5.31	3	2.60	0.87
4	4.33	4	3.18	0.80
5	3.97	5	3.47	0.69
6	3.68	6	3.74	0.62
7	3.28	7	4.20	0.60
8	3.40	8	4.05	0.51

Table 5.3: Performance analysis of the respiratory cycles detection algorithm.

time reduction resulting of a larger number of CPUs would be residual. In this situation, the non-constant CPU usage when the system is idle (when no user programs are running) becomes fundamental and may explain the 0.12 s difference between execution times. In other words, the time difference between executions is so small that is affected by CPU usage fluctuations caused by the operating system itself.

Despite this 0.12 s increase in the processing time when using 8 processors comparing to the processing time using only 7 processors, the overall speedup is unequivocal.

6

Conclusions

This chapter summarizes the results accomplished during this work. Ways of continuing this work are described so that the main ideas here presented can be given good use and help fulfil their full potential.

6.1 Accomplishments

The main goal of this thesis was to apply parallel programming techniques in the field of biomedical signal processing. The need for the design of parallel algorithms arose from the object of the processing techniques: extremely large records, which could not be analysed by conventional techniques. Another goal was to make the developed algorithms available to the non-expert in programming, namely a clinician or a medical researcher.

A parallel programming model very similar to *MapReduce* was presented, which attempts to explain a general way of designing parallel algorithms that may not be embarrassingly parallel.

To address both of the proposed goals, two signal processing tools were developed, each one employing data-parallel programming techniques: a HRV analysis tool and a respiratory analysis tool. These tools provide a user-friendly interface and hide all the programming from the user, which does not have to know how to code in order to use all the implemented features and explore the medical records thoroughly.

The tools integrate a set of algorithms specifically meant for the analysis of ECG and respiratory records. Besides on-screen biosignal analysis, there is also the possibility of generating a *.pdf report wich summarizes all of the analysis results. Such document makes the processing results portable and promptly accessible to medical personnel, having the potential of being an extension of the patients medical file.

The tools were tested on long-term records from ALS patients. These patients have cardiorespiratory problems and must be monitored continuously, thus originating representative test data.

The performance analysis of two events detection algorithms (executed in the pre-processing steps of each tool), a ECG QRS detector and a respiratory cycles detector, was made and the results were very satisfactory overall, considering that the analysis was performed over 7 hour long records.

The concept of remote processing was used to assess the run times of the parallel algorithms using different number of processing units. Processing the signals remotely has a huge potential, since more computing resources become available than those in a typical personal computer or laptop computer. This approach allows the exploration of the scalability of the parallel algorithms, allowing an increase in the number of processors until the speedup factor reaches its limit, as explained in chapter 5.

6.2 Future work

This work is not in itself a finished product. The topics approached can be further explored and new biosignal processing tools can be created.

Both the HRV and respiratory analysis tools are not definitive, since new works are published every day and new ways of analysing the ECG and respiratory records are presented frequently. In this sense, such tools may never really be considered definitive and new analysis features must be added/removed as time passes by.

The portability of these signal processing tools also makes them suitable for the analysis of tailor-made experiments, i.e., experiments with highly specific protocols which may require a combination of analysis algorithms. This would allow the extraction of significant features that would help to relate several biosignals, in a cause-and-effect kind of way.

A multivariate analysis where the respiratory analysis and the HRV analysis are not made separately, but together, in order to relate the cardiac and the respiratory systems is not new [53, 54], and can be easily achieved with this kind of tools.

A new biosignal processing tool that automatizes the process of running algorithms in parallel would be a great feature to add to the existing tool. This tool would not be directed at the analysis of any particular record, and instead of "taking" a signal record as input would take algorithms as inputs.

The algorithmic basis of such tool was written during this work, but unfortunately the interface was not developed. This tool would avoid the need to manually design data-parallel algorithms by hand (which was done during the pre-processing steps of the developed tools). The user would only have to specify an algorithm and a signal (in a web-based interface) and all the details concerning parallelism would be dealt with without any concern for him.

The design of these signal processing tools and the underlying need to understand a

previously developed work, in order to extend it and help it fulfil its potential was very gratifying and was accompanied by a remarkable learning process. The opportunity to contribute to the growing demand for fast signal processing tools that are able to deal with huge amounts of data efficiently was an enriching and gratifying experience.

Bibliography

[1] J.D. Enderle and J.D. Bronzino. *Introduction to biomedical engineering*. Academic Pr, 2011.

[2] J.D. Bronzino. *The biomedical engineering handbook*, volume 2. CRC Pr I Llc, 2000.

[3] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, December 2011.

[4] PLUX (2012. http://wicardioresp.plux.info. Accessed: 25/05/2012.

[5] Z. Yang, Y. Zhu, and Y. Pu. Parallel image processing based on cuda. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 3, pages 198–201. Ieee, 2008.

[6] E.I. Konstantinidis, C.A. Frantzidis, L. Tzimkas, C. Pappas, and P.D. Bamidis. Accelerating biomedical signal processing algorithms with parallel programming on graphic processor units. In *Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on*, pages 1–4. IEEE, 2009.

[7] D.F. Sittig and K.H. Cheung. A parallel implementation of a multi-state kalman filtering algorithm to detect ecg arrhythmias. *International journal of clinical monitoring and computing*, 9(1):13–22, 1992.

[8] TH Sander. Parallel implementation of temporal decorrelation independent component analysis. In *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*, pages 1783–1785. Springer, 2010.

[9] A. Ben Abdallah, Y. Haga, and K. Kuroda. An efficient algorithm and embedded multicore implementation of ecg analysis in multi-lead electrocardiogram records. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 99–103. IEEE, 2010.

[10] R-R Interval Processing with the BIOPAC HRV Algorithm. http://www.biopac.com/r-r-interval-processing-hrv-algorithm. Accessed: 22/05/2012.

[11] HRVAS: HRV Analysis Software. http://sourceforge.net/projects/hrvas/. Accessed: 22/05/2012.

[12] A. Travaglini, C. Lamberti, J. DeBie, and M. Ferri. Respiratory signal derived from eight-lead ecg. In *Computers in Cardiology 1998*, pages 65–68. IEEE, 1998.

[13] P. Langley, E.J. Bowers, and A. Murray. Principal component analysis as a tool for analyzing beat-to-beat changes in ecg features: application to ecg-derived respiration. *Biomedical Engineering, IEEE Transactions on*, 57(4):821–829, 2010.

[14] M. Campolo, D. Labate, F. La Foresta, FC Morabito, A. Lay-Ekuakille, and P. Vergallo. Ecg-derived respiratory signal using empirical mode decomposition. In *Medical Measurements and Applications Proceedings (MeMeA), 2011 IEEE International Workshop on*, pages 399–403. IEEE, 2011.

[15] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[16] R. Gomes. Long-term biosignals vizualization and processing. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2011.

[17] R.E. Klabunde. *Cardiovascular physiology concepts*. Lippincott Williams & Wilkins, 2005.

[18] Eric P. Widmaier, Hershel Raff, and Kevin T. Strang. *Vander's Human Physiology: The Mechanisms of Body Function*. McGraw-Hill Science/Engineering/Math, 10 edition, January 2005.

[19] J. Milic-Emili and M.M. Orzalesi. Mechanical work of breathing during maximal voluntary ventilation. *Journal of Applied Physiology*, 85(1):254–258, 1998.

[20] M.L. Ryan, C.M. Thorson, C.A. Otero, T. Vu, and K.G. Proctor. Clinical applications of heart rate variability in the triage and assessment of traumatically injured patients. *Anesthesiology research and practice*, 2011, 2011.

[21] A.J. Camm, M. Malik, JT Bigger, G. Breithardt, S. Cerutti, RJ Cohen, P. Coumel, EL Fallen, HL Kennedy, RE Kleiger, et al. Heart rate variability: standards of measurement, physiological interpretation, and clinical use. *Circulation*, 93(5):1043–1065, 1996.

[22] F. Pisano, G. Miscio, G. Mazzuero, P. Lanfranchi, R. Colombo, and P. Pinelli. Decreased heart rate variability in amyotrophic lateral sclerosis. *Muscle & nerve*, 18(11):1225–1231, 1995.

[23] S. Pavlovic, Z. Stevic, B. Milovanovic, B. Milicic, V. Rakocevic-Stojanovic, D. Lavrnic, and S. Apostolski. Impairment of cardiac autonomic control in patients with amyotrophic lateral sclerosis. *Amyotrophic Lateral Sclerosis*, 11(3):272–276, 2010.

[24] D.J. Ewing, JM Neilson, and P. Travis. New method for assessing cardiac parasympathetic activity using 24 hour electrocardiograms. *British heart journal*, 52(4):396–402, 1984.

[25] J.A. Dempsey, S.C. Veasey, B.J. Morgan, and C.P. O'Donnell. Pathophysiology of sleep apnea. *Physiological reviews*, 90(1):47–112, 2010.

[26] C.J.E. Wientjes. Respiration in psychophysiology: methods and applications. *Biological Psychology*, 34(2):179–203, 1992.

[27] D.T.H. Lai, R. Begg, and M. Palaniswami. *Healthcare Sensor Networks: Challenges Toward Practical Implementation*. CRC Press, 2011.

[28] The HDF Group. http://www.hdfgroup.org/why_hdf/. Accessed: 22/02/2012.

[29] Guido Van Rossum. *The Python Language Reference Manual*. Network Theory Ltd., manual edition, September 2003.

[30] HDF5 for Python. http://alfven.org/wp/hdf5-for-python/. Accessed: 22/02/2012.

[31] T. Oliphant. *Scientific Computing Tools For Python - Numpy*. Tregol Publishing, 2006.

[32] A.S. Tanenbaum. *Modern operating systems*, volume 2. Prentice Hall New Jersey, 1992.

[33] David Padua, editor. *Encyclopedia of Parallel Computing*. Springer, 2011 edition, September 2011.

[34] M. Lundstrom. Moore's law forever? *Science*, 299(5604):210–211, 2003.

[35] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1):1–10, 2007.

[36] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.

[37] Michael J. Quinn. *Parallel Computing: Theory and Practice*. McGraw-Hill Companies, 2 sub edition, September 1993.

[38] Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1st edition, August 1998.

[39] J.P. Shen and M.H. Lipasti. Modern processor design: fundamentals of superscalar processors. *Recherche*, 67:02, 2004.

[40] B.L. Chamberlain, D. Callahan, and H.P. Zima. Parallel programmability and the chapel language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.

[41] S. Marlow. Parallel and concurrent programming in haskell, 2011.

[42] J.L. Gaudiot, W. Bohm, W. Najjar, T. DeBoni, J. Feo, and P. Miller. The sisal model of functional programming and its implementation. In *Parallel Algorithms/Architecture Synthesis, 1997. Proceedings. Second Aizu International Symposium*, pages 112–123. IEEE, 1997.

[43] A.V. Nguyen, R. Wynden, and Y. Sun. Hbase, mapreduce, and integrated data visualization for processing clinical signal data. In *2011 AAAI Spring Symposium Series*, 2011.

[44] G. Fox, X. Qiu, S. Beason, J. Choi, J. Ekanayake, T. Gunarathne, M. Rho, H. Tang, N. Devadasan, and G. Liu. Biomedical case studies in data intensive computing. *Cloud Computing*, pages 2–18, 2009.

[45] Thomas Rauber and Gudula Rünger. *Parallel Programming: for Multicore and Cluster Systems*. Springer, 1st edition. edition, March 2010.

[46] Karl Heinz Hoffmann and Arnd Meyer, editors. *Parallel Algorithms and Cluster Computing: Implementations, Algorithms and Applications*. Springer, 1 edition, September 2006.

[47] Fayez Gebali. *Algorithms and Parallel Computing*. Wiley, 1 edition, April 2011.

[48] S. Pai, R. Govindarajan, and MJ Thazhuthaveetil. Limits of data-level parallelism. 2007.

[49] An HTML Notebook IPython. `http://ipython.org/ipython-doc/dev/interactive/htmlnotebook.html`. Accessed: 21/05/2012.

[50] Overview - Numpy and Scipy Documentation. `http://docs.scipy.org/doc/`. Accessed: 13/02/2012.

[51] Amazon Web Services. `http://aws.amazon.com/`. Accessed: 24/08/2012.

[52] J. Pan and W.J. Tompkins. A real-time qrs detection algorithm. *Biomedical Engineering, IEEE Transactions on*, (3):230–236, 1985.

[53] MA García-González, C. Vázquez-Seisdedos, and R. Pallàs-Areny. Variations in breathing patterns increase low frequency contents in hrv spectra. *Physiological measurement*, 21:417, 2000.

[54] A. Mortara, P. Sleight, G.D. Pinna, R. Maestri, A. Prpa, M.T. La Rovere, F. Cobelli, and L. Tavazzi. Abnormal awake respiratory patterns are common in chronic heart failure and may prevent evaluation of autonomic tone by measures of heart rate variability. *Circulation*, 96(1):246–252, 1997.

[55] J. Hunter and D. Fale. *The Matplotlib Users Guide*. May 2007.

[56] Vitalii Vanovschi. Parallel Python. http://www.parallelpython.com. Accessed: 30/03/2012.

[57] Douglas Crockford. *JavaScript: The Good Parts*. Yahoo Press, 1 edition, May 2008.

[58] Jon Duckett. *HTML and CSS: Design and Build Websites*. Wiley, 1 edition, November 2011.

[59] HTML/CSS to PDF converter written in Python. http://www.xhtml2pdf.com/. Accessed: 17/05/2012.

[60] Amazon Compute Cloud. http://aws.amazon.com/ec2/. Accessed: 24/08/2012.

[61] PuTTY User Manual. http://the.earth.li/~sgtatham/putty/0.58/htmldoc/. Accessed: 24/08/2012.

[62] FileZilla - The free FTP solution. http://filezilla-project.org/. Accessed: 24/08/2012.

Tools

A variety of tools was used during the course of this work. The programs, language packages and online services used throughout its realization are listed in this section.

The signal processing algorithms were essentially developed using the Python programming language [29]. The Python packages utilized were the numpy [31], scipy [50], matplotlib [55] and h5py [30]. To run the parallel algorithms the parallel python [56] package was utilized. This package makes it relatively easy to execute multiple tasks in parallel in computers with multiple processors, using load-balancing.

The IPython Notebook [49] was also used as a signal processing web-based environment.

The interface of the heart rate variability and respiratory analysis tools was made using JavaScript [57] and HTML [58].

The report generation feature of the developed signal processing tools was implemented using xhtml2pdf [59].

The Amazon Elastic Compute Cloud [51, 60] was used in order to execute the parallel algorithms remotely on machines with multiple processors, allowing the performance analysis of chapter 5. The communication to the remote machine was made via SSH. For that purpose the PuTTY [61] SSH client was used. The FileZilla [62] FTP client was extremely useful to transfer the signal files and the code to be executed on the remote machine.

B

Publications

During the development of this work, one paper was submitted and presented in an international conference. It is entitled "A New Tool for the Analysis of Heart Rate Variability of Long Duration Records", and was presented in SIGMAP 2012: *Internatinal Conference on Signal Processing and Multimedia Applications*.

A New Tool for the Analysis of Heart Rate Variability of Long Duration Records

Ricardo Chorão[1], Joana Sousa[2], Tiago Araújo[1,2] and Hugo Gamboa[1,2]
[1]*Physics Department, FCT-UNL, Lisbon, Portugal*
[2]*PLUX Wireless Biosignals S.A., Lisbon, Portugal*
r.domingos.chorao@gmail.com, jsousa@plux.info, taraujo87@gmail.com, hgamboa@fct.unl.pt

Keywords: ECG: Heart Rate Variability: Interactive Tool.

Abstract: The increased masses of data confronting us, originate a pressing need for the creation of a user interface for better handling and extracting knowledge from it. In this work we developed such a tool for the analysis of Heart Rate Variability (HRV). The analysis of HRV in patients with neuromuscular diseases, sleep disorders and cardiorespiratory problems has a strong impact on clinical practice. It has been widely used for monitoring the autonomic nervous system (ANS), whose regulatory effect controls the cardiac activity. These patients need to be continuously monitored, which originates data with huge sizes. Our interactive tool can perform a fast analysis of HRV from such data. It provides the analysis of HRV in time and frequency domains, and from non-linear methods. The tool is suitable to be run in a web environment, rendering it highly portable. It includes a programming feature, which enables the user to perform additional analysis of the data by giving direct access to the signals in a signal processing programming environment. We also added a report generation functionality, which is extremely important from a clinical standpoint, on which the evolution in time of relevant HRV parameters is depicted.

1 INTRODUCTION

The ever increasing development of clinical systems for patients' biosignals monitoring has given the clinician and the researcher a way of assessing the patients' health state (Silva, Palma and Gamboa, 2011). The possibility of drawing relevant medical conclusions from the analysis of biosignals arises from the fact that they contain information which is directly related to the physiological mechanisms that originated them (Bronzino, 2000).

The constant monitoring of ambulatory patients, with conditions such as neuromuscular diseases, sleep disorders or chronic heart problems, has great value and may provide the clinician with a way for a more objective therapy (Davenport, Ross and Deb, 2009). It also plays a preventive role, enabling the clinician to undertake a faster medical response.

Recordings of several hours originate huge amounts of data, which need to be processed and analysed rapidly.

The Heart Rate Variability (HRV) is a very important tool to analyse ECG signals with detail. In HRV analysis, the oscillations between consecutive heartbeats are measured, both in time and frequency domains.

Since the HRV is strongly associated with neuro-regulation mechanisms, it also allows one to evaluate the modulation of the heart rate by the ANS (Ryan, Thorson, Otero, Vu and Proctor, 2011).

The first step of HRV analysis is the measurement of all the inter-beat intervals, which can be achieved through the detection of the peaks in the ECG record, as shown in figure 1.

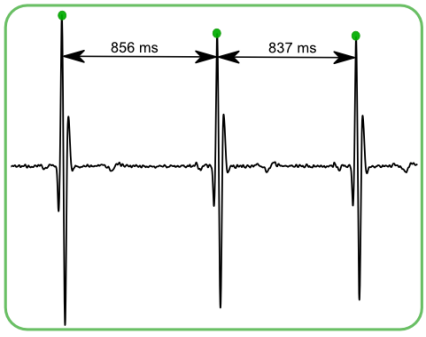


Figure 1: Highlight of 3 ECG peaks (the R peaks) and measurement of 2 heartbeat intervals (or RR intervals).

There are several methods of analysis of HRV that can be divided mainly into Time and Frequency domain methods (Camm et al., 1996).

Time domain analysis of HRV allows calculation of statistical parameters concerning the RR intervals, tachogram (evolution of the RR interval duration in time) and instant heart rate representations. These visual representations are extremely important to identify disturbances of the cardiac rhythm, such as arrhythmias or skipped heartbeats.

Frequency domain analysis methods are particularly useful for analysing long term ECGs. They are based on power spectral density (PSD) analysis of RR intervals and provide information about how power distributes itself among the various frequencies. The power in different frequency bands is closely related to different branches of the ANS, which makes the power distribution a very important tool to identify ANS related problems.

We present a new tool capable of performing a HRV analysis from several hour long ECG records.

There are many HRV analysis tools available, such as "Biopac HRV Algorithm", "HRVAS: HRV Analysis Software" (using Matlab) and the "HRV Toolkit" (PhysioNet). They all provide frequency and time domain analysis of ECG records. However they are not interactive or do not allow one to deal directly with long duration ECGs, which was our main concern. In our tool, the HRV analysis begins in the very ECG records, in spite of the available tools, which assume the ECG peak detection was made elsewhere. Our tool also surpasses the currently available tools in what portability is concerned, since it can be executed in a web environment.

The analysis takes only a few seconds despite of the recording length of several hours. Its flexibility and possibility of a personalized and detailed analysis make it suitable for both the clinician and the researcher.

Its user-friendliness makes it pleasant to use and easy to learn, two very important characteristics in any software project according to (Holzinger, 2005).

In the following section the programming architecture of the application is explained and the features of the developed tool are depicted. Finally, we discuss some improvements that can be made, and conclude the work at the end of the paper.

2 HRV ANALYSIS TOOL

We developed a user-friendly and interactive tool which runs in a web environment, and therefore can be accessed easily from anywhere with an internet connection. It gives the user control over powerful signal processing algorithms, enabling a human-computer interaction and the exploration of large datasets comprising biological signals.

The developed tool allows a detailed analysis of ECG records with several hours. It is coupled to a previously developed biosignals visualization tool (Gomes, 2011; Gomes, Neuza, Sousa and Gamboa, 2012) making it possible to simultaneously visualize the portion of the signal being analysed and the processing results.

The analysis tool also includes a report generation feature, in which the evolution of the patient's heart rate, among other significant parameters can be tracked. The report is extremely important from a medical point of view, making the processing results portable and an extension of the patient's clinical file.

For fast and random accessing of processing results, we used the HDF5 file format, specifically designed to store and access large datasets (The hdf group, 2012).

2.1 Acquisition System

The signals were acquired with the aid of a chest wrap, which integrates respiration, ECG and accelerometer sensors. They are sampled at 1 kHz and with a 12 bit resolution and are then sent by Bluetooth to a mobile phone.

The acquisition process is occurring under project "wiCardioResp". This project is developing technology to remotely monitor patients with neuromuscular diseases and cardiorespiratory problems while the patients are comfortably at home (PLUX, 2012). The acquisitions were carried out with their agreement, during the night, and last approximately 7 hours.

2.2 Application Platform

We coupled the HRV analysis tool to a previously developed web-based biosignals visualization tool. The application is executed in a web browser, rendering it highly portable, and giving it the potential of being accessed from anywhere with an internet connection.

All signal processing algorithms were written in Python and used SciPy (SciPy, 2012), a Python package for scientific computing. The application platform was created using JavaScript and HTML, browser-supported languages.

2.3 Programming Architecture

HRV analysis requires the detection of the different heartbeats. This is typically achieved using an ECG peak detection algorithm, because the ECG peak morphology (see figure 1) makes it an easily detectable feature in the ECG waveform.

The utilized peak detector was an adaptation of the Pan and Tompkins algorithm (Pan and Tompkins, 1985).

To apply an ECG peak detection algorithm over a long duration record, as a whole, is unfeasible, because of the amount of memory required to perform the computation. The strategy employed to solve this issue consisted in partitioning the record into several equally sized portions, processing them individually. This approach gives rise to possible detection errors in the borders, which we overcame considering a fixed number of overlapping samples between consecutive portions. We used an overlap of 400 ms, carefully chosen after analysing several ECG records. The bigger the overlapping period, the longer the processing will last. However, when dealing with long duration records, the additional processing time due to the overlapping size is negligible (at worst a few seconds more in a record with several hours).

The strategy that we undertook regarding double peak detections (one of them in an overlapping region and the other in the beginning of the subsequent portion of the signal) has a physiological basis, as following explained:

When a muscle contracts, an action potential is generated. Then, there is a long absolute refractory period, which, for the cardiac muscle, lasts about 250 ms. During this period, the cardiac muscle can't be re-excited, which results in an inability for heart contraction (Widmaier, Raff and Strang, 2005). Thus, we can theoretically consider a maximum heart rate of about 4 beats per second, or 240 beats per minute. Under this assumption, considering that in a signal sampled at 1 kHz, 250 ms would be represented by 250 samples, and after a thorough observation of the behaviour of the peak detection algorithm, we stated that if the same peak were detected twice, the detection would never occur with such deviation in samples. Therefore, we considered that if two R peaks were detected and separated by less than 50 ms, we were dealing with a double detection and only considered one of them. With a tolerance of 50 ms, we ensured that double detections would not occur on the overlapping regions, while not dismissing any occurrence of a physiologically possible RR interval.

The amount of time it takes for the peak detection to complete depends on several factors: the sampling frequency at which the signals were acquired, the signals' duration, the peak detection algorithm and the processing unit of the machine in which the computations take place. It took us about 8 minutes to process a 7 hour long ECG, sampled at 1 kHz on a single 2.2 GHz processor.

The peak detection phase can be regarded as pre-processing and only has to be done once. Its duration may be easily decreased applying parallel programming techniques, due to the embarrassingly parallel nature of the problem.

The peak detection information was stored in a .h5 file (HDF5 file extension) for fast and random access, a necessary feature to make the analysis of portions of the signal possible.

For analysing a specific part of the ECG, for instance, between the first and second hour of the recording, all that has to be done is accessing the processing results in the .h5 file and select the samples between the first and second hour. This information is then analysed in just a few seconds. The simultaneous visualization of the signal and the processing results is an extremely important capability allowing a more detailed and accurate analysis by the clinician and the researcher.

2.4 Tool Features

The HRV analysis provided is composed of time and frequency domain parameters and also of some visual representations, which, allied to the biosignals visualization tool allow a more detailed data inspection.

The time and frequency domain parameters provided are depicted in figure 2.

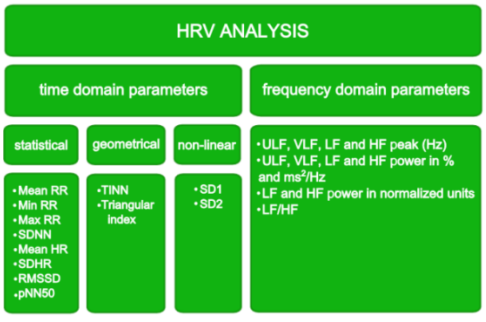


Figure 2: The most important HRV analysis parameters provided by our tool.

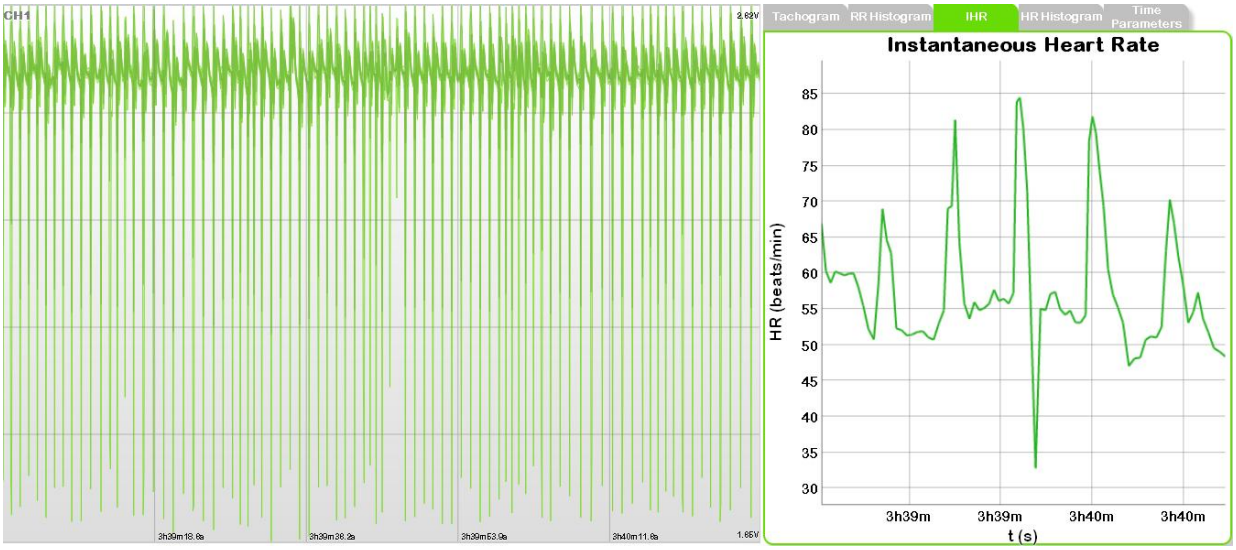


Figure 3: Instantaneous Heart Rate of a 1m30s period from a night recording with a duration of 7h17m.

Besides time and frequency domain parameters, our tool includes several important visualization features as well:

- tachogram and instantaneous heart rate (figure 3). Since each point in these representations is directly associated with a cardiac cycle, we included a zooming feature, directly synchronized with the ECG visualization. This makes the inspection of the processing results interactive and gives the clinician the possibility to rapidly identify periods of arrhythmia or other significant events and observing them on the ECG record;
- RR Interval and Instantaneous Heart Rate histograms with a bin size of 1/128 s (standard bin size according to Camm et al., 1996);
- Power Spectral Density, with highlight of the relevant frequency bands and zooming feature for a more detailed analysis. The frequency range of the different frequency bands is as follows: ULF (0 - 0.003 Hz), VLF (0.003 - 0.04 Hz), LF (0.04 - 0.15 Hz), HF (0.15 - 0.04 Hz);
- Poincaré plot.

Extremely important as well is a report generation feature. The report includes a global analysis in which the parameters and representations already described are presented. It also illustrates the evolution of those parameters in time.

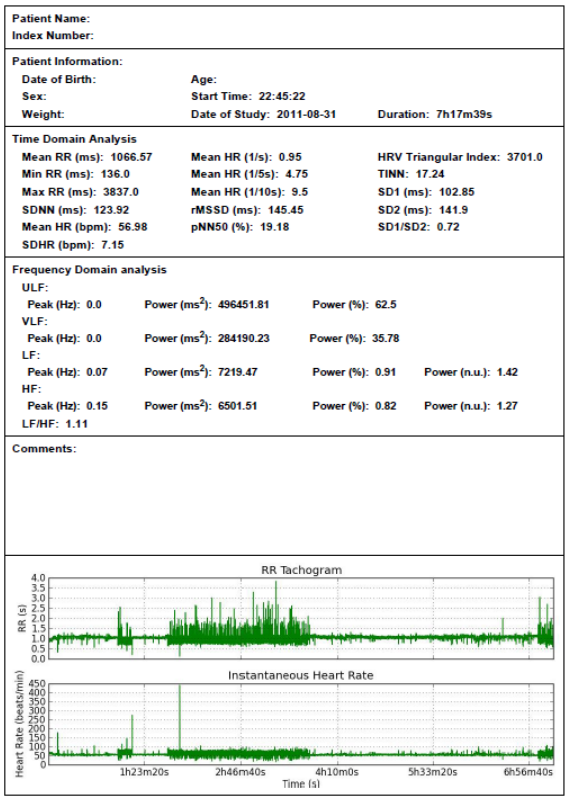


Figure 4: First page of a HRV analysis report from a night recording which lasts 7h17m. This page presents a global time and frequency domain analysis.

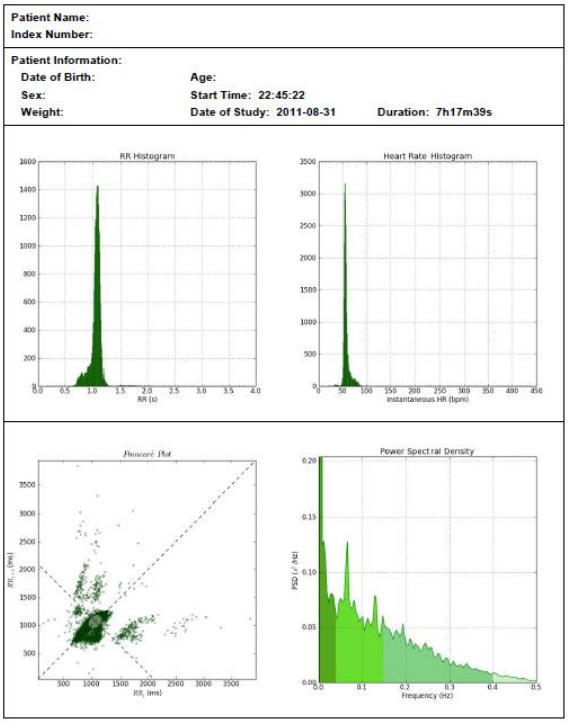


Figure 5: Second page of the report showing power spectral density, histograms and Poincaré plot.

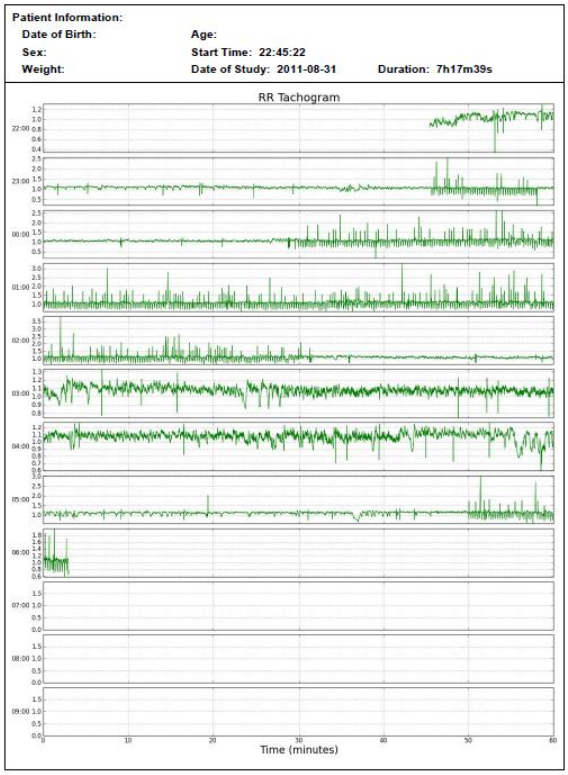


Figure 6: Tachogram hour-by-hour evolution. This representation is extremely useful for identifying periods of arrhythmia and other events through the night.

We also included a programming feature, particularly focused on the researcher, in a console. This way, additional signal processing can be done, using all of the SciPy capabilities (SciPy, 2012). We loaded several important variables into the namespace: the sampling frequency, an array with all the R peaks, in samples, that were detected in the selected time range, the beginning and ending of the selected time range, in samples. All these variables contain the necessary and sufficient information to enable a personalized analysis by the programmer or the researcher.

3 CONCLUSIONS AND FUTURE WORK

In this work we developed an HRV analysis tool suited for the analysis of long duration records.

Taking into account all described tool features, we conclude that it offers a flexible, detailed and accurate way of analysing long duration ECGs. The report generation and the programming mode features are very important and give the tool a greater flexibility and portability, besides that of a web-based application.

The zooming capabilities and the synchronism with a biosignals visualization tool make it highly interactive and provide a way for better and faster data discovery.

The user interface allows control over powerful signal processing algorithms and can be regarded as a way to allow the non-expert to still utilize them for clinical or research purposes.

In future work we intend to develop an algorithm which will allow us to determine the minimum number of overlapping samples for optimal ECG peak detection. We also aim to add the functionality of analysing several records simultaneously and to generate the corresponding HRV analysis reports using parallel programming techniques.

ACKNOWLEDGEMENTS

This work was partially supported by National Strategic Reference Framework (NSRF-QREN) under project ”wiCardioResp” and ”AAL4ALL”, and Seventh Framework Programme (FP7) program under project ICT4Depression, whose support the

authors gratefully acknowledge. The authors also thank PLUX, Wireless Biosignals for providing the acquisition system and sensors necessary to this work.

REFERENCES

H. Silva, S. Plama, H. Gamboa., 2011. AAL+: Continuous Institutional and Home Care Through Wireless Biosignal Monitoring Systems. *Chapter in Handbook of Digital Homecare, Lodewijk Bos, Adrie Dumay, Leonard Goldschmidt, Griet Verhenneman and Kanagasingam Yogesan*, Springer.

J.D. Bronzino., 2000. *The biomedical engineering handbook*, volume 2. CRC Press. Boca Raton , 2nd edition .D.M. Davenport, FJ Ross, and B. Deb., 2009. Wireless propagation and coexistence of medical body sensor networks for ambulatory patient monitoring. In

D.M. Davenport, FJ Ross, and B. Deb., 2009. Wireless propagation and coexistence of medical body sensor networks for ambulatory patient monitoring. In Wearable and Implantable Body Sensor Networks. *Sixth InternationalWorkshop on pages 41–45. IEEE, 2009*.

M.L. Ryan, C.M. Thorson, C.A. Otero, T. Vu, and K.G. Proctor., 2011. Clinical applications of heart rate variability in the triage and assessment of traumatically injured patients. *Anesthesiology research and practice, 2011*.

A.J. Camm, M. Malik, JT Bigger, G. Breithardt, S. Cerutti, RJ Cohen, P. Coumel, EL Fallen, HL Kennedy, RE Kleiger, et al., 1996. Heart rate variability: standards of measurement, physiological interpretation, and clinical use. *Circulation, 93(5):1043–1065*.

PhysioNet – http://physionet.org/tutorials/hrv-toolkit/. Accessed 26/05/2012

Holzinger, A., 2005. Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74.

R. Gomes, N. Neuza, J. Sousa, H. Gamboa., 2012. Long-term biosignals visualization and processing. *Proceedings of Biosignals - 5th International Conference on Bio-Inspired and Signal Processing (BIOSIGNALS 2012)*, Vilamoura, Portugal.

Ricardo Gomes., 2011. Long-term biosignals visualization and processing. Master thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, 2011.

The hdf group – http://www.hdfgroup.org/why_hdf/. Accessed: 26/05/2012

PLUX (2012) – http://wicardioresp.plux.info/pt-pt. Accessed: 25/05/2012

Scipy – http://www.scipy.org/. Accessed: 26/05/2012

Pan, J. and Tompkins, W. (1985). A real-time qrs detection algorithm. *Biomedical Engineering, IEEE Transactions on, (3):230–236*.

Eric P.Widmaier, Hershel Raff, and Kevin T. Strang., 2005. *Vander's Human Physiology: The Mechanisms of Body Function*. McGraw-Hill International Edition, 10th edition.