Universidade de Aveiro
2010/2011

Departamento de Electrónica, Telecomunicações e Informática

Francisco Xavier dos Santos Fonseca

Processamento de Imagens Médicas usando GPU

GPU Power for Medical Imaging

Universidade de Aveiro
2010/2011

Departamento de Electrónica, Telecomunicações e Informática

Francisco Xavier dos Santos Fonseca

Processamento de Imagens Médicas usando GPU

GPU Power for Medical Imaging

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática (M.I.E.C.T.), realizada sob a orientação científica do professor Doutor José Maria Amaral Fernandes, professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Mestre Ilídio Castro Oliveira, Assistente Convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, tendo a colaboração do professor Doutor Guilherme Campos, professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho à minha família, a todos os meus amigos que me apoiaram no meu percurso académico e a todas as pessoas que, de alguma forma, contribuíram para o meu sucesso.

o júri

presidente

Professor Doutor Tomás Oliveira e Silva
professor associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Alberto José Proença
professor catedrático do Departamento de Informática da Universidade do Minho

Professor Doutor José Maria Amaral Fernandes
professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Mestre Ilídio Castro Oliveira
assistente convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Agradecimentos / Acknowledgements

Professor Doutor José Maria Fernandes, Mestre Ilídio Oliveira, Professor Doutor Guilherme Campos, Eduardo Dias.

palavras-chave Imagem Médica, Paralelismo, GPU, CUDA, SVM, CapView

resumo

A aplicação CapView utiliza um algoritmo de classificação baseado em SVM (*Support Vector Machines*) para automatizar a segmentação topográfica de vídeos do trato intestinal obtidos por cápsula endoscópica. Este trabalho explora a aplicação de processadores gráficos (GPU) para execução paralela desse algoritmo. Após uma etapa de otimização da versão sequencial, comparou-se o desempenho obtido por duas abordagens: (1) desenvolvimento apenas do código do lado do *host*, com suporte em bibliotecas especializadas para a GPU, e (2) desenvolvimento de todo o código, incluindo o que é executado no GPU. Ambas permitiram ganhos (*speedups*) significativos, entre 1,4 e 7 em testes efetuados com GPUs individuais de vários modelos. Usando um *cluster* de 4 GPU do modelo de maior capacidade, conseguiu-se, em todos os casos testados, ganhos entre 26,2 e 27,2 em relação à versão sequencial otimizada. Os métodos desenvolvidos foram integrados na aplicação CapView, utilizada em rotina em ambientes hospitalares.

keywords Medical Imaging, Parallelism, GPU, CUDA, CapView

abstract The CapView application uses a classification algorithm based on SVMs (Support Vector Machines) for automatic topographic segmentation of gastrointestinal tract videos obtained through capsule endoscopy. This work explores the use graphic processors (GPUs) to parallelize the segmentation algorithm. After an optimization phase of the sequential version, two new approaches were analyzed: (1) development of the host code only, with support of specialized libraries for the GPU, and (2) development of the host and the device's code. The two approaches caused substantial gains, with speedups between 1.4 and 7 times in tests made with several different individual GPUs. In a cluster of 4 GPUs of the most capable model, speedups between 26.2 and 27.2 times were achieved, compared to the optimized sequential version. The methods developed were integrated in the CapView application, used in routine in medical environments.

Contents

1. Introduction 1
1.1. Motivation and context 2
1.2. Objectives 2
1.3. Dissertation structure 3
2. Background 5
2.1. Medical Imaging: endoscopy 5
2.1.1. Conventional Endoscopy 6
2.1.2. Endoscopic Capsule 6
2.2. Topographic segmentation of digestive tract 8
2.2.1. Segmentation methods 9
2.2.2. CapView and the endoscopic capsule 10
2.3. Graphical Processing Units 11
2.3.1. Relevance and usage of GPUs 11
2.3.2. GPUs in medical imaging 12
2.4. The CUDA architecture 13
3. CapView's Topographic Segmentation Algorithm 17
3.1. Video Processing 18
3.2. Support Vector Machines in CapView 19
3.3. CapView SVM classifier 22
3.4. Classification's Aggregation and Topographic Markers 24
3.4.1. From topographic locations to segmentation 24
3.4.2. Aggregation example 25
3.5. Example of the Execution Flow 26
4. Parallelization of the Segmentation Algorithm 29
4.1. Data Alignment in Memory 30
4.2. Amdahl's Law 31
4.3. Parallel Code Implementation and Evaluation 34
4.3.1. CUBLAS Approach 35
4.3.2. Initial Approach 37
4.3.3. Independent Thread Approach 40
4.4. Parallelization's profiling 43
4.5. Multi GPU usage 45
4.6. Comparison with Other Parallel Approaches 48

5. Integration of GPUs in CapView 51
5.1. The GUI interaction 51
5.2. Evaluation 53
5.3. Integration issues 53
5.3.1. Multithreaded GPU 53
5.3.2. FFmpeg: the video 54
5.4. Integration Considerations 54
6. Conclusions and Future Work 57
6.1. Assessment of purposed objectives 57
6.2. Future work 59
7. References 61
8. Appendix 67

List of Figures

FIGURE 1 BASIC DESIGN - CONTROL HEAD AND BENDING SECTION [23] 6
FIGURE 2 WIRELESS ENDOSCOPIC CAPSULE [27] 7
FIGURE 3 EXAMPLES OF ENDOSCOPIC CAPSULE IMAGES [9] FOR THE FOUR DIFFERENT TOPOGRAPHIC ZONES: (A) ESOPHAGUS, (B) STOMACH, (C) SMALL INTESTINE AND (D) LARGE INTESTINE. 8
FIGURE 4 IDENTIFICATION OF THE THREE TOPOGRAPHIC BARRIERS [32] 9
FIGURE 5 CAPVIEW'S LAYOUT 10
FIGURE 6 EXECUTION OF A PROGRAM THAT USES THE HOST AND DEVICE [72] 14
FIGURE 7 LOGICAL SCHEMA OF A GPU FOLLOWING A CUDA ARCHITECTURE [72] 15
FIGURE 8 STREAMMING PROCESSOR TOPOLOGY [72] 15
FIGURE 9 ACTIVITIES DIAGRAM OF THE SEQUENTIAL CODE 17
FIGURE 10 VIDEO PROCESSING FLOW CHART 19
FIGURE 11 SVM CLASSIFICATION – REPRESENTED GRAPHICALLY AS THE BOUNDARY OF SQUARES CLASS. (ADAPTED FROM [83]) . 21
FIGURE 12 LINEAR SEPARATING HYPERPLANES (ADAPTED FROM [12]) FOR: 21
FIGURE 13 TRAINING STAGE OF CAPVIEW'S APPLICATION 22
FIGURE 14 CLASSIFICATION SCHEME 23
FIGURE 15 VIDEO EXAMPLE CLASSIFICATIONS 25
FIGURE 16 TOPOGRAPHIC BARRIERS ESTIMATION 26
FIGURE 17 DECODED VIDEO FRAME EXAMPLE 27
FIGURE 18 HISTOGRAM OF VIDEO FRAME EXAMPLE 27
FIGURE 19 NORMALIZED HISTOGRAM 28
FIGURE 20 MODEL FILE 30
FIGURE 21 SUPPORT VECTORS STORED IN MEMORY (PER MODEL FILE) 31
FIGURE 22 GENERIC PROGRAM'S EXECUTION 32
FIGURE 23 PROFILING OF CAPVIEW'S SEQUENTIAL CODE 33
FIGURE 24 AMDAHL'S THEORETICAL LIMITS 34
FIGURE 25 CUBLAS SCALAR VECTOR CALCULATION 36
FIGURE 26 INITIAL CLASSIFICATION APPROACH SCHEME 38
FIGURE 27 INDEPENDENT THREAD APPROACH 40
FIGURE 28 MEMORY ACCESS BEFORE EFFICIENCY MAXIMIZATION 41
FIGURE 29 SUPPORT VECTORS IN A COLUMN MAJOR DISPOSAL 42
FIGURE 30 COALESCED MEMORY ACCESS 42
FIGURE 31 PROFILING OF CAPVIEW AFTER GPU OPTIMIZATIONS ON THE CLASSIFICATION 44
FIGURE 32 VIDEO SEGMENTS AND N.º GPUS 45
FIGURE 33 EXECUTION FLOW OF THE PARALLEL CODE IN MULTI-GPU APPROACH 46
FIGURE 34 PARALLEL CODE'S EXECUTION IN MULTI-GPU MODE 47
FIGURE 35 AVERAGE EXECUTION TIMES PER 100 MB 50
FIGURE 36 INTEGRATION OF PARALLEL IMPLEMENTATION IN CAPVIEW'S APPLICATION 52
FIGURE 37 TOPOGRAPHIC EVENTS MARKED IN THE MAIN APPLICATION. 52

List of Tables

TABLE 2-1 EXAMPLES OF GPU APPLICATION IN MEDICAL IMAGING 12
TABLE 4-1 TEST VIDEOS 29
TABLE 4-2 LM VALUES FOR ALL MODEL FILES 30
TABLE 4-3 OVERALL SPEEDUP 31
TABLE 4-4 MAJOR ACTIVITIES OF CAPVIEW'S EXECUTION 32
TABLE 4-5 MAXIMUM THEORETICAL SPEEDUPS FOR MACHINES 1 AND 2 34
TABLE 4-6 SEQUENTIAL REFERENCE VS. CUBLAS APPROACH 36
TABLE 4-7 CUBLAS APPROACH'S OVERALL SPEEDUPS 37
TABLE 4-8 SEQUENTIAL CLASSIFICATION VS. INITIAL APPROACH 39
TABLE 4-9 OVERALL SPEEDUPS, IN THE INITIAL APPROACH 39
TABLE 4-10 SEQUENTIAL CLASSIFICATION VS. INDEPENDENT THREAD APPROACH 42
TABLE 4-11 OVERALL SPEEDUPS OF THE INDEPENDENT THREAD APPROACH 43
TABLE 4-12 PROFILING VALUES AFTER PARALLELIZATION 43
TABLE 4-13 THEORETICAL VALUES FOR MACHINES 1 AND 2 44
TABLE 4-14 OVERALL EXECUTION TIMES COMPARISON OF HISTOGRAM OPS 44
TABLE 4-15 EXECUTION TIMES FOR MULTIPLE GPU USAGE 48
TABLE 4-16 SEQUENTIAL EXECUTION TIMES IN BOTH MACHINES 48
TABLE 4-17 EXECUTION VALUES WITH 1 GPU 48
TABLE 4-18 EXECUTION VALUES WITH VARIOUS GPUS 49
TABLE 4-19 EXECUTION VALUES IN GRID INFRAESTRUCTURE 49
TABLE 4-20 EXECUTION VALUES IN CLUSTER COMPUTING 50
TABLE 5-1 INTEGRATION EVALUATION IN MACHINE 3 53
TABLE 8-1 SPECIFICATIONS 67

List of Acronyms

API	Application Programming Interface
CPU	Central Processing Unit
GI	Gastro Intestinal
GPU	Graphic Processing Unit
GRID	Global Resource Information Database
HSV	Hue, Saturation and Value color system
MB	Mega Byte
MJPEG	Motion JPEG video codec
PC	Personal Computer
SC	Scalable Color
SM	Streaming Multiprocessors
SP	Streaming Processors
SPMD	Single Program Multiple Data
SVM	Support Vector Machine
WEC	Wireless Endoscopic Capsule

1. Introduction

Medical imaging has become gradually relevant, both in research and in clinic applications [1]. Image acquisition, processing and visualization depends on great computational resources [2], but human interaction is still needed most of the times, either for data validation [3], interactive tuning [4] or making decisions regarding the presented data [5] (either raw or as a result of post processing). The data to treat can be overwhelming, given their complexity (e.g. multidimensional data) or volume (long monitoring). One good example is capsule-based endoscopy.

Traditional endoscopy allows gastrointestinal tract observation [6], performed by specialized and experienced clinicians. Though, with this method, it is not possible to reach certain zones (especially in the small intestine [3]), which are relevant in clinical diagnosis. Recently, a video-based system supported by an automated endoscopic capsule enabled the visualization of such areas without resort to more invasive methods [7]. The main problem of this method it still demands both time and concentration for the clinician to analyze the resulting videos (typically more than 6 hours of video). In the analysis clinicians must segment the gastrointestinal tract in anatomic zones, in order to look of specific pathologies (like ulcers or bleedings) on pathology related zones. For that reason, applications like CapView [8] were developed to support a more efficient analysis of the endoscopic capsule data. CapView presents an interactive interface that allows the review and classification of the video data. It also provides an automated segmentation method for the digestive tract [9]. This segmentation provides a quick way for clinicians to navigate directly to areas of interest improving the

clinician's analysis process [8]. However, automated segmentation still can take over 1 hour [10]. In this context, accelerating the analysis can be an added value in the clinical environment as it can reduce the overall endoscopic capsule data analysis time.

1.1. Motivation and context

Recently GPUs (Graphical Processing Units) appeared as a solution to support medical imaging analysis, an analysis that can deeply profit from graphical card power [11]. GPUs are used to perform parallel operations on graphics data (with games and movies as first targets), and as a pack of processors, able to execute parallel computations on sets of data simultaneously, they are increasingly being adapted to general purpose applications [12]. As almost every computer has at least one GPU, it seems natural to explore GPUs as an option to improve the efficiency of the automated segmentation in CapView application.

However, to be able to extract the most of GPU architectures, different problems may imply different algorithmic approaches [13]. From the start, it may not be clear which GPUs related customizations should be used and be able to predict the overall impact of them in terms of specific runtime requirements (on the clarity or obfuscation of original algorithms) that may be crucial when having portability and extendibility in mind.

1.2. Objectives

The objective of this dissertation is to assess the applicability of GPUs to a specific medical imaging problem: automated segmentation of the digestive track in CapView. Our main purpose is to optimize the segmentation algorithm in CapView through the use of GPUs.

We investigated in detail the contributions that GPUs of different models (specifications in appendix 0) can offer in terms of computational speedup. The results of that analysis were measured in several equipments, with different costs.

To fulfill our objectives we addressed three concrete questions:

- Is it worth applying the GPUs parallelization to this algorithm?
    - For answering this, we will analyze the sequential algorithm, in order to see where relevant speedups can be achieved.
- How can the algorithm optimization occur while preserving the algorithm

integrity?

- We will analyze the sequential algorithm for the purpose of assessing the best way to take advantage of GPUs without compromising the algorithm's correctness.

- How does the GPU implementation compare with other advanced computing solutions?
  - We will present a comparison between the GPU based solution and previously developed algorithms (for cluster and Grid computing).

1.3. Dissertation structure

Here is a summary of the following chapters:

- **Chapter 2 - Background:** gives a quick overview of medical imaging in endoscopic applications, and briefly presents CapView's automatic segmentation tool. A review on GPUs and their achievements in medical imaging is also presented.

- **Chapter 3 – CapView's Topographical Segmentation Code:** discusses the pre-existing sequential segmentation code of CapView's application, the basis of the optimization and parallelization work. The methods for image classification and topographic barrier definition are described. Examples of execution behaviors are also given.

- **Chapter 4 - Parallelization:** presents different parallelization approaches for CapView's automatic segmentation tool and describes the optimization work carried out prior to parallelization. The maximum theoretical speedup is estimated based on Amdahl's law. The application of multiple GPUs is explored and the various solutions are compared in terms of computation time.

- **Chapter 5 - Integration of GPUs in CapView:** describes the process of integration of the CapView GPU parallel version into CapView's initial application. The integration process is described, and commented on.

- **Chapter 6 - Conclusions and Future Work:** summarizes and compares the

outcome of the various parallelization approaches, and discusses the viability of some ideas.

2. Background

2.1. Medical Imaging: endoscopy

Digital medical imaging is a challenging area, requiring constant development and innovation in computational techniques [2]. Numerous examples are available, such as sophisticated algorithms for image registration [14], image segmentation [5] (for automatic identification of regions or objects) or simply image enhancement [15] for noise reduction (very important in medical imaging). Visualization of medical data is a field that grew rapidly, with the appearance of new display devices [16] (capable of showing 3D data) and the drive to represent increasingly complex data [17]. For instance, segmentation has tremendous importance in the medical field, since it can be used, among other things, to automatically identify desired objects in a given image (for instance the segmentation of the heart's chambers in echocardiography [18] or abdominal CT segmentation [19]).

A good case study in medical imaging is the Gastroenterology field, a medical field that diagnoses and treats gastric diseases. Endoscopy is now the standard method for diagnostics and therapeutic treatment in this field since the early trials in the late fifties [20]. Currently the traditional tool is the fiberoptic gastroscope [6] that is capable of examine inner organs and hidden cavities [21], a procedure that was not possible through using older rigid endoscopes. The main drawback is that is not possible to track the actual position of the camera, or field of view, during the exam[22] and, by consequence, is difficult to map observed images to the location in GI tract.

2.1.1. Conventional Endoscopy

Despite the variety of endoscopes available most follow a basic design, as illustrated in Figure 1. The endoscope tube is conducted through the GI (Gastro Intestinal) tract and bends the tip of the tube for a better perception of hidden cavities (such as duodenum). The light in the extremity and the imaging system (that displays the images in an external monitor) makes it possible to see, in real time and with precision, the conditions of inner organs.

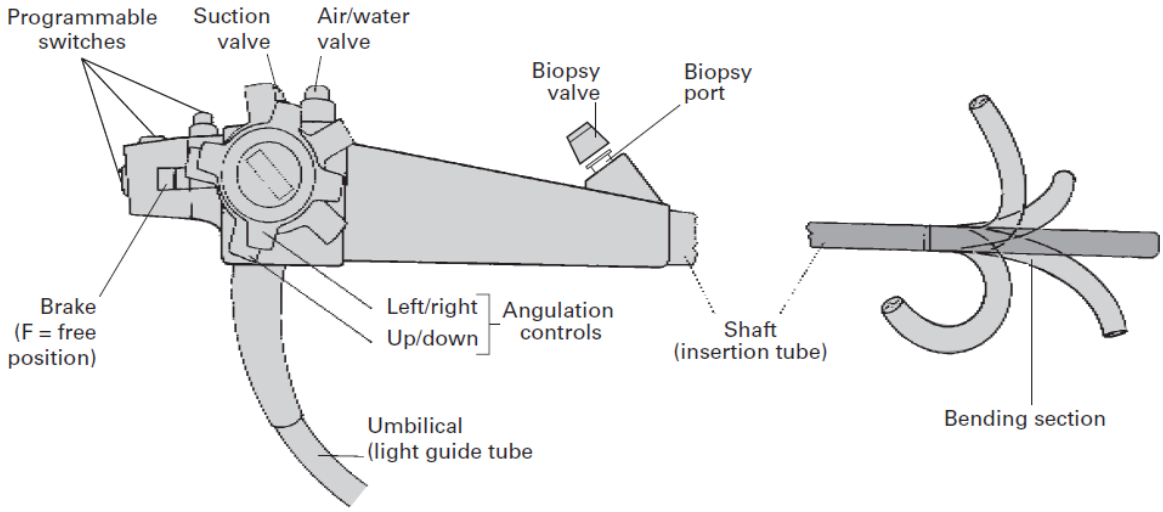

Figure 1 Basic design - control head and bending section [23]

Furthermore, it has suction capabilities that may be used in localized treatment of pathologies (such as bleedings), which makes it a preferred method for direct intervention, compared to more invasive methods like surgery [24]. Endoscopy can be used in the detection of gastric ulcers, detection of cancer (that can be located from the esophagus to the duodenum, as well as colon cancer) or even performing biopsies. Unfortunately, endoscopy is still considered an invasive technique, so it suffers of all risks associated with invasive procedures namely complications related to sedation [25].

2.1.2. Endoscopic Capsule

The endoscopic capsule is the first solution to film the full human gastrointestinal tract in an autonomous way [26]. Endoscopic capsule enabled the production of images in all of the small intestine's length (around 6 meters long), being able to reach places that traditional

endoscopy can't [3] with the added value of being painless and with reduced complications [21].

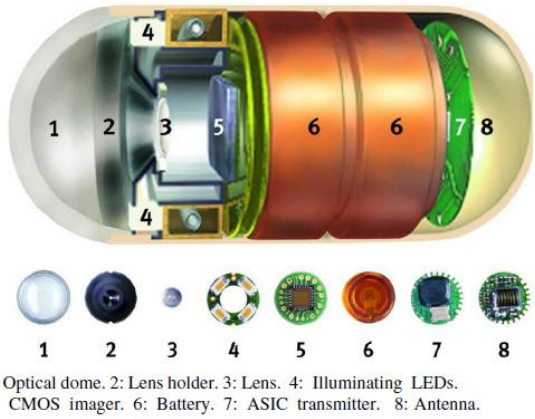

Figure 2 Wireless Endoscopic Capsule [27]

The endoscopy through a wireless capsule consists in [9]:

- The capsule itself (Figure 2): Plastic capsule with a weight of 3.7 g and 11 mm in diameter x 26 mm in length. Contains a chip responsible for producing images, a focal lens, lights for illumination, a radio transmitter antenna and a battery;
- Antenna: An external antenna, responsible for receiving the images produced by the capsule;
- Hard Drive: A portable hard drive for video storage, placed in the patient's belt;
- Power supply: Located in the belt of the patient and used to power the hard drive and the antenna;

After the exam is completed [28], the hard drive is connected to the workstation (a computer capable of analyzing the video, with proprietary software) and the produced video is uploaded and analyzed by the medical specialist.

In the specific WEC (Wireless Endoscopic Capsule) used, the resulting video contains color images of 256 x 256 pixels, recorded at 2 frames per second. Since the capsule is not self-propelled or controlled externally, it moves along the GI tract through peristalsis (natural movements of the digestive tract organs) [3]. This results in video lengths typically between 6 and 8 hours [29] (assuming enough capsule battery charge). The images annotated during the exams are similar to those presented in Figure 3, taken in each of the four topographic zones (a: esophagus; b: stomach; c: small intestine; d: large intestine).

One of the most inconvenient aspects of this technology is the video length. That takes medical specialists at least 1 hour in the video analysis [30] to search for unusual situations. In average there are over 50 000 video frames (similar to those illustrated in Figure 3) per exam. The clinical analysis consists in searching of known pathologies like gastric ulcers, bleedings or even masses that may be cancers related.

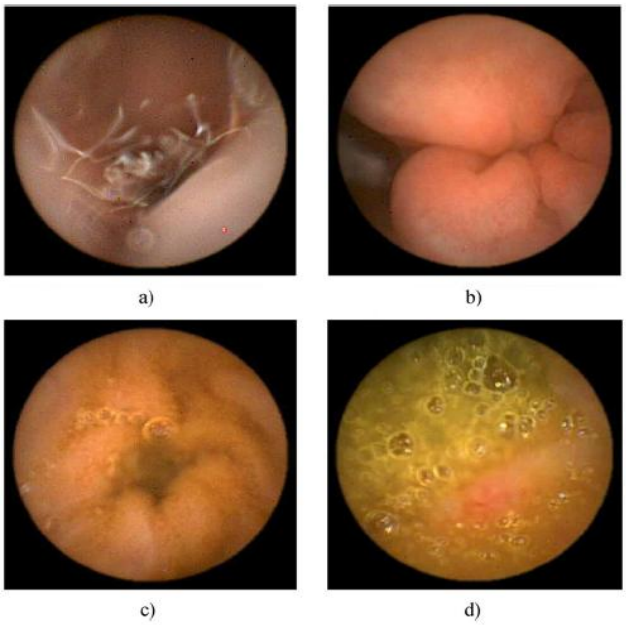

Figure 3 Examples of endoscopic capsule images [9] for the four different topographic zones: (a) esophagus, (b) stomach, (c) small intestine and (d) large intestine.

This analysis is the most time consuming and expensive part of clinical process given the large amount of time spent in a single exam, (which must, of course, be multiplied by the total number of exams reviewed by an expert and expert cost per hour fee).

2.2. Topographic segmentation of digestive tract

As an important and complex structure composed by a set of organs, the human digestive system can be segmented in different parts or areas [31], depending on the purpose of segmentation (e.g. target pathologies).

2.2.1.Segmentation methods

The most relevant topographic segmentation for endoscopy is performed by dividing the digestive tract by zones, commonly separated by entrances or valves that distinctly divide the purpose or function of each zone [31]:

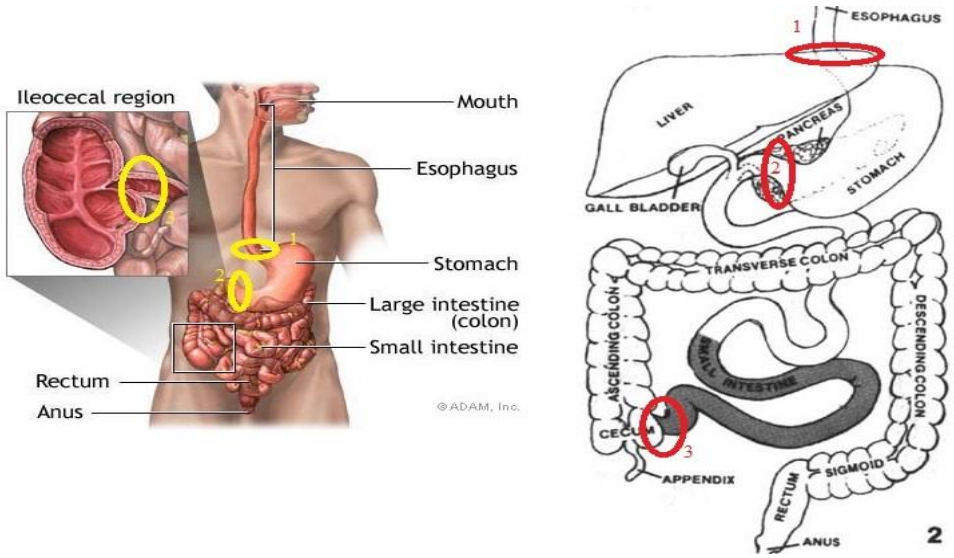

Figure 4 Identification of the three topographic barriers [32]

It focuses on the path taken by the food and disregards the surrounding organs, dividing the digestive tract into four major distinct and relevant zones to endoscopy exams:

- Zone 1, Esophagus: Between the entrance (mouth) and the Eso-gastric junction. The number 1 mark the transition from the esophagus to the stomach;
- Zone 2, Stomach: Between the Eso gastric junction and the pylorus. The number 2 mark the transition from the stomach to the small intestine;
- Zone 3, Small Intestine: Between the pylorus and the Ileocecal valve. The number 3 mark the transition from the small intestine to the large intestine;
- Zone 4, Large Intestine: Between the Ileocecal valve and the anus;

By segmenting in this way, if a certain zone is required for further analysis, the others will be withdrawn.

2.2.2. CapView and the endoscopic capsule

The CapView application [8] objective was to support medical specialists in the analysis of endoscopic capsule exams to reduce the long annotation times. There are some tools available for endoscopic capsule data analysis such as CapView or Rapid Software [33], but since CapView is a product developed in the University of Aveiro, it was the elected tool. CapView's major contribution was a multi-functional software also named CapView (from now on CapView will refer to this software when not explicit stated otherwise) capable of reviewing and creating reports about endoscopic capsule exams is the management facilities of capsule exams that were made through the years [9]. With CapView, medical specialists are now able to label video frames with specific comments which, in case of being marked as important, are added to a final report, as well as small videos related to those stored comments. The layout of CapView's tool is shown in Figure 5, where the video is displayed in a sequence of images, and a global search can be conducted.

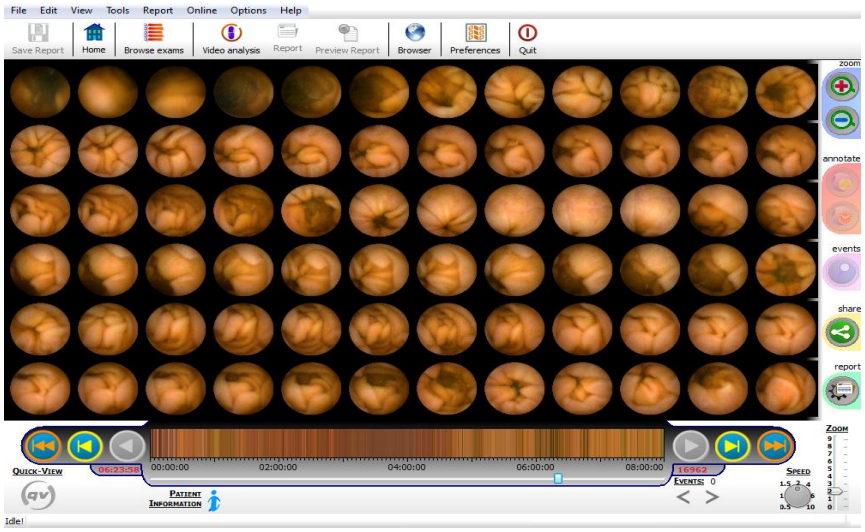

Figure 5 CapView's Layout

Another interesting contribution of CapView is the imposition of a controlled vocabulary system that is placed in the reports, which is a plus, given the different annotation nomenclatures that can be used by different medical specialists. Those annotations made by CapView are performed by automatic algorithms, created to detect specific pathologies.

CapView is also used to automatically annotate the video for the medical specialist, being capable of estimating the total transit time of the capsule and marking the detected events with the respective topographic zones.

Many other features are supported by CapView [8], such as global search, backup of the exam into server and segmentation tools [34]. In CapView is also possible to add detection algorithms capable of identifying other pathologies, which makes it an extensible tool.

One of the most important features of CapView is the topographic segmentation of the gastrointestinal tract (given the need to assign the detected events to a topographic zone), a process that takes about 15 minutes to the medical specialist if performed manually [9]. Given its importance in CapView, and for this work, further details about its implementation will be given in chapter 0.

2.3. Graphical Processing Units

Almost every computer has at least one GPU (Graphical Processing Unit), a specialized processor capable of manipulating three dimensional scenes. A scene is partitioned into triangles and a pack of images, which provides the texture (surface) for each triangle (made with specialized hardware [35]). GPUs were initially thought mainly to support graphics operations in computers namely in graphics and multimedia display (e.g. games and movies) but are starting to be more popular in performing more specific computing intensive parallel operations on data as they have the capability of being used for general computations [36] (other computations than those with multimedia or gamming purposes) as a parallel co-processor.

2.3.1. Relevance and usage of GPUs

By design, GPUs are well suited to apply the same algorithm to several datasets at the same time [37] – the SPMD (Single Program Multiple Data) parallel paradigm [38]. Image processing is a good example that is well illustrated by the computer intensive graphics present in games or multimedia applications.

Despite that fact, GPUs have been used mainly for graphical processing. Step by step, high skilled programmers matured the usage of a new parallel programming to support SPMD applications in GPUs, thus being able to scale the parallel execution code to as many processors as needed. As a result of that process, new programming languages (e.g. OpenCL [39] or CUDA Fortran [40]), APIs (e.g. Direct Compute or NVAPI [40]), standards (e.g. OpenGL [41]), tools (e.g. CG toolkit [40]) and architectures (e.g. AMD Stream [42] or

NVIDIA CUDA [43]) started to make the parallel programming and usage of GPUs more accessible to all programmers. This proliferation of GPU's usage was most welcome especially inside scientific community, which has a tremendous scope of generic and heavy processing problems to solve in an achievable time. A great number of such processing problems presents rich amount of data parallelism, which is a property that allows the performance of many arithmetic operations on data structures at the same time (some scopes may be atmospheric science [44], networking [45] or molecular modeling [46]). The medical imaging, and CapView in particular case, also fit the SPMD scenario as it involves applying the same processing algorithm in different frames in video.

2.3.2.GPUs in medical imaging

Medical imaging is one of the earliest applications for GPUs and where their impact was most felt namely in their adoption in commercial medical imaging equipment [47]. The GPUs seem to fit well the medical imaging requirements namely in data acquisition, image processing or visualization [48]).

First, a great number of algorithms in medical imaging have fewer or no dependencies between operations or between the datasets to treat, thus being easily executed by GPUs (such common operations may be filtering [49], projection [50], interpolation [51] or blending [52]). There are several applications of GPUs in medical imaging, from rendering shaders (the first flexible shader [53], volume reconstruction [54] or Programmable shader introduced [55]) to volume rendering (tomographic reconstruction [56]). This is clear when observing the substantial number of new developments every year in the field illustrated by some examples described in Table 2-1 (and available for consult in the following web site: http://www.gpucomputing.net/?q=node/55).

Workflow stage	**Applications**		
Data Acquisition	CT [57]	MRI [58]	PET [59]
Image Processing	Segmentation [60]	CT Registration [61]	Data assessm. [62]
Visualization	Enhanced Visualization [63]	Simulation [64]	Augmented Reality [65]

Table 2-1 Examples of GPU application in Medical Imaging

The endoscopy field is no exception and it has been reported new findings regarding GPU application (such as virtual endoscopy system algorithms [66] or volume rendering in sinus endoscopy [67]).

2.4. The CUDA architecture

The chosen GPU architecture for this work was the CUDA architecture (Compute Unified Device Architecture) from NVIDIA. Three reasons for such choice are:

- There is already a great amount of scientific work for CUDA (that can be consulted in http://www.gpucomputing.net/?q=og), as a result of an architecture (and free tools to use it [40]) that is available for anyone;
- CUDA-enabled computing resources were already available and at hand in our lab (appendix 0);
- CUDA API provides the required abstractions to pursue the objective of assess the use of GPU in endoscopic video analysis;

As the CUDA architecture is not universal, NVIDIA maintains a list of all GPUs CUDA enabled online [68]. For development, besides the NVIDIA toolkits installed, the CUDA software development kit (SDK) is also recommendable as it has has considerable number of useful examples.

The CUDA basic system architecture is composed by two parts, which are the host and the device. The host is the CPU (such as the microprocessors from Intel or AMD vendors) or the system that holds the GPU device composed by one or more GPUs processors. CUDA architecture follows the SPMD parallel paradigm [69]. For benefiting the most from the GPUs, the data parallel should be mainly executed using the GPUs, delegating the sequential phases of a given program to be executed in the host.

The NVIDIA CUDA solution uses the NVCC compiler [70]. As a programming language, CUDA C is an option where two different flavors exist for the host (ANSI C [71]) and for the code executed in the GPUs defined as ANSI C functions (called kernel functions) that using extends specific CUDA extensions and keywords [40]. The kernel functions (or simply kernels) can generate thousands of threads in each execution, and rely on GPUs hardware for optimized execution namely in the thread generation in few clock cycles, in contrast with CPU threads that requires thousands of clock cycles to generate and schedule them [72]. The typical

execution of a CUDA program starts with the execution of one thread in the host. At the beginning of the execution of a kernel function, a specific number of blocks and threads are defined (respectively nBlk and nTid in Figure 6). nTid means the number of threads forming each block (represented by the small boxes depicted in the Figure 6), and nBlk is the number of blocks per grid (represented by the large box), with each grid being composed by blocks of threads. At the end of kernel execution, program execution returns to the host.

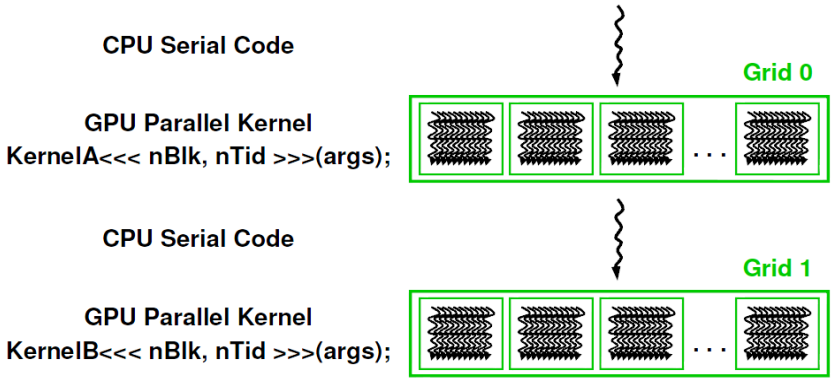

Figure 6 Execution of a program that uses the host and device [72]

A possible CUDA GPU architecture is organized into SMs (streaming multiprocessors), and each SM contains SPs (streaming processors) (as shown in Figure 7). These numbers of SMs and SPs are not fixed and will most likely vary from a generation of GPUs to another.

The threads running inside the grid are launched in the execution of a kernel and are assigned to the SMs. In the GPU example presented in Figure 7, up to eight blocks of threads can be assigned to a SM [13] (16 SMs in Figure 7, each SM with 8 SPs). When a SM finishes the execution of a block, the GPU architecture assigns a new block to be processed, thus enabling a GPU to process any number of blocks. This, along with the transparent scalability of CUDA architecture for the programmer, allows the execution of the same program in different GPUs (with different resource implementations).

Each SM as a limited number of threads that is able to run at once. For instance, NVIDIA "G80" GPU series (which is one of the lowest series of GPUs, in terms of hardware resources) supports the simultaneous execution of up to 768 threads per SM. With this limit per SM, the total number of threads that the architecture presented in Figure 7 is able to run can be found: $16 * 768 = 12\,288$ threads (with 16 as the number of SMs). This means that kernel function can be executed by over 12 000 threads.

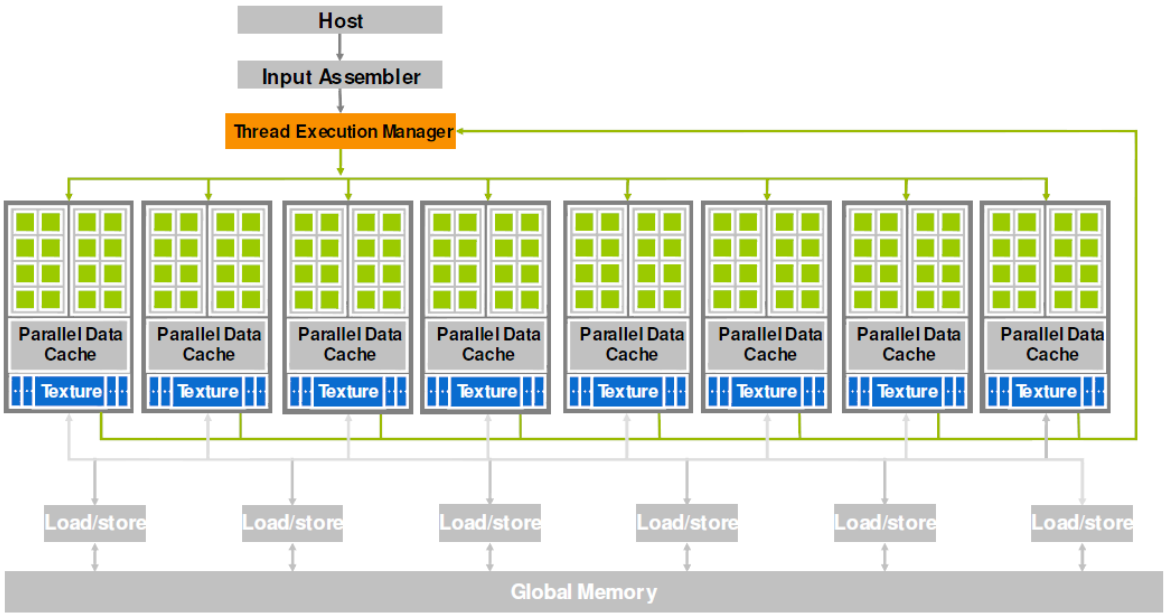

Figure 7 Logical schema of a GPU following a CUDA architecture [72]

The memory model in CUDA (depicted in Figure 8) assumes that both host and device have distinct memory spaces, which means that a manipulation of memory is mandatory (allocation and transition of data to/from device memory), prior and after to the kernel execution. This host to GPU data exchange is supported through the global memory of the GPU. The host interacts with the global memory of GPU, which is a memory that the GPU has available for interaction with CPU. Global memory is used to put values for processing, and it is used to retrieve the calculated values to the host, once computed by the GPU. Figure 8 illustrates, not only the communication relationship between host and device, but also the different types of memories that a GPU possesses.

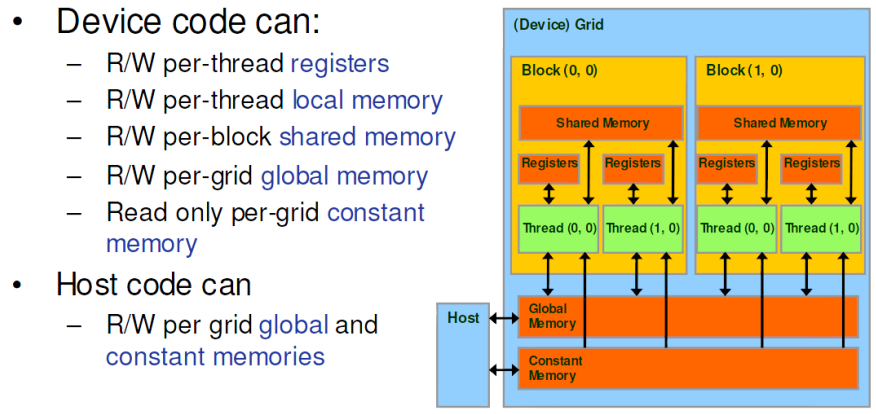

Figure 8 Streamming Processor topology [72]

As global memory is much slower, the shared memory, registers, constant memory inside the GPUs should be used in the calculations for a higher efficiency in the kernel's computation. If this is not taken into account by the programmer, the resultant application will not have an efficient execution under the GPUs and it can even have degradation in the overall execution times compared to the sequential version [73].

A more complete detailed description of the CUDA architecture can be found [13] and practical examples are available online [40].

3. CapView's Topographic Segmentation Algorithm

The implementation of CapView's topographic segmentation algorithm can be split into several phases depicted as activities in the diagram of Figure 9:

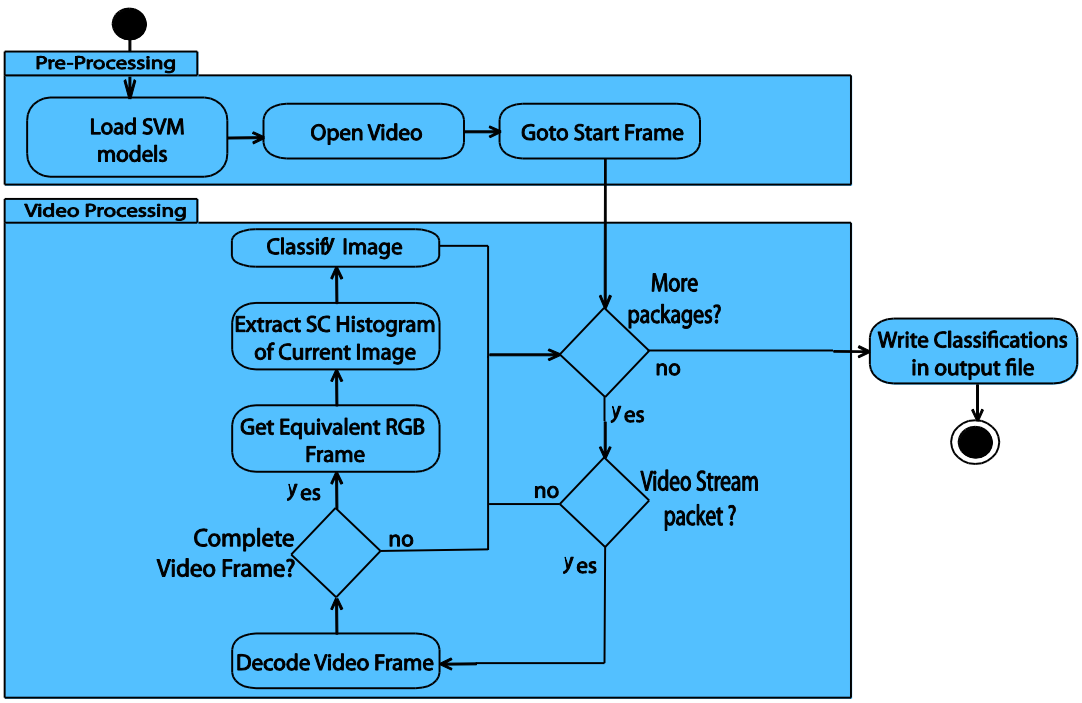

Figure 9 Activities Diagram of the Sequential Code

The algorithm is divided into pre-processing and processing phases. The classification results are stored in a file for further processing (aggregation of classification results and attribution of topographic markers).

3.1. Video Processing

Prior to any video processing and classification tasks, at the beginning of the execution it is necessary to load into memory some pre-existing data, namely the classification models and some video information.

The video preprocessing consists in opening the video of the endoscopic exams and setting up the necessary data structures for handling decoded frames. To accomplish those activities, the pre-existing CapView sequential code (or simply, CapView) uses a few libraries from the FFmpeg project[74]. These libraries are capable of recording, converting (between formats or color spaces) and streaming, as well as encoding / decoding data under many media formats (both in audio and video).

All the video processing stages are performed for each individual frame (Figure 9). For an exam of 6 hours, more than 43 000 frames would be processed. The video processing phase consists in decoding the video frame, getting the SC (Scalable Color) descriptors for the decoded image according to the MPEG-7 Scalable Color standard [75], and performing the classification. To obtain scalable color descriptors from the image to be tested, intermediary steps must be performed:

- The image color space must be converted to the HSV (Hue, Saturation and Value) color space. MPEG-7 SC standard states that the color histogram is extracted in HSV color space (H quantized to 16 bins, S and V quantized to 4 bins).
- Normalization of the SC histogram;

To improve performance, CapView skips the encoding part of the SC histogram with Haar transformation [9], as it is not mandatory. The video processing phase is described in Figure 10. At step 1, the video frame is decoded, i.e. converted from MJPEG-encoded video frame to a raw image in the YCbCr color space. In step 2, to ease the conversion process from YCbCr to HSV color space, a conversion from YCbCr to RGB is carried out.

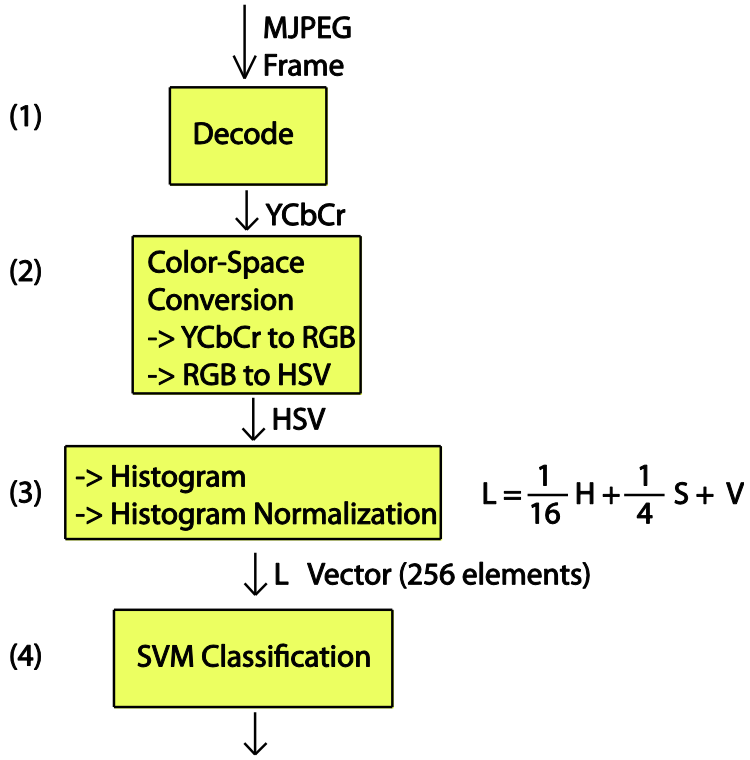

Figure 10 Video Processing Flow chart

It is in step 3 that the SC histogram is calculated. After the conversion from the RGB color space to HSV color space, for each HSV value of the image (which is a 256 x 256 pixel image), the following formula is applied:

$$L = \frac{1}{16}H + \frac{1}{4}S + V \tag{3.1}$$

The image histogram is the histogram of the L values obtained by the equation above. A normalization [76] of the histogram is performed, greatly reducing the amplitude values in the histogram bins and thus obtaining a SC histogram that represents the color pixels distribution of the [H S V] values, which will be a 256 bin histogram. The classification process is performed with SVM's over L, a method that will be described.

3.2. Support Vector Machines in CapView

CapView's segmentation algorithm is supported on a classification stage that is implemented using SVMs (Support Vector Machines). Thus, in order to be able to understand the algorithm we must understand the basic concepts of SVMs and their implementation.

SVMs [77] are commonly applied on instantiations of generic classes of problems, namely pattern recognition, regression estimation or linear operator inversion. Good examples are handwritten digit recognition [77], object recognition [78], speaker identification [79], face detection in images [80] or text categorization [81], among many others [82].

The traditional classification solution with SVMs implies a two-stage approach: a training stage to model the classification based on already classified input data and a test stage where the models are applied to classify given inputs.

At the training stage, the data already classified will be used to train the SVM models that describe the different types of possible groups that new incoming images may fit. The test stage will evaluate unknown data (new images) against those models created in the previous stage, and it will assign a classification to the tested data, deciding to which model it belongs to.

In SVM, the training stage consists in trying to find a n-dimensional space where it is possible to split the different groups represented on the data by hyperplanes. When the best hyperplane is found, it is possible to quickly map any input data into a given category (to either side of the hyperplane) through the respective mathematical expression that encodes the n-dimensional transformation and the position in relation to the plane. Such mapping occurs by calculating the distance between the input data and the created hyperplane, and it is typically the signal of that distance that dictates the classification (as illustrated in Figure 11). Depending on the methods used and on the used SVM model, this model can be linear or non-linear. The optimal hyperplane (that better splits the two classes) is found when the margin distance (between the two classes) is maximized (thus maximizing the accuracy of the classification). The process of finding the optimal hyperplane is iterative.

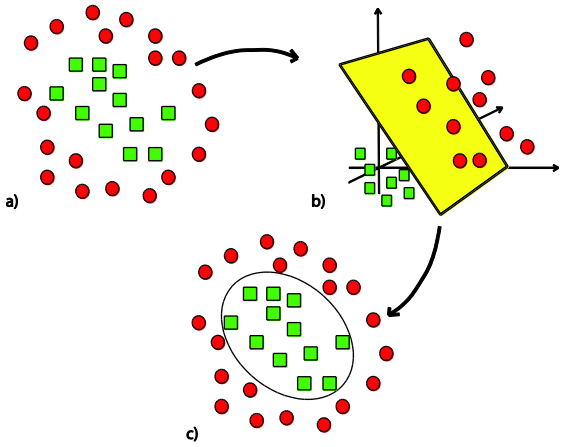

Figure 11 SVM classification – represented graphically as the boundary of squares class. (adapted from [83])

Figure 11 shows the original data (A) that is projected into a higher dimension where is possible to find a plane that splits both classes (B); Having found this plane (B) it is possible to project back this plane into lower dimension and classify the data in the original dimension (C). The mechanism that guides the parameterization for each iteration (e.g. margin, window, dimension, error tolerance), and the establishment of the stop conditions for the process (i.e. a optimal solution found or max number of iterations reached), are outside the scope of this dissertation and details can be found elsewhere [84].

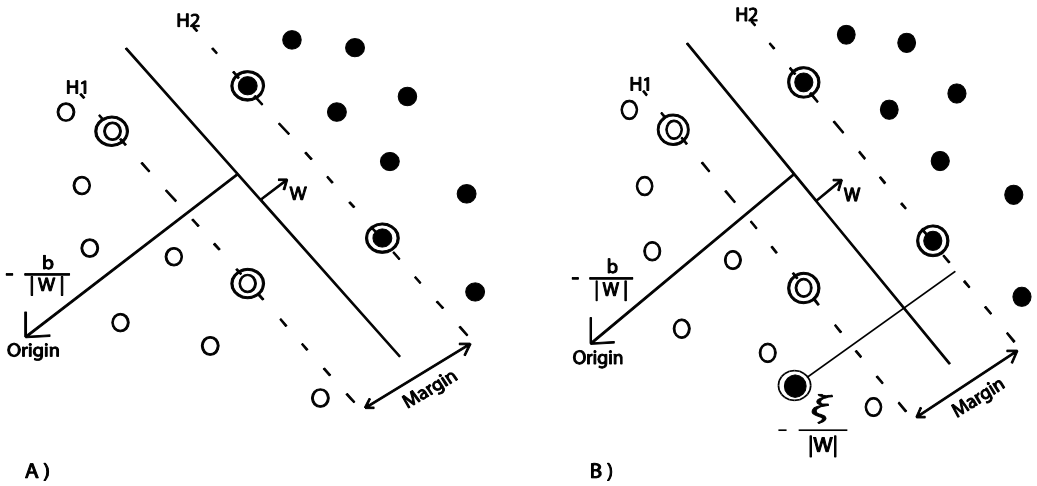

Figure 12 Linear separating Hyperplanes (adapted from [12]) for:

A) separable clase and B) non-separable case.

H1 and H2 (Figure 12) are two parallel hyperplanes that divide the two classes, without either one of the training points being between them. H1 and H2 are defined in each iteration, with each one representing the shortest distance between a hyperplane and each one of the

classes. The most likely candidate to be the optimal hyperplane is gradually placed between them (being equally parallel and at the same distance from both hyperplanes). The training points pictured with a circle around them are the support vectors that define the computed hyperplane. If any of them got removed, the candidate hyperplane could not be equal to the previously defined. Given such property, such training points constitute the support vectors for that hyperplane.

3.3. CapView SVM classifier

CapView's SVM models [9] are supported on a polynomial inhomogeneous kernel [85] (which was applied in both training and test phases), since its ratio of classification errors and performance is better, compared to other evaluated kernels. For performing the topographic segmentation, CapView considers four classes, each one describing a distinct topographic zone. Analogously to the cases where only two classes are considered, a specific SVM model was obtained for each of the 4 distinct areas:

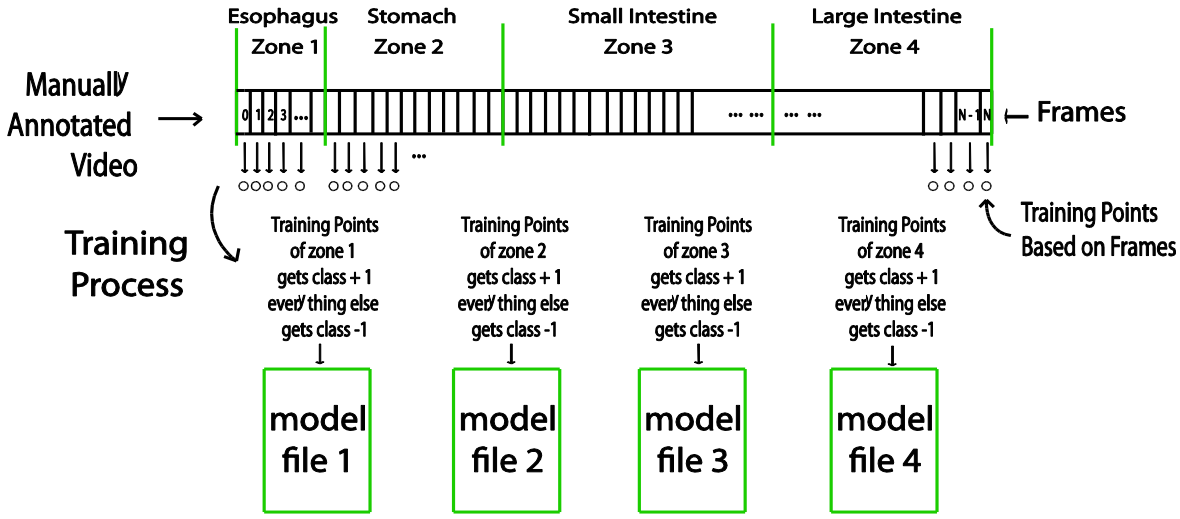

Figure 13 Training stage of CapView's Application

The immediate implication of that is the mandatory consideration of all 4 models for the classification computation. When the calculated descriptor is positive, the image to be classified is considered to be in the zone associated with the respective SVM model, or not, in case its value is negative. To obtain the final classification we must evaluate each of these four descriptor values ($D_1(z)$, $D_2(z)$, $D_3(z)$ and $D_4(z)$ as in Figure 14). This combination is obtained by selecting the model that presents the highest descriptor value that will attribute the

final classification to the test subject. That, in turn, will be the final classification corresponding to the test subject's topographic zone. Normally, only one of the descriptors is positive, because the test subject can only belong to one class (representing a zone).

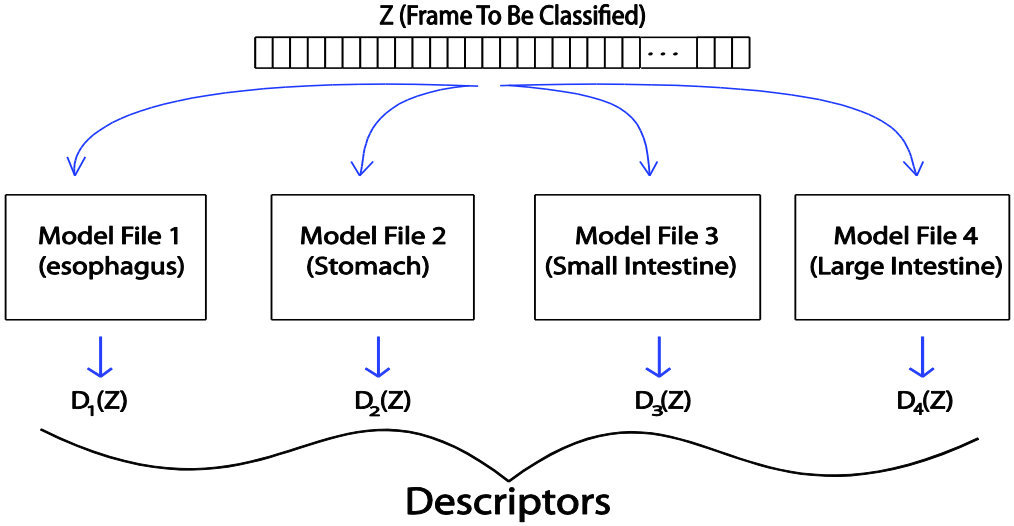

Figure 14 Classification scheme

In fact, the descriptors mentioned earlier are calculated based on the distance between the test subject (z) and each zone statistic descriptors (file models). This distance is calculated by finding the signal of $sgn(w \,.\, z + b)$ equation [86], or simply, finding in which side of the hyperplane the test subject belongs to.

Due to the use of the polynomial inhomogeneous kernel in the training process, the distance calculation is then calculated for each model file with:

$$D_m(z) = b_m + \sum_{i=1}^{l_m} [z \,.\, x_m(i) + 1]^3 \, \alpha_m(i) \tag{3.2}$$

Where $[z \,.\, x_m(i) + 1]^3$ is the Kernel responsible for the inner product calculation (between test subject and every training point inside model files), $\alpha_m(i)$ are the weights (or the relevance) of the training points $x_m(i)$ and b_m the threshold.

Such calculation produces four values ($D_1(z)$, $D_2(z)$, $D_3(z)$ and $D_4(z)$, in Figure 14), which represents the four intermediary classifications for each class referred earlier (one class for each zone description).

3.4. Classification's Aggregation and Topographic Markers

The classification algorithm is applied in each individual frame, more precisely to the video frame SC histogram that, in the process, is mapped to a specific topographical location (TL), using a numerical code described in [9]:

$$TL(z) = \begin{cases} 1\ (z \in esophagus) & if\ D_1(z) = \max\{D_i(z)\} \\ 2\ (z \in stomach) & if\ D_2(z) = \max\{D_i(z)\} \\ 3\ (z \in small\ intestine) & if\ D_3(z) = \max\{D_i(z)\} \\ 4\ (z \in large\ intestine) & if\ D_4(z) = \max\{D_i(z)\} \end{cases}$$

3.4.1.From topographic locations to segmentation

To obtain a segmentation of the digestive tract, from the individual frame topographic location, a further step consisting in an aggregation must be done. This aggregation estimates the location of all barriers that divides the four topographic zones:

- Barrier Z_{12}: Divide zone 1 (Esophagus) and zone 2 (Stomach);
- Barrier Z_{23} Divide zone 2 (Stomach) and zone 3 (Small Intestine);
- Barrier Z_{34}: Divide zone 3 (Small Intestine) and zone 4 (Large Intestine);

The aggregation process estimates the barrier positions based on both the individual frame classifications and on temporal distributions of each zone, based on the analysis of several exams as described in [9]. In this paper produced by J.P. Cunha, *et al.*, charts with temporal positions (regarding the classifications distribution probability through time) for each topographic zone are presented. Besides the characterization of the temporal distribution, those charts allow the establishment of maximum ranges, which dictate where the transition from a zone x to a zone y has to occur.

As a result, for every barrier that needs to be found, the search scope for that barrier is limited (i.e. it must occur between a concise range of frames). After that delimitation, the barrier Z_{xy} (marking the transition between zone x and zone y) is found in the index where the TE value is the lowest. In that process, formula (3.3) is used:

$$TE = TE + \sum_{i=1}^{N} [\, E_1(t_i) + E_2(t_i)] \tag{3.3}$$

For every Z_{xy}, $E_1(t_i)$ and $E_2(t_i)$ are:

$$E_1 = \begin{cases} 0 & , \quad if\ TC(t_i)\ equals\ y \\ -1 & , \quad otherwise \end{cases} \quad , \quad E_2 = \begin{cases} 0 & , \quad if\ TC(t_i)\ equals\ x \\ +1 & , \quad otherwise \end{cases}$$

Being $TC(t_i)$, the topographical classification of the frame t_i.

The TE value varies with the consideration of errors $E_1(t_i)$ and $E_2(t_i)$ for each frame t_i, being the lowest value of TE that marks the position of the barrier Z_{xy}. The initialization of TE is relevant, and an example will be provided in the following section to better explain the overall process of finding the topographic barriers.

3.4.2. Aggregation example

To demonstrate how the topographic barriers are found, the following example will be given. In spite of being a simple example, it is closely similar to what actually happens in CapView's application. Figure 15 illustrates possible classifications of a 25 frames video, represented as a row major array:

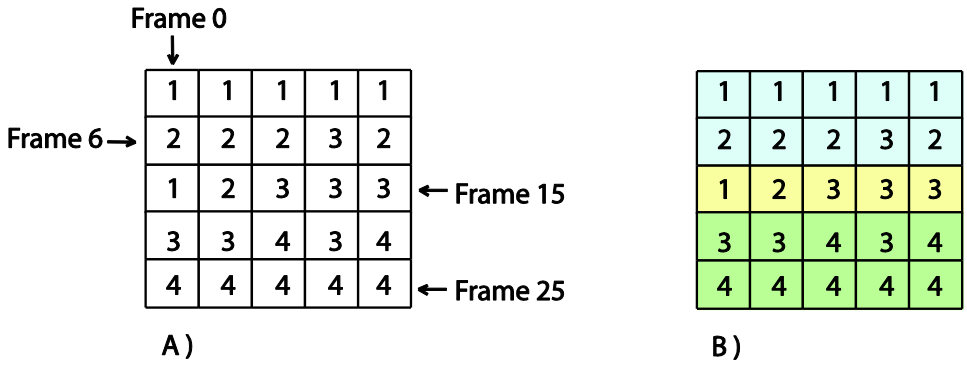

Figure 15 Video Example Classifications

After calculating the frame's classification of the video example (performed by CapView and presented in A of Figure 15), the maximum ranges to be considered, were illustrated with different colors in picture B) of the same figure. Those maximum ranges are merely illustrative, but the maximum ranges that CapView's sequential code uses are based on the probabilities discussed in the paper mentioned above. Such ranges mean that:

- Zone 2 have to begin between the first frame and frame 10;
- Zone 3 have to begin between frame 11 and frame 15;
- Zone 4 have to begin between frame 16 and the end;

Before the computation of the barriers using the equation (3.3), the initial value of TE (representing the initial error) must be set up. This initialization is done by counting, in each maximum range (where the barrier Z_{xy} needs to occur), the number of classifications different from y:

Initial Error = 6

Frame n.º	$E_1(T_i)$	$E_2(T_i)$
1	5	5
2	4	4
3	3	3
4	2	2
5	1	1
6	1	2
7	2	3
8	3	4
9	3	4
10	4	5

Z_{12}

Initial Error = 2

Frame n.º	$E_1(T_i)$	$E_2(T_i)$
11	1	2
12	1	1
13	1	2
14	2	3
15	3	4

Z_{23}

Initial Error = 3

Frame n.º	$E_1(T_i)$	$E_2(T_i)$
16	2	2
17	1	1
18	1	2
19	1	1
20	1	2
21	2	3
22	3	4
23	4	5
24	5	6
25	6	7

Z_{34}

Figure 16 Topographic Barriers Estimation

Figure 16 shows the variations that TE receives after the evaluation of the errors 1 and 2 for each maximum range (column two and three of each table, respectively), as well as TE initial value for each barrier. Therefore, the minimum values that TE ever had (for each case), which are marked with a different color, dictates the position where each barrier will be placed (Z_{12} in frame 5, Z_{23} in frame 12 and Z_{34} in frame 19).

3.5. Example of the Execution Flow

In this section we will illustrate the steps of CapView's segmentation algorithm over a frame example (Figure 17). The video processing stage represents the core of CapView's algorithm (as depicted in Figure 10), and so, this example will emphasize its stages. Like in the following example, the process starts with individual decoded video frame, obtained from a capsule endoscopy exam:

Figure 17 Decoded Video Frame example

Figure 17 is a decoded frame in the YCbCr color space, and the image conversion between color spaces (from YCbCr to RGB) is the next step. As the decoding and color space conversion are steps made at the data representation level, the resulting frame is not visually different from Figure 17. After that, the calculation of the Scalable Color histogram needs to be done. This requires the calculation of the histogram of the L values obtained by the equation 3.1, with the result being shown in Figure 18 for the given image:

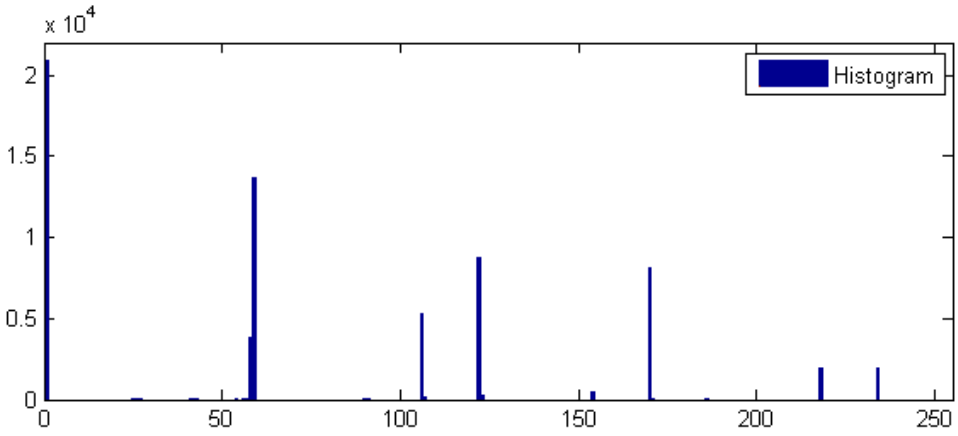

Figure 18 Histogram of Video Frame example

As can be observed in the histogram above, the image has a great amount of black color, along with other sparse values that represent the distribution of specific colors. As can be observed, this histogram is sparse and presents values with disparate values (e.g. high values and values that can barely be seen in the histogram). Using a normalization process, this fact can be corrected (small values enhancement and high values reduction, as illustrated in Figure 19). The following histogram corresponds to the SC histogram, or color descriptor of the given image (according to MPEG-7 Scalable Color standard):

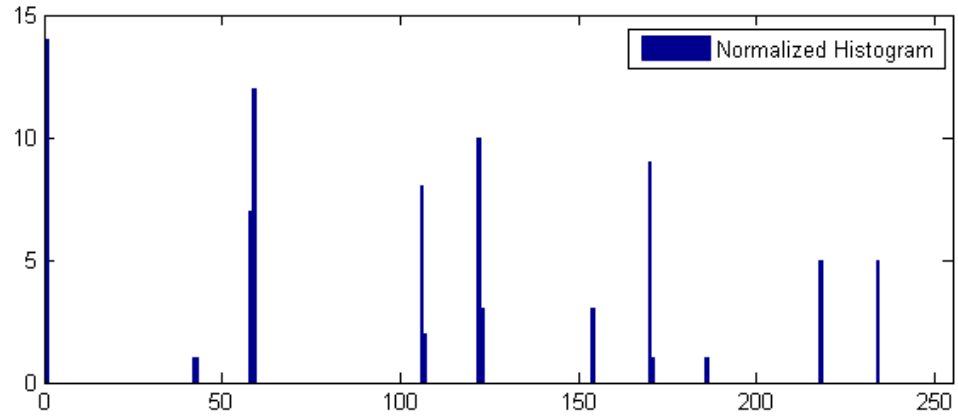

Figure 19 Normalized histogram

At this point, the classification of the given image can be performed by using the calculated SC histogram and the model files. Performing the classification illustrated in Figure 14, the values obtained for the letters $D_1(z)$, $D_2(z)$, $D_3(z)$ and $D_4(z)$ of the given image are:

$$D_1(z) = -2.38767$$
$$D_2(z) = 2.65937$$
$$D_3(z) = -2.60438$$
$$D_4(z) = -2.37746$$

These values represent an SVM classification for each model file. In this example, the letter $D_2(z)$, calculated by the model file number 2, presents the higher value, which means the model file number 2 is the one describing the zone belonging to the given image (i.e. the stomach).

According to the values representing the topographic locations (defined at the beginning of the previous section, as $TL(t)$), the classification of the given image will be $t = 2$, because $z \in \{stomach\}$.

4. Parallelization of the Segmentation Algorithm

In this chapter, different GPU parallelization approaches of CapView's segmentation algorithm will be explored. Some pre-parallel optimizations were carried out, thus enhancing the code's efficiency and avoiding unnecessary parallelization efforts. In addition to those parallelization approaches, the usage of multiple GPUs will be explored for the best algorithm developed. At the end, the results of these experiences will be confronted with other existing advanced computing solutions, such as Grid computing [87] and cluster computing [88].

From this point onwards when performing execution time measurements, we will use three examples of WEC videos produced in clinical routine, as presented in Table 4-1 (values in hh:mm:ss: hours, minutes and seconds respectively):

Video	Length	Size (MB)	Number of Frames	Size per image	Codec	Frequency
1	02:56:00	205	21204			
2	06:30:00	409	46913	256 x 256 pixels	MJPEG	2 fps
3	08:07:00	610	58467			

Table 4-1 Test Videos

We selected 3 videos with different lengths to investigate the effect of size in overall execution times. Video 1 in the shortest; Video 2 is of normal length (between 41000 and 55000 frames); Video 3 is the longest. All times from now on is the average of 5 runs (except when stated otherwise).

4.1. Data Alignment in Memory

Before attempting to parallelize a given algorithm, it is good practice to try and optimize its sequential implementation, as any inefficiency would be magnified as a parallel operation[89]. To observe this rule, the initial task consisted in a detailed analysis of the sequential CapView segmentation code (or simply, CapView, for now on) currently in use (described in the previous chapter).

The main outcome was a more efficient memory access:

- Data storage structures were re-organized so that operations could involve only the data strictly necessary for classification;
- This data was stored in contiguous memory locations;

Each model file is represented by an l_m amount of x_m 256-element vectors, as illustrated in Figure 20. Each x_m has a coefficient $\propto_m$ (explained in equation 3.2).

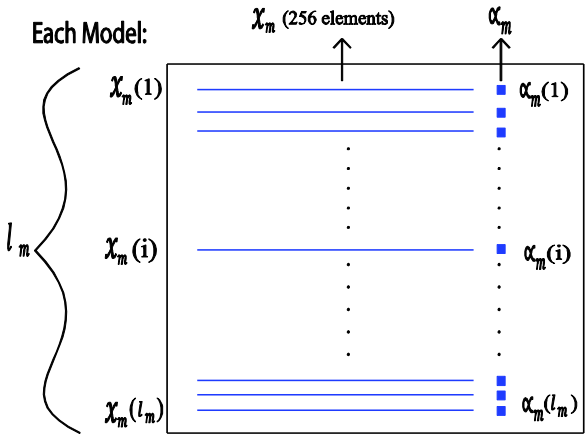

Figure 20 Model File

The l_m values are defined in Table 4-2:

Model Files	l_m
1	830
2	8441
3	17481
4	13816

Table 4-2 Lm values for all model files

This mandatory information, presented in each model file, was being stored in several data structures. Those data structures were stored non-contiguously in memory, which reduced access efficiency. There was a substantial latency in the memory accesses, since 256 memory accesses were needed for complete retrieval of each x_m vector.

One way of avoiding this amount of memory access latency was to rearrange the data in a contiguous way, thus retrieving every needed x_m vector with only one memory access. For example, the Figure 21 shows how the support vectors for a single model file became stored, with this new data alignment:

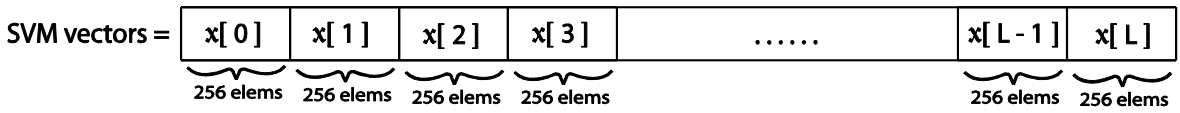

Figure 21 Support vectors stored in memory (per model file)

This kind of organization directly affects the execution times for the "Load SVM Models into Memory" and "Classify image" activities (see Figure 9). The overall execution times of CapView were measured for the test videos (Table 4-1), before and after this data alignment without GPU usage. The machines used for the tests are specified in appendix 0:

	Machine 1			Machine 2		
Videos	Non-Optimized	Optimized	Performance gain (%)	Non-Optimized	Optimized	Performance gain (%)
1	00:40:32	00:30:42	24,27	00:20:14	00:17:30	13,33
2	01:29:00	01:08:00	23,86	00:44:42	00:38:33	13,64
3	01:52:00	01:25:00	23,93	00:55:51	00:47:51	14,31

Table 4-3 Overall Speedup

Table 4-3 shows the mean execution times of CapView for a number of runs, before and after the data alignment consideration, as well as the obtained speedups. The speedups were calculated by applying the following relation: $performanceGain = \frac{|B-A|}{B}$ (with A (after) and B (before) as the execution times of different versions).

By considering the performance gains, it can be concluded that the overall execution was improved (almost 25% and 15% for, respectively, machines 1 and 2). The discrepancy of speedups between the two machines can be explained by different memory access latencies.

4.2. Amdahl's Law

Amdahl's law [90] is important in this context because it quantifies the maximum theoretical overall speedup that can be achieved in one application, by parallelizing a fraction of the program's execution.

Let's consider Figure 22 as the execution of a generic program, where T_1 is the total execution time for a single processor (with $T_1 = 1$):

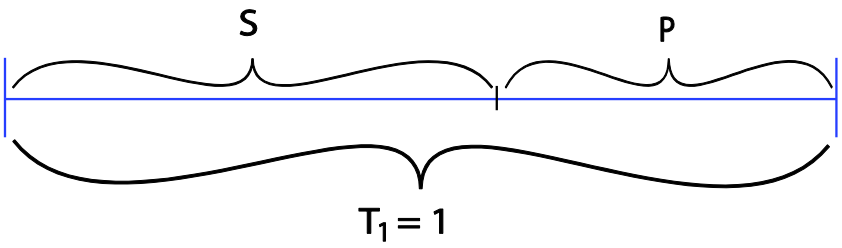

Figure 22 Generic program's execution

From the execution identified as T_1, S identifies the part that is exclusively sequential, while P identifies the part that can be executed in parallel by more than one processor. Knowing that

$$T_1 = S + P \tag{4.2}$$

For n processors, equation (4.2) now becomes:

$$T_n = S + \frac{P}{n} \tag{4.3}$$

Amdahl's law then becomes:

$$S_n = \frac{T_1}{T_n} = \frac{1}{S + \frac{P}{n}} = \frac{1}{(1-P) + \frac{P}{n}} \tag{4.4}$$

S_n is the theoretical overall speedup that the application experiences, by executing the P part with n processors. The equation (4.4) also implies two things:

- If P is small, overall speedup may not worth the parallelization effort;
- No matter how large n is, overall speedup can never exceed $\frac{1}{1-P}$;

CapView was tested to determine the execution time of each segment.

Videos	Decode	YcbCr->RGB	Hist. Ops	Classifications	Others
1	0,0051%	0,0001%	6,0766%	93,6438%	0,29%
2	0,0051%	0,0001%	6,0896%	93,8448%	0,14%
3	0,0051%	0,0001%	6,0908%	93,8625%	0,12%

Table 4-4 Major Activities of CapView's Execution

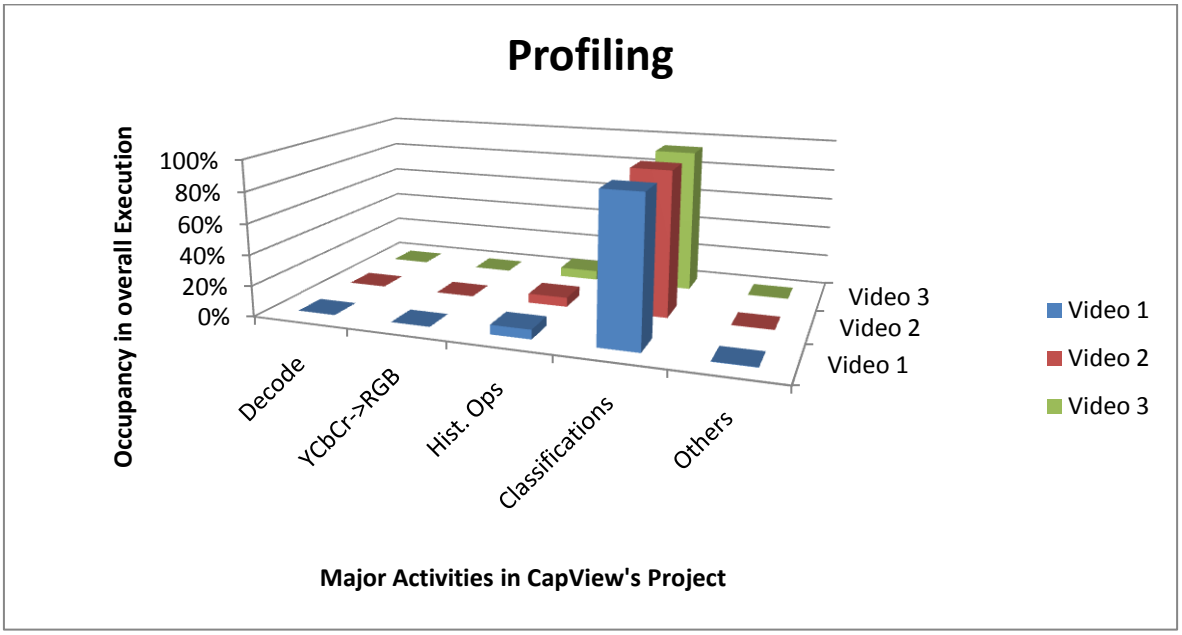

Figure 23 Profiling of CapView's Sequential Code

The resulting profile is presented in Table 4-4 and depicted in Figure 23 (one chart per video). "Others" integrates all the activities considered as strictly sequential (i.e. that can't be parallelized); these do not reach even 0.5% of the overall execution time. The remaining activities are parallelizable. However, both "Decode" and "YCbCr to RGB" take a negligible amount of the overall execution time. SC Histogram calculation takes longer, but it is in "classifications" that CapView spends over than 93% of its computation time, which makes it the best candidate for parallelization according to Amdahl's law (see Figure 24, which shows the result of applying Amdahl's law, considering only one of the activities parallelized).

Figure 24 is obtained by varying the number of cores (processors) in formula (4.4). There would be almost no gain in the parallelization of the first three fractions. There is, however, a huge potential speedup in the classification, with a maximum gain of a just over 15 in the classification's performance. With a closer look at the table presented in appendix 0, the maximum number of cores used in both GPU's (one for each machine) can be identified: Machine 1 uses 8 cores and machine 2 uses 240 cores.

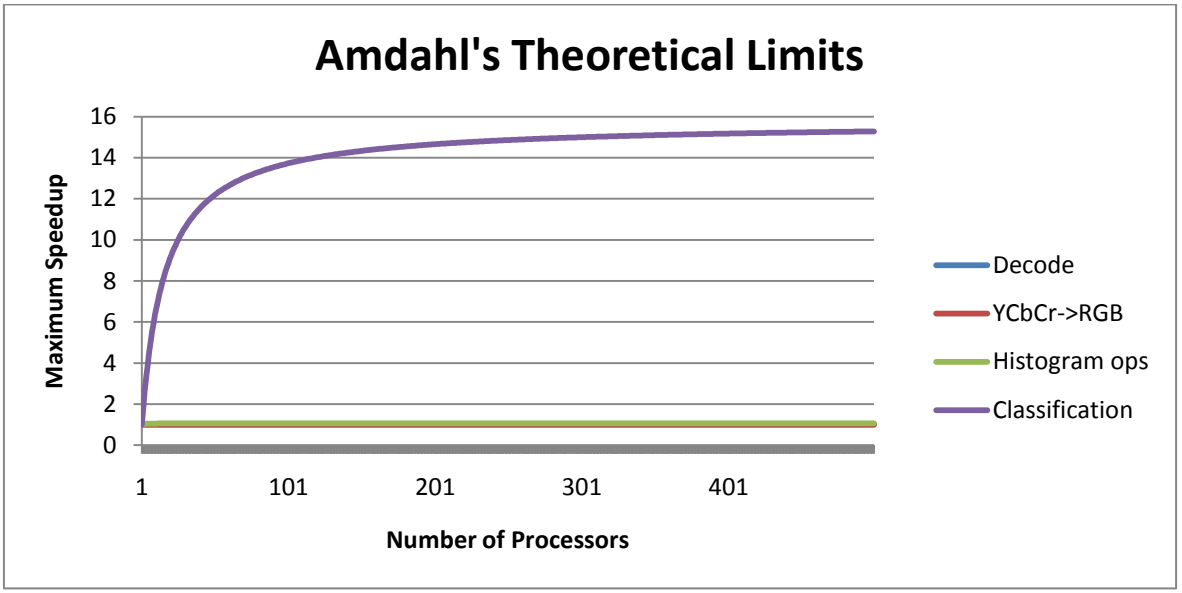

Figure 24 Amdahl's theoretical limits

With these particular values, the overall speedup formula was applied, in order to find the exact maximum theoretical values for this amount of processors. The results are specified in Table 4-5:

Machine	n (cores)	Decode	YCbCr->RGB	Histogram Ops	Classifications
1	8	1,000044662	1,000001	1,05615574	5,53658
2	240	1,00005	1,000001328	1,0644101	14,8227478

Table 4-5 Maximum theoretical speedups for machines 1 and 2

Parallelization's effort may only be justifiable in classification, reaching, theoretically, almost 15 times in overall speedup for the second machine (240 cores), and over 5.5 times in overall speedup for the first machine (8 cores).

Next, we will present three different approaches we used for parallelizing the classification: using CUBLAS, our initial approach and the independent thread approach.

4.3. Parallel Code Implementation and Evaluation

Three different approaches were made, as attempts to evaluate the best outcome for parallelization: CUBLAS approach, initial approach and independent thread approach. For each approach, methodologies are discussed, as well as the respective performances and limitations. A second profiling will be performed to assess if further parallelizations or

optimizations may be profitable. Apart from implementations, for the best approach, it is evaluated the overall impact of multi-GPUs utilization, instead of a single GPU usage.

As last consideration, it is compared the CapView's application execution in a CUDA parallel architecture with other existent parallel environments, such as Grid and cluster computing. By doing so, a better perception of the GPU's suitability in this specific problem is given, as well as the best parallel solution that CapView's application can benefit.

4.3.1.CUBLAS Approach

This approach explores the CUBLAS library [91], which is a library that implements BLAS subroutines (Basic Linear Algebra Subprograms) on top of CUDA architecture, taking advantage of its power implementation. BLAS [92] is a standard with the purpose of executing basic linear algebra operations, such as multiplications of vectors or matrices. These kinds of operations are broadly used in high-performance computing, as they are extremely efficient and have a general purpose. There is no need to interact directly with CUDA's API, because CUBLAS routines do it for the user.

CUBLAS was used to calculate the dot product between each frame SC histogram and the support vectors. By doing so, this calculation is performed inside GPUs, instead of doing it sequentially inside the host (GPU's host). Every time a classification is needed, a dot product is performed between the frame SC histogram and every support vector in a model file (for all the model files). The best way to delegate this work to CUBLAS is to make a dot product in a matrix × vector way, with the result being a vector full of scalars. The CUBLAS method, allowing such matrix × vector operation, is the cublasSgemv [91], which is able to calculate the inner product between the vector and each row of the matrix, according to the function:

$$y = alpha \times op(A) \times x + beta \times y, \qquad where\ op(A) = A\ or\ op(A) = A^T$$

After this major dot product, the rest of the equation (3.2) will be computed in the host for each element of the resultant vector (computed in the GPU), as illustrated in Figure 25:

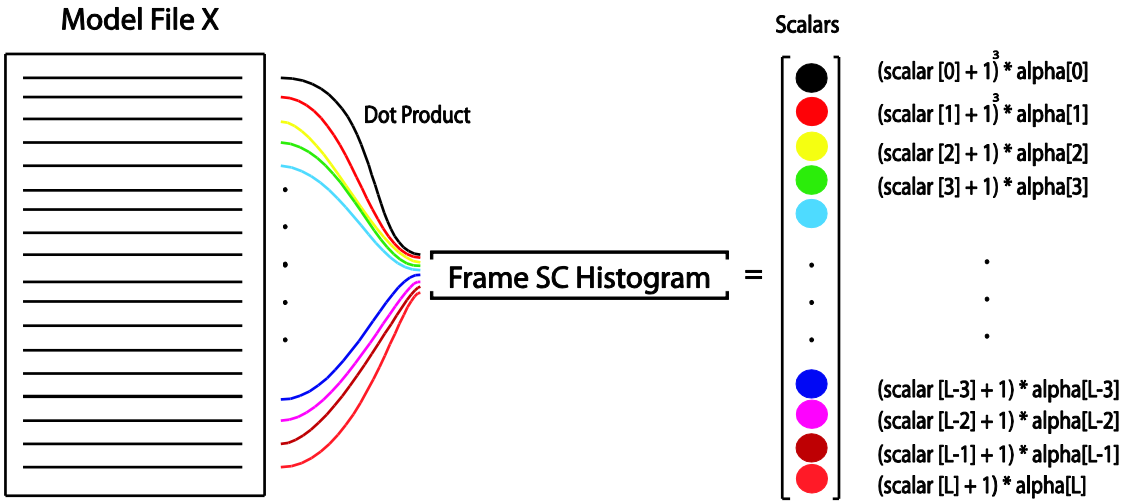

Figure 25 Cublas Scalar Vector Calculation

The execution time means for the classification activity for each frame in both versions (sequential version as VS and CUBLAS approach as A1) were measured for a substantial number of runs (over 20 000). All the values in the Table 4-6 are from both machines and are presented in milliseconds:

	Machine 1	Machine 2
Per frame classification (ms)	Classifications	Classifications
SV	81,50842	46,37788
A1	57,28418	4,610936

Table 4-6 Sequential Reference Vs. CUBLAS Approach

To quantify the improvement (or degradation) shown, the speedup for the classification fraction in both machines was measured:

Machine 1:

$$SpeedupClassifm1 = \frac{SV}{A1} = \frac{81{,}50842}{57{,}28418} = 1{,}4228784$$

Machine 2:

$$SpeedupClassifm2 = \frac{SV}{A1} = \frac{46{,}37788}{4{,}610936} = 10{,}0582355$$

The overall speedup can be measured in the following way:

$$overallSpeedup = \frac{1}{\sum_{i=1}^{N_{fractions}} \frac{l_i}{s_i}} \quad (4.5)$$

In formula (4.5), s_i is the speedup that the fraction i received, and l_i is the time of the fraction i in percentage, with $l_i = \frac{L_i}{\sum_{i=1}^{N_{fractions}} L_i}$ (L_i as the time of the fraction i in units of time). Table 4-7 shows the overall speedup for each one of the considered videos, after the application of equation (4.5), as well as the global execution times of the topographic segmentation algorithm:

Videos	Machine 1			Machine 2		
	Non-Parallelized	Parallelization approach	Overall Speedup	Non-Parallelized	Parallelization approach	Overall Speedup
1	00:30:42	00:22:36	1,385369778	00:17:30	00:02:42	6,377499552
2	01:08:00	00:49:58	1,385252189	00:38:33	00:05:57	6,424672593
3	01:25:00	01:02:21	1,385375320	00:47:51	00:07:24	6,431736741

Table 4-7 CUBLAS approach's Overall Speedups

The overall speedup indicates that this approach makes the CapView's application execution run over six times faster in machine 2, and about 1.38 times in machine 1. This means that the CUBLAS attempt to parallelize CapView's sequential code has proved successful with an overall improvement in relation to the sequential version of the segmentation algorithm. However, CUBLAS model implies that some operations must be performed in the host and not in GPUs, as its API only provides limited management of kernel functions. As illustrated in Figure 25 the right side of the scalars vector has to be performed in the host instead of being performed inside the GPU. The computation of the sum on formula 3.2 that combines parcels of vector has to be performed in the host for each scalar produced by CUBLAS, which implies a slightly degradation of the overall classification speedup.

4.3.2.Initial Approach

The 2nd approach concentrates in placing most of the classification effort (formula 3.2) in the GPU side, by creating a custom kernel for the GPU. To maximize the contribution of the GPU power (in the classification image process), the next steps were performed:

- In the beginning of CapView's execution, all support vectors from all model files are copied to GPU memory. By doing so, every time that a support vector is needed for a calculation, it can be retrieved directly from GPU's global memory;
- After this step, every time a classification is performed, the following is done:
    - Z (frame SC histogram from formula 3.2) is copied into the GPU memory;

- The classification is performed in a parallel way, by using Z and every support vector (x_i, from formula 3.2) from model files;
- Results are copied back to host;
- The values produced in the GPU are summed and the threshold amount (b, from formula 3.2) is subtracted in the sum result;

For a better understanding of the process, Figure 26 schematizes classification's parallelization:

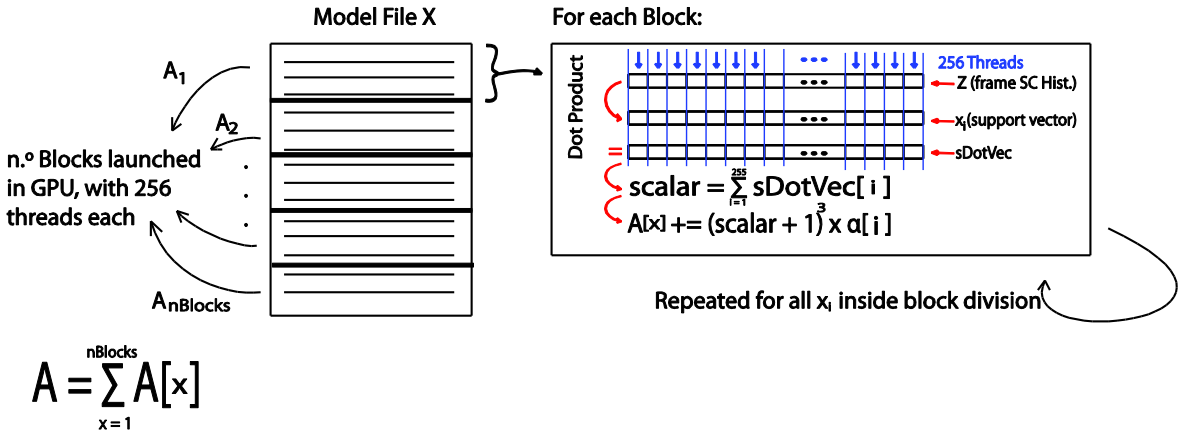

Figure 26 Initial Classification approach scheme

Figure 26 shows that the model files are divided in a way that each block (launched in GPU for the kernel execution) becomes responsible for a small amount of support vectors, with each block being composed by 256 threads. As illustrated in the Figure 26, each thread will be responsible for computing a part of the dot product between Z (the SC histogram of the image to be classified) and each x_i (support vector), delegated to that block. When each thread finishes the dot product, a vector reduction is done, in order to form the final scalar of the dot product. This vector reduction is performed accordingly to the most optimized parallel reduction created by NVIDIA [89]. When the scalar is found, a partial classification (between Z and a single x_i) is calculated and added to a partial sum variable ($A[x]$). After the repetition of this process for all the delegated support vectors in a block, the block saves the sum variable in global memory and returns. The final classification value for the specific model file is found after:

- Fetching these partial sums (created from each block) from GPU memory;
- Adding all partial sums: $A = \sum_{x=1}^{nBlocks} A[\,x\,]$ in host;

- Subtracting threshold value from the classification formula (3.2) in the host;

In order to observe real impact of this approach, the execution time means for the classification activity only of each frame in both versions (sequential code as SV and Custom approach as A2) are shown. All the values in the Table 4-8 are from both machines and are presented in milliseconds:

	Machine 1	Machine 2
Per frame classification(ms)	Classifications	Classifications
SV	81,50842	46,37788
A2	272,2713	57,09392

Table 4-8 Sequential Classification vs. Initial approach

To quantify the degradation, the speedup for the classification fraction is measured:

Machine 1:

$$SpeedupClassifm1 = \frac{SV}{A2} = \frac{81{,}50842}{272{,}2713} = 0{,}2993647145$$

Machine 2:

$$SpeedupClassifm2 = \frac{SV}{A2} = \frac{46{,}37788}{57{,}09392} = 0{,}8123085611$$

As done in the 1st approach, the overall speedup was measured with the equation (4.5). The result is the following table, which shows the impact of this approach in CapView's application, as well as the global execution times of the topographic segmentation algorithm:

	Machine 1			Machine 2		
Videos	Non-Parallelized	Parallelization approach	Overall Speedup	Non-Parallelized	Parallelization approach	Overall Speedup
1	00:30:42	01:38:38	0,313304451	00:17:30	00:21:34	0,822023462
2	01:08:00	03:37:56	0,312779007	00:38:33	00:47:38	0,8212654
3	01:25:00	04:31:29	0,312739618	00:47:51	00:59:27	0,821245614

Table 4-9 Overall Speedups, in the Initial approach

This means that the impact on the performance of the 2nd attempt to parallelize CapView has slightly degraded in machine 2 and highly degraded in machine 1, compared to the sequential execution. However, this attempt identifies some issues preventing a good performance in the GPU classification, like a huge amount of dependencies between threads. In each partial classification (between Z and a single x_i), each thread contributes to the dot product, however, it has to wait for the other threads to accomplish the same; after that, the vector reduction is made and, once again, while some threads are doing some work, others

have to wait. Even the final calculation ($(scalar + 1)^3 \times \alpha[\, i\,]$) is done by only one thread (for consistency issues), out of the 256 threads available in each block.

Without such dependencies between threads, this 2nd attempt could have a positive impact over the overall sequential execution time.

4.3.3. Independent Thread Approach

The objective of the last approach is to parallelize the classification fraction, in a way that dependencies among threads (per block) are minimized.

Figure 27 gives an overview on how parallelization based on threads was done of the overall process of classification. Each model file will be partitioned and delegated to blocks of 256 threads, in order to delegate an entire support vector per thread. By doing so, each thread is now able to perform an entire dot product between the frame SC histogram and its own support vector, and proceed to the partial classification, without a single dependency.

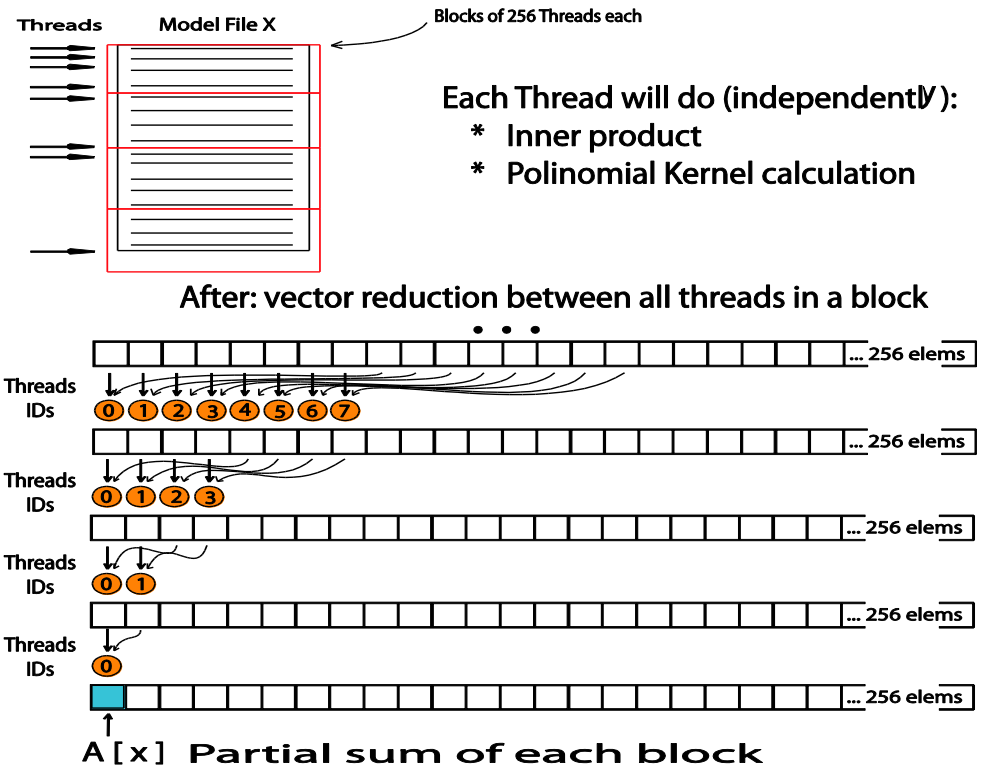

Figure 27 Independent Thread approach

After this step, there is a thread synchronization that guarantees that each of the 256 threads of a block calculated a partial classification. The next step consisted in data gathering (reduction) of the resultant partial classifications vector, where results were added incrementally along each calculation in the GPU values:

$$A = \sum_{j=1}^{sv_{num}} partialClassif[\,j\,]$$

The reduction was similar to the one used in the previous approach, but instead of reducing each dot product to find the scalar value (as done in the previous approach), in this approach, the reduction will drastically diminish the amount of partial classifications that the host will have to sum (a reduction from thousands of values to one value per block launched):

$$A = \sum_{x=1}^{nBlocks} A[\,x\,]$$

To maximize this solution we also optimized the memory access. The original memory access pattern from the threads is not efficient because it is not coalesced [13], that is, the memory accesses are not contiguous, in the GPU:

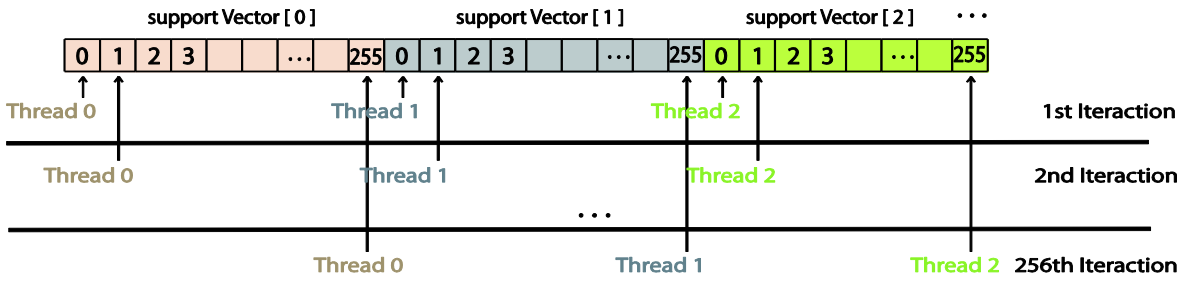

Figure 28 Memory access before efficiency maximization

Figure 28 shows the memory accesses pattern. Each thread, in the process of acquiring the respective support vector, accesses the memory in a way that it is not possible to retrieve all the data from the memory at once (each thread requests a value 256 positions away from the previous memory request). The immediate consequence of this is that multiple memory accesses are required (one access per thread request), in order to fetch all the needed support vectors. If the memory data requests were contiguous, the memory accesses would be combined to a single memory request (and the requested data would be delivered at a rate close to the peak of the global memory bandwidth).

To solve this issue, the support vectors were rearranged in memory, in a column major way (for every model files):

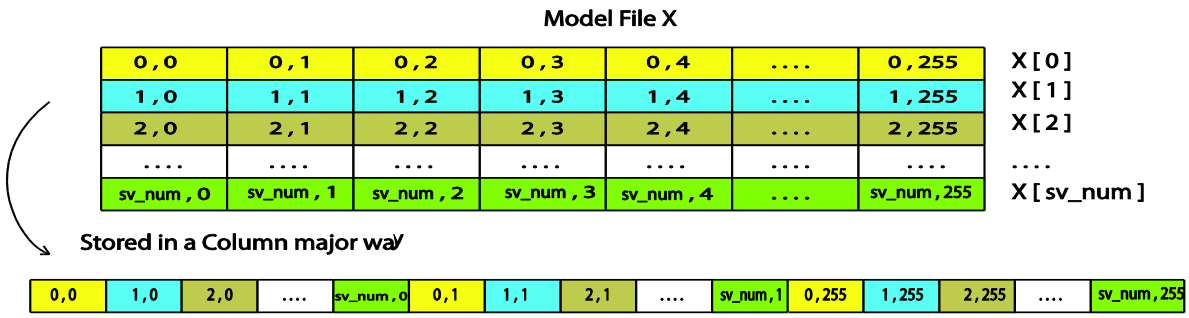

Figure 29 Support vectors in a column major disposal

With the support vectors displayed in a column major way, every time that a value is requested from the GPU memory, the result will be a single combined memory request, for all the consecutive locations:

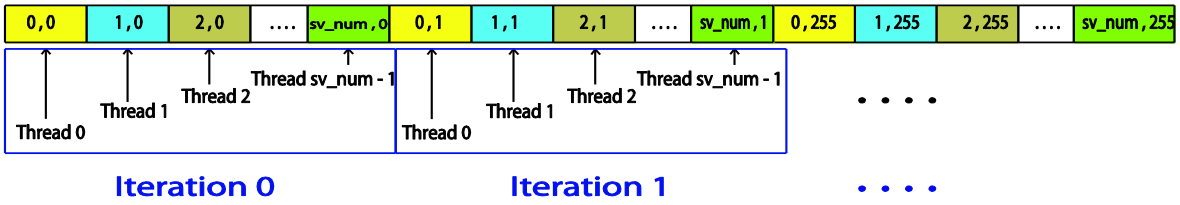

Figure 30 Coalesced memory access

Figure 30 illustrate the GPU memory access patterns. Every time a value is needed from GPU memory, the accesses will be coalesced. With this approach, model files will be partitioned in amounts of 256 support vectors. Naturally when the number of support vectors in a model file is not a multiple of 256, the last block launched will have threads with no work to do. The mean execution times of the classification activity only for each frame were calculated for both versions (sequential version as SV and independent thread approach as A3) as presented in Table 4-10 for both machines (times are presented in milliseconds):

	Machine 1	Machine 2
Per frame classification(ms)	Classifications	Classifications
SV	81,50842	46,37788
A3	55,23863	4,027326

Table 4-10 Sequential Classification vs. independent thread approach

To quantify the improvement (or degradation), the speedup for the classification fraction was calculated using the equation (4.5):

Machine 1:

$$SpeedupClassifm1 = \frac{SV}{A3} = \frac{81{,}50842}{55{,}23863} = 1{,}475569091$$

Machine 2:

$$SpeedupClassifm2 = \frac{SV}{A3} = \frac{46{,}37788}{4{,}027326} = 11{,}51579981$$

The global execution times of topographic segmentation algorithm were also collected as presented in Table 4-11:

Videos	Machine 1			Machine 2		
	Non-Parallelized	Parallelization approach	Overall Speedup	Non-Parallelized	Parallelization approach	Overall Speedup
1	00:30:42	00:21:53	1,431991877	00:17:30	00:02:29	6,89572819
2	01:08:00	00:48:23	1,431969651	00:38:33	00:05:30	6,952134404
3	01:25:00	01:00:22	1,432110338	00:47:51	00:06:52	6,960514743

Table 4-11 Overall Speedups of the independent thread approach

The overall speedup in the execution using this independent thread approach is almost 7 times in machine 2 and over 1.4 times in machine 1, compared to the sequential code of CapView. With fewer dependencies between threads, as well as with the data arranged differently in memory, this approach explored the GPU power and took advantage of its architectural aspects in this context.

4.4. Parallelization's profiling

At this point, and after analysis of several parallel solutions, it is important to understand the impact of the optimizations we introduced in the CapView algorithm. A new profiling was necessary and performed as described in both Figure 31 and Table 4-12.

Videos	Decode	YcbCr->RGB	Hist. Ops	Classifications	Others
1	0,0443%	0,0099%	42,6935%	57,1331%	0,1192%
2	0,0442%	0,0099%	42,6656%	57,0959%	0,1844%
3	0,0442%	0,0099%	42,6440%	57,0670%	0,2349%

Table 4-12 Profiling values after parallelization

From the presented results, it can be observed the relative impact of the classification after the parallelization efforts. Its execution time has decreased to 57% from the initial ~93%. A relative impact is also shown in the new profile, which regards to the histogram operations (a

raise from ~6 to ~42%). This suggests that the overall execution times can be further diminished in the parallelization of histogram operations.

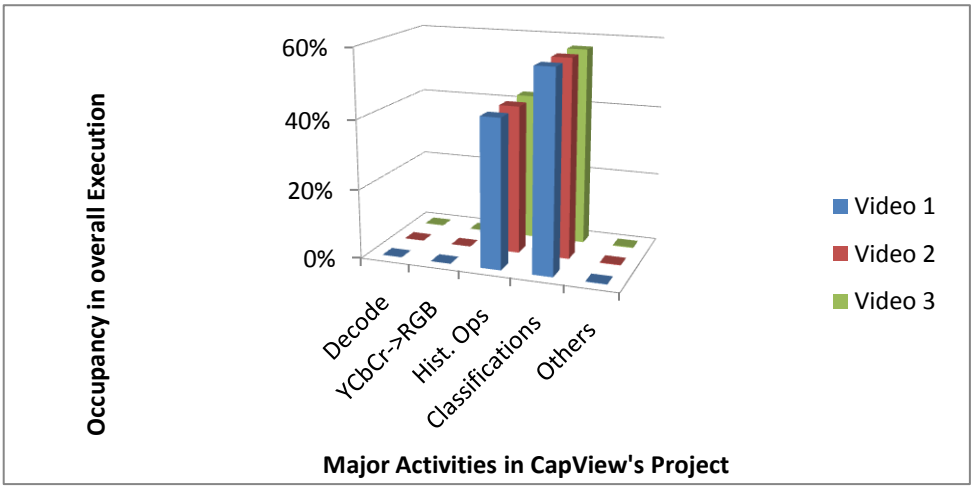

Figure 31 Profiling of CapView after GPU optimizations on the classification

As previously done, Amdahl's law was applied to find the maximum theoretical speedups on both machine 1 and 2 as the maximum theoretical speedups:

Machines	cores	Histogram Ops
1	8	1,595240236
2	240	1,738112837

Table 4-13 Theoretical values for machines 1 and 2

Although the maximum theoretical speedup values presented by Amdahl's law do not indicate a substantial speedup, the 42% of the overall execution time dedicated to histogram operations may imply sufficient gain in its parallelization. NVIDIA has a reference implementation code to perform simple histogram calculation[93], prepared for 64 bins or 256 bins. That code was adapted to be included in the CapView's normalization as well. The global execution times using the NVIDIA's adapted histogram code are presented in Table 4-14 for both machines, where the thread independent approach is presented as reference (times are in the format (hh:mm:ss)).

	Machine 1			Machine 2		
Videos	Independent Thread Approach	Histogram Parallelization	Overall Speedup	Independent Thread Approach	Histogram Parallelization	Overall Speedup
1	00:21:53	00:26:11	0,92286967	00:02:29	00:02:30	0,99956986
2	00:48:23	00:57:54	0,92215684	00:05:30	00:05:31	0,99891169
3	01:00:22	01:12:12	0,92215326	00:06:52	00:06:53	0,99892317

Table 4-14 Overall execution times comparison of Histogram Ops

Overall degradation on performance was observed as shown in overall speedups. The parallelization of the histogram operations does not benefit from the histogram optimization as initially expected. In both machines there is a degradation of the execution time (especially in machine 1). These prospective results led to exclusion of the GPU based histogram solution from our GPU optimized version of the CapView.

4.5. Multi GPU usage

So far, only one GPU has been used in all the approaches. Beyond the potential of a single CUDA-enabled GPU, it is possible to combine as many CUDA-enabled GPUs as desired, thus allowing further scaling of the existing parallel solution.

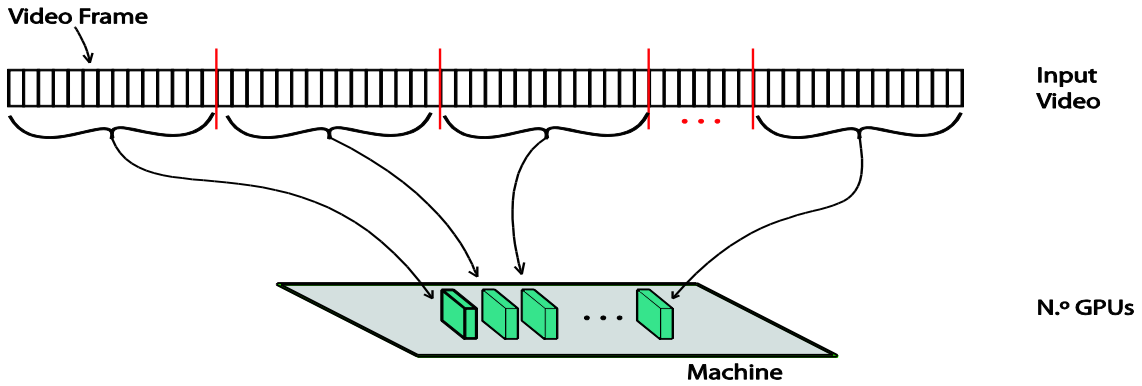

Figure 32 Video Segments and n.º GPUs

To do so, and in order to guarantee that every GPU contributes to the final solution of the CapView application, the video to be classified can be partitioned into smaller segments. As depicted in Figure 32, each one of those parts will be delegated to a single GPU, with the number of segments being equal to the number of GPUs to be used.

To handle multiple GPUs and multiple video segments, the GPU-based solution developed (under Linux OS) uses a multi-threading system where the several GPUs that are available are managed by individual threads. Although this is no longer explicitly needed in the newest toolkit from NVIDIA [40], for compatibility reasons with the computational capability of all CUDA devices (and toolkits as well [40]), this management was performed explicitly.

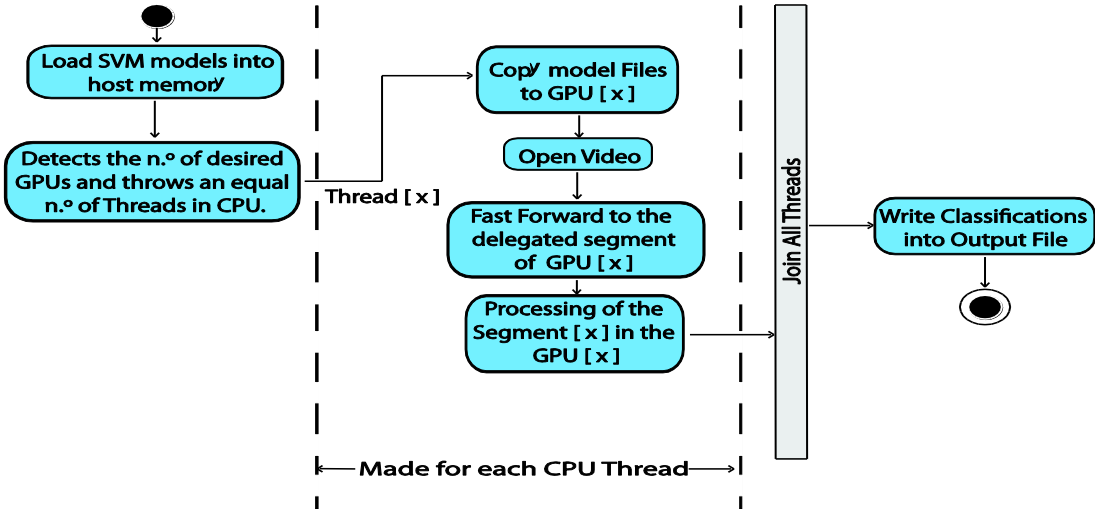

Figure 33 Execution flow of the Parallel Code in multi-GPU approach

In order to use multi-GPUs (combine many GPUs in a parallel solution), it is necessary to transform the existing parallel solution into a multi-threading solution, instead of a single threading solution [94] (here, the Independent Thread approach will be considered as the parallel solution, as it is the one that presents higher speedups). Thus, the CapView's parallel code is now translated into the activities diagram of Figure 33. The presented activities diagram is very similar to the flowchart of the sequential code (Figure 9); nevertheless, it has some differences, in order to support the multi-GPU feature. Those differences are mainly due to:

- The fact that a fast forward is required (performed by each thread), in order to be able to classify a single and distinct segment of the video;
- The redundancy of model files (every GPU must have all model files), thus allowing full independence between the host and the GPUs themselves;
- A mandatory join operation, in order to guarantee coherence in the overall execution and consistency in the last activity to be made (write all the classifications performed in an output file);

The "Processing of the segment [x] in the GPU[x]" is equivalent to the "Video Processing" activity in the sequential code; however, the classification is performed in a parallel way.

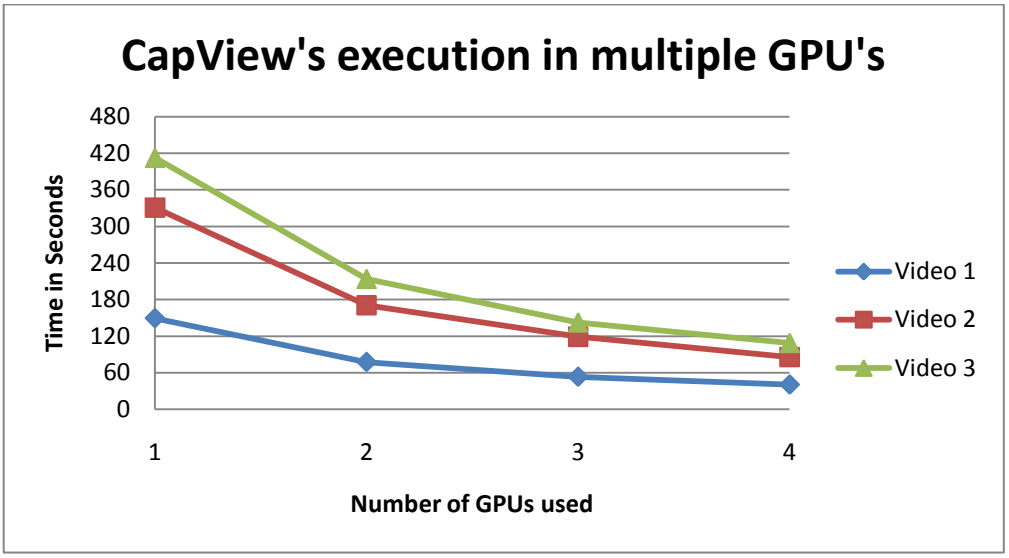

Figure 34 Parallel code's execution in multi-GPU mode

The execution times of the parallel solution are illustrated in the chart of Figure 34, as a relation between each one of the four GPUs that machine 2 offers and the time needed to classify the specific video. The maximum speedups (Table 4-15) seem above the maximum theoretical gain stated by Amdahl's law (Table 4-5). This is due to the fact that the multi-threading approach in the host parallelized other activities (decode, color-space conversions, histogram operations and classification), as opposed to the single classification's parallelization approach. The number of threads launched in the host was one per GPU (used in machine 2), which could be well supported by the i7 processor used in the same machine (quad-core, supporting easily 8 threads in simultaneous execution [95]). That factor contributed to a substantial increase performance of this approach. As it was observed in the chart above, raising the number of GPUs used caused a sub non-linear reduction in the execution times. A thorough evaluation of the fact was not conducted, although we hypothesize that the need of a fast forward before the classification of each video's segment (a sequential process with execution time proportional to the initial frame position of the video segment) contributed to that fact. Amdahl's law also foresees a non-linear evolution when raising of number of processors, as an higher number of processors leads to an higher overhead in the parallelism management [96].

Table 4-15 (and Figure 34) proves a drastic reduction in the execution times of the CapView's parallel code, by using this scheme. We executed the processing for each video in a

total of 5 runs and extracted the average execution times of the global execution times of the topographic execution times expressed in hh:mm:ss (hours, minutes and seconds):

Videos	Sequential Reference Optimized	1 GPU	SP	2 GPU	SP	3 GPU	SP	4 GPU	SP
1	00:17:30	00:02:29	7,05	00:01:17	13,64	00:00:53	19,81	00:00:40	26,25
2	00:38:33	00:05:30	7,01	00:02:50	13,61	00:01:59	19,44	00:01:25	27,21
3	00:47:51	00:06:52	6,97	00:03:33	13,48	00:02:22	20,22	00:01:48	26,58

Table 4-15 Execution times for multiple GPU usage

Maximum overall speedup (SP) was measured for all the GPU application, compared to the sequential reference. In this approach, even the longest video can be completely classified in less than two minutes, with overall speedups between 26.2 and 27.2 times with 4 GPUs.

4.6. Comparison with Other Parallel Approaches

CapView sequential topographic segmentation algorithm was also deployed and tested in both Grid and cluster computing [97]. In this section we present a comparison between our GPU-based solution and the existing Grid and cluster computing. For this analysis the videos used in [97] were tested in both our configurations for the sequential versions and our GPU-based solutions. Results can be seen in Table 4-16 for sequential implementation, in Table 4-17 for single GPU and in Table 4-18 for the multi-GPU (machine 2 only).

We executed the processing in both machines for each video in a total of 5 runs and extracted both the minimum, maximum and average execution times expressed in hh:mm:ss (hours, minutes and seconds):

NO GPU	Machine 1				Machine 2			
Videos	Min	Max	Avg	Std Dev	Min	Max	Avg	Std Dev
1	00:30:41	00:30:43	00:30:42	00:00:01	00:17:20	00:17:49	00:17:30	00:00:12
2	01:08:13	01:08:54	01:08:28	00:00:19	00:38:20	00:39:18	00:38:33	00:00:25
3	01:24:41	01:25:53	01:25:14	00:00:35	00:47:50	00:47:54	00:47:51	00:00:04

Table 4-16 Sequential execution times in both machines

WITH GPU	Machine 1				Machine 2			
Videos	Min	Max	Avg	Std Dev	Min	Max	Avg	Std Dev
1	00:25:00	00:25:00	00:25:00	00:00:00	00:02:28	00:02:29	00:02:29	00:00:00
2	00:53:40	00:54:18	00:53:55	00:00:15	00:05:30	00:05:31	00:05:31	00:00:00
3	01:06:58	01:09:53	01:07:38	00:01:15	00:06:49	00:06:53	00:06:53	00:00:02

Table 4-17 Execution Values with 1 GPU

N.º GPUs	Videos	Min	Max	Avg	Std Dev
1 GPU	1	00:02:28	00:02:29	00:02:29	00:00:00
	2	00:05:30	00:05:31	00:05:31	00:00:00
	3	00:06:49	00:06:53	00:06:53	00:00:02
2 GPUs	1	00:01:16	00:01:18	00:01:17	00:00:01
	2	00:02:50	00:02:51	00:02:51	00:00:01
	3	00:03:33	00:03:35	00:03:33	00:00:01
3 GPUs	1	00:00:52	00:00:58	00:00:53	00:00:03
	2	00:01:53	00:02:08	00:01:59	00:00:06
	3	00:02:22	00:02:23	00:02:22	00:00:01
4 GPUs	1	00:00:39	00:00:43	00:00:40	00:00:02
	2	00:01:25	00:01:25	00:01:25	00:00:00
	3	00:01:46	00:01:54	00:01:49	00:00:03

Table 4-18 Execution Values with various GPUs

To compare the execution times with Grid and cluster, we used the timings presented in [97] with the same datasets.

As described, the video analysis in such parallel environments was made accordingly to three data partitioning strategies: 4, 8 and 16 parts. The video was divided into smaller parts and every part is executed in different processors (e.g., in data partition with 4 parts, each part will be processed in one different node).

In the Grid, Oliveira, *et al.* measured the times between CapView sending the data to Grid or cluster until the moment that the results are received (network latencies included). It is important to note that in the cluster and Grid study no optimizations were done on the classification algorithm other than minor adaptations to handle video segments, instead of full video as in the original CapView implementation. If such optimizations were made, execution times could be improved as illustrated in our memory related optimization over the original sequential version.

N.º Parts	Videos	Min	Max	Avg	Std Dev
1/16	1	00:07:05	00:10:38	00:08:54	00:00:59
	2	00:08:34	00:14:36	00:11:24	00:01:52
	3	00:09:35	00:13:36	00:10:59	00:01:19
1/8	1	00:08:39	00:12:37	00:10:23	00:01:12
	2	00:11:36	00:13:38	00:12:33	00:00:35
	3	00:12:35	00:22:43	00:15:46	00:02:59
¼	1	00:11:39	00:19:11	00:16:03	00:03:35
	2	00:18:38	00:22:09	00:20:40	00:01:35
	3	00:20:13	00:24:39	00:22:02	00:02:03

Table 4-19 Execution Values in Grid infraestructure

N.º Parts	Videos	Min	Max	Avg	Std Dev
1/16	1	00:02:24	00:02:31	00:02:28	00:00:03
	2	00:04:33	00:04:45	00:04:38	00:00:06
	3	00:06:06	00:06:21	00:06:12	00:00:08
1/8	1	00:03:11	00:03:19	00:03:14	00:00:04
	2	00:06:20	00:06:29	00:06:25	00:00:04
	3	00:08:23	00:08:40	00:08:31	00:00:08
1/4	1	00:04:44	00:04:56	00:04:52	00:00:07
	2	00:09:55	00:10:13	00:10:04	00:00:09
	3	00:12:35	00:12:51	00:12:40	00:00:09

Table 4-20 Execution Values in Cluster computing

The GPU-based solution obtained comparable results compared to existent Grid and cluster solutions. Even with the usage of a modest model in single GPU (comparing to existent models [68]), the overall execution times easily surpassed the values produced in the Grid, being as well too close to the most promissory values produced in cluster (16 parts). The execution times gap between Grid and cluster solutions when compared with the multi-GPU approach is very substantial, being even less than one third in the longest video on cluster. It is worth to mention that in Grid the overall execution times were less homogeneous (high standard deviation) in contrast with both cluster and GPU solution.

We can also observe the same results in terms of throughput (average time spent per 100 MB) for the several solutions that are presented in Figure 35 (the best configuration for sequential, Grid and cluster were selected). Using this "normalized" measure, the comparison between different implementations becomes more intuitive; as expected, the GPU solutions can deliver better execution times per data chunk.

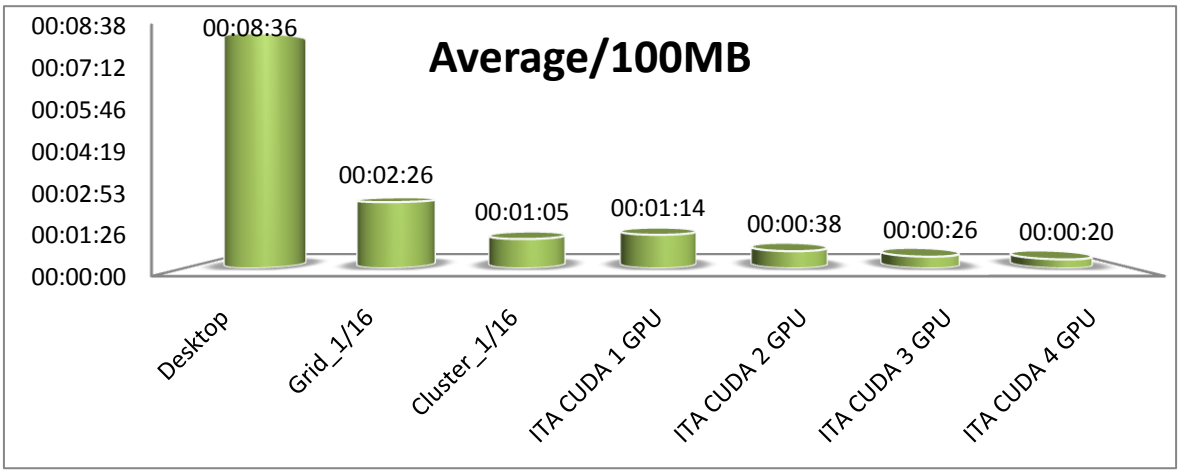

Figure 35 Average execution times per 100 MB

5. Integration of GPUs in CapView

Considering the improvements obtained using GPUs in the topographic segmentation based on CapView algorithm, it was logic to study the integration of this new solution in the current CapView application. The parallel solutions of CapView application were developed in Linux environment using a command line interface for the topographic segmentation algorithm interaction. The CapView version with a graphical user interface (GUI) was originally developed for Windows OS - the operating system in medical facilities [98]. The CapView GUI uses the same command line solution to interact with the parallel topographic segmentation algorithm.

5.1. The GUI interaction

To integrate the GUI with the topographic segmentation application an extra option was added to the menu that allows the execution of the topographic segmentation in the CUDA enabled GPU (Figure 36).

When that option is selected, the segmentation process starts to run in background of CapView's main application. While waiting for the end of segmentation the main CapView GUI shows an incremental bar with the progress of the segmentation.

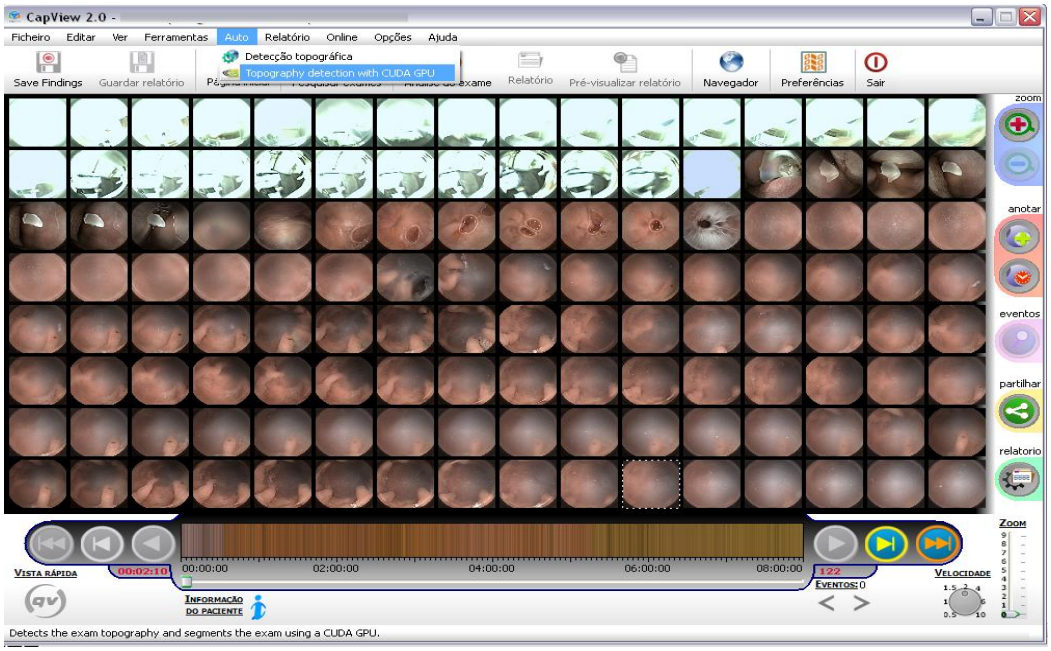

Figure 36 Integration of parallel implementation in CapView's application

After the topographic segmentation is performed, the GUI receives the topographical markers/barriers, and presents them as events in the correspondent place of the bar that is located under the set of images. Likewise, the topographic barriers that mark the transition from one zone to another are equally put on top of the correspondent image as a red circle, to provide a more intuitive navigation throughout the video analysis.

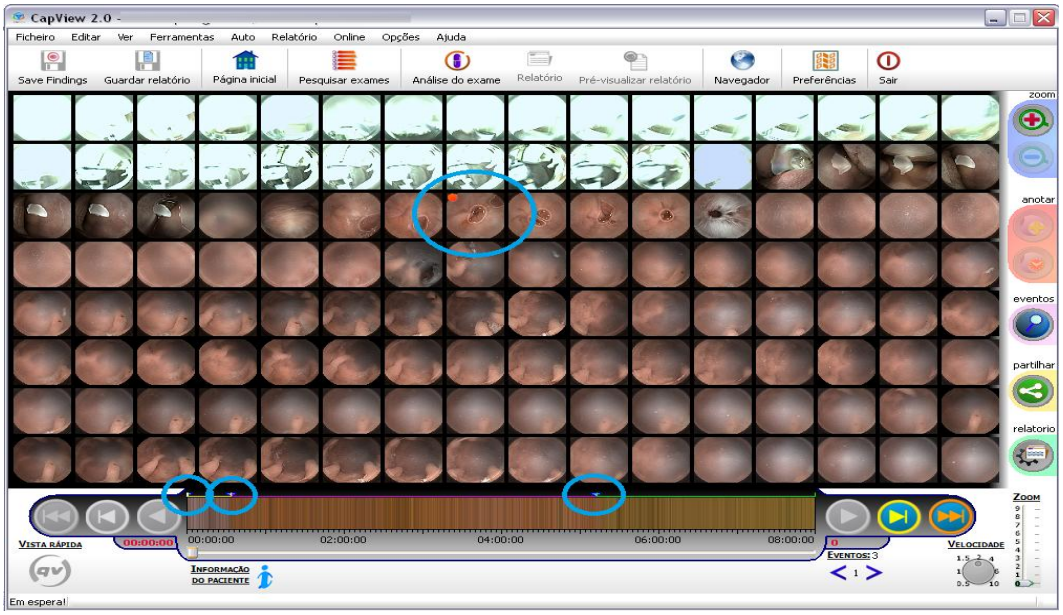

Figure 37 Topographic events marked in the main application.

5.2. Evaluation

We executed the GPU solution integrated in CapView in a third machine, an up-to-date desktop PC running windows OS and with a CUDA enabled GPU similar to those available in the current clinical environments (specifications are in Table 8-1, appendix 0).

A total of 5 runs was performed for each video and the minimum, maximum and average execution times where extracted (presented in Table 5-1). All the execution times were measured between the beginning of CapView's execution and the returning of topographic markers. Values are expressed in hh:mm:ss (hours, minutes and seconds):

Videos	Sequential Execution	Min	Max	Average	Std Dev	Overall speedups
1	00:18:41	00:01:53	00:01:55	00:01:53	00:00:01	9,92
2	00:49:12	00:03:53	00:03:54	00:03:54	00:00:01	12,62
3	00:51:34	00:04:51	00:04:52	00:04:51	00:00:01	10,63

Table 5-1 Integration Evaluation in machine 3

Table 5-1 shows that with the machine 3 we obtained smaller execution times when compared with the other configuration (Table 4-11).

5.3. Integration issues

The integration in Windows environment caused some adaptation considerations, given the appearance of some issues. Such substantial incompatibility (multithreading system and libraries for video handling), capable of compromising the whole adaptation, arose from the explicit differences of both operating systems. The relevant adaptations made are concerning to the multithreading system and the FFmpeg libraries, and despite the fact that only one of the two is sufficiently strong to prohibit the adaptation, the other highly limits future computational scalabilities.

5.3.1. Multithreaded GPU

The first one to be pointed out is the limitation that states its execution in only one GPU. There are large differences between Windows and Linux thread implementations and usages. Even in Windows only, from preemptive approaches to different thread implementations from one Windows version to another [99], there are many aspects regarding multithreading in

Windows. Relevant to this dissertation is the fact that a windows application must use a specific application programming interface (API) [100] to interact and manage threads. The original multithreaded Linux solution uses the POSIX thread standard [101], a completely different standard API from the Win32 threads API.

Since the thread implementation differences are substantial (not only because of the API issue pointed out, but also for other equally important issues like multithreading models [102] or threading system implementations [103]), a single GPU solution in Windows OS was chosen (avoiding in this way threads handling). It was also opted this way to be able to provide a validated and operational solution in time that could be used in clinical environments within the temporal scope of this dissertation. A fully working multi-GPU approach in Windows is achievable but seemed not possible within this time constraints, namely to the extra learning curve needed to address the specificities of the thread programming models in comparison to the Linux based solution.

5.3.2.FFmpeg: the video

The most crucial problem when porting the original solution from Linux to Windows was the dependency on the FFmpeg libraries needed to access and process the videos. Originally, FFmpeg project was developed in Linux, using, in most of the code, the C99 standard [104]. This standard is not used in Windows OS, which implies using external tools capable of creating a favorable environment to compile those libraries in Windows OS namely MSYS or MinGW [105]. However, the task of compiling and building windows compatible FFmpeg libraries is not straight forward implying patches and tweaks. In this context we opted to rely on existing pre-compile FFmpeg. The version selected is a 32 bits architecture version and supporting the basic functionalities needed for running the CapView Segmentation. The main drawback of the current solution is that with changes in Windows OS, it is possible that those builds will no longer function properly.

5.4. Integration Considerations

A detailed comparison between the tested GPU solutions PC / host versus PC only in this dissertation would imply not only considering the algorithm implementation but also machine configurations namely CPU and GPU specification. Such comparison was considered to be out

of scope of these dissertation objectives, as the number of variables involved would be very high and with complex interactions (e.g. CPU, GPU and the GPU use strategy in the algorithm implementation). However, supported on our results, it is reasonable to conclude that it is clear that memory is not a major factor in the CapView executions times, as despite the fact that machine 3 GPU had four times less memory in comparison with machine 2 GPU, it had a superior performance (single GPU, Table 5-1 and Table 4-11).

6. Conclusions and Future Work

6.1. Assessment of purposed objectives

One of the objectives of the present dissertation was to assess the feasibility of applying GPUs in a concrete medical imaging problem: the topographical segmentation of endoscopic capsule video. After studying several approaches, it was clear that the GPU can be applied to this field, contributing to attain relevant gains in the processing times. This was observed in all the hardware configurations we used. In the two reference machines used, with a single GPU, we could observe the following speedups:

- ~1.38 and ~6.4 for machine 1 and 2 respectively (using the CUBLAS library, a GPU enabled implementation of the BLAS library [91]). This can be translated in a reduction (worst case scenario) from 1 hour and 25 minutes to 1 hour and 2 minutes (machine 1), and from 47 minutes and 51 seconds to 7 minutes and 24 seconds (machine 2).
- ~1.43 and ~6.9 for machine 1 and 2 respectively (using a personalized kernel, to take advantage of the CUDA architecture specificities). This can be translated (worst case scenario) in a reduction from 1 hour and 22 minutes to 1 hour and 22 seconds (machine 1), and from 47 minutes and 51 seconds to 6 minutes and 52 seconds (machine 2).

In a multi-GPU approach (4 GPU), the observed speedups were between 26.2 and 27.2, when compared to the optimized sequential version. In such approach, the speedups caused a reduction of almost 50 minutes to less than 2 minutes in the execution time of the worst case scenario.

All the produced results for the topographic segmentation were validated, being exactly the same as the ones produced in the sequential algorithm.

With a careful analysis, the number of cores and frequency presented in a GPU card can be considered the relevant factor for performance variation in CapView's algorithm, along with the bandwidth between computer's RAM and GPU's global memory. Likewise, a small number of cores (as present in machine 1) caused a reduction of almost 25 minutes in the classification of the longest video. The performance gap between both machines is very substantial, reflecting and sound relation between performance gains and GPU cards: a bigger number of cores and frequency used will generate a better overall speedup in the application, given that the heavy processing is (re)designed to take advantage of the GPU.

The optimizations in CapView's segmentation algorithm respected the logical modular structure of the original algorithm. Although this option did not allow more in depth integration/optimization, it enabled easier and orthogonal evolutions of each individual logical module. With a more depth integration, through merging of logical modules in one implementation block, it would be possible, for instance, to avoid extra memory operations when sharing context between functional parts – not possible when enforcing the original modular design.

Our results, when compared to previous work using other advanced computing solutions, namely Grid and cluster infrastructures, show that the performance of Grid and cluster can be surpassed by a GPU based solution in terms of overall processing times. As the prices of GPUs are extremely competitive compared to Grid [106] and cluster [107] solutions, and as they also offer a tremendous transparency namely in the usage of different configurations (e.g. number of GPUs or number of cores), GPUs present an affordable option to increase the processing resources for personal and professional applications in desktop-based applications. Cluster and Grid, imposing high operational costs, have shown to be suitable for science (supporting e-science infrastructures). One should note that the use of GPU as explored in this work requires the redesign of algorithms.

The current GPU solution was integrated in the CapView application already in use in clinic environment. This integration was not trivial and was constrained by several platform specific details (e.g. FFmpeg implementations compatibility with windows OS, or different multi-threading implementation on Windows and Linux OS) and time constraints. Despite all problems, the application is running and, as shown, provides a significant improvement when compared to the original sequential implementation in terms of execution time.

6.2. Future work

It may be interesting to add remote on-demand analysis in the CapView's application, submitting the work in a shared, high-end CUDA-enabled GPUs resource. This feature would allow using CUDA-enabled GPUs in a computer without such hardware (with the cost of data transfers).

Another possible addition to CapView is the implementation of the GPU-based solution in a language that is supported by many GPU vendors (beyond NVIDIA), with OpenCL [39] as a possible candidate for that purpose. This would allow a more extensive application of CapView's parallel algorithm (by allowing its execution by any GPU under the OpenCL standard).

Since the endoscopic techniques used in this kind of exams are in a constant evolution (e.g.: new capsules with new features), it would be helpful to rely on an international standard capable of ruling those evolutions. This is relevant because a single modification in the video acquisition process can easily lead to incorrectness of the SVM model files already created, either in topographic segmentation or in the hypothetical model files for event detection.

7. References

[1] M. W. Vannier and R. A. Robb, "Medical imaging research and development groups," *Medical Imaging, IEEE Transactions on* vol. 20, p. 853 2001.

[2] I. N. Bankman, *Handbook of Medical Imaging: Processing and Analysis*. San Diego: Academic Press, 2000.

[3] M. Delvaux and G. Gay, "Capsule endoscopy in 2005: Facts and perspectives," *Best Practice & Research Clinical Gastroenterology,* vol. 20, pp. 23–39, 2006.

[4] M. Liévin, *et al.*, "Interactive 3D Segmentation and Inspection of Volumetric Medical Datasets," *Biomedizinische Technik/Biomedical Engineering,* vol. 47, pp. 75–78, 2009.

[5] D. L. Pham, *et al.*, "Current Methods In Medical Image Segmentation," *Annual Review of Biomedical Engineering,* vol. 2, pp. 315-337, 2000.

[6] K. Mönkemüller, *et al.*, *Interventional and Therapeutic Gastrointestinal Endoscopy* vol. 27: Karger, 2010.

[7] P. L. Dwyer, *Atlas of Urogynecological Endoscopy*. United Kingdom: Informa UK Ltd, 2007.

[8] IEETA. *CapView*. Accessed 14 May 2011. Available: http://www.capview.org/

[9] J. P. S. Cunha, *et al.*, "Automated Topographic Segmentation and Transit Time Estimation in Endoscopic Capsule Exams," *IEEE Transactions On Medical Imaging,* vol. 27, p. 1, 2008.

[10] M. Mackiewicz, *et al.*, "Wireless Capsule Endoscopy Color Video Segmentation," *IEEE TRANSACTIONS ON MEDICAL IMAGING,* vol. 27, 2008.

[11] J. Y. Chen, "GPU Technology Trends and Future Requirements," *Electron Devices Meeting (IEDM), 2009 IEEE International* p. 1, 2010

[12] Ricardo Marroquim and A. Maximo, "Introduction to GPU Programming with GLSL," *Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing,* 2009

[13] D. B. Kirk and W. m. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 1 ed. USA: Morgan Kaufmann, 2010.

[14] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," *Oxford University Press,* vol. 2, pp. 1–36, 1998.

[15] T.-L. Ji, *et al.*, "Adaptive Image Contrast Enhancement Based on Human Visual Properties," *Medical Imaging, IEEE Transactions on* vol. 13, pp. 573 - 586 1994.
[16] Z. Soferman, *et al.*, "Advanced Graphics Behind Medical Virtual Reality: Evolution of Algorithms, Hardware, and Software Interfaces," 1997.
[17] C. Pope and S. Ziebland, "Analysing qualitative data," *BMJ*, vol. 114, 2000.
[18] M. Marsousi, *et al.*, "Segmenting Echocardiography Images using B-Spline Snake and Active Ellipse Model," *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, p. 3125 2010.
[19] B. Menze, *et al.*, *Medical Computer Vision*. USA: Springer, 2010.
[20] M. I. Kuz'min-Krutetskii, *et al.*, "Video Systems for Endoscopy," *Biomedical Engineering*, vol. 37, pp. 212-216, 2003.
[21] G. Triadafilopoulos, "Endoscopy by nonphysicians," *American Society for Gastrointestinal Endoscopy*, vol. 69, 2009.
[22] S. Gross and M. Kollenbrandt, "Technical Evolution of Medical Endoscopy," *Acta Polytechnica*, vol. 49, pp. 1-5, 2009.
[23] P. B. Cotton, *et al.*, *Practical Gastrointestinal Endoscopy: The Fundamentals*, 2008.
[24] Peter Cotton and J. Leung, *Advanced Digestive Endoscopy: ERCP*. Massachusetts: Blackwell Publishing, 2005.
[25] D. J. Green, "Complications of Gastrointestinal Endoscopy," *BSG Guidelines in Gastroenterology*, 2006.
[26] N. J. Greenberger, *et al.*, *Current Diagnosis & Treatment: Gastroenterology, Hepatology, & Endoscopy*. New York: Mc Graw Hill Lange, 2009.
[27] G. Imaging. *PillCam SB*. Accessed 17 July 2011. Available: http://www.givenimaging.com/en-us/healthcareprofessionals/Products/Pages/PillCamSB.aspx
[28] M. Delvaux and G. Gay, "Capsule endoscopy: Technique and indications," *Best Practice & Research Clinical Gastroenterology*, vol. 22, pp. 813–837, 2008.
[29] A. Karargyris and N. Bourbakis, "Wireless Capsule Endoscopy and Endoscopic Imaging: A Survey on Various Methodologies Presented," *Engineering in Medicine and Biology Magazine, IEEE* vol. 29, 2010.
[30] F. Vilariño, *et al.*, "Intestinal Motility Assessment With Video Capsule Endoscopy: Automatic Annotation of Phasic Intestinal Contractions," *Medical Imaging, IEEE Transactions on* vol. 29, 2010.
[31] K. Rogers, *The digestive System* vol. 1. New York: Britannica, in association with Rosen, 2011.
[32] H. Junction. *Inflammatory Bowel – A disease getting common among youngsters*. Accessed 22 June 2011. Available: http://healthjunction.org/?tag=system
[33] G. Imaging. *RAPID Software*. Accessed 17 July 2011. Available: http://www.givenimaging.com/en-us/healthcareprofessionals/Products/Pages/Software.aspx
[34] J. P. S. Cunha and M. C. a. J. Soares, "capview: computer-aided diagnosis for endoscopic capsule," *Revista do DETUA*, vol. 4, pp. 1-4, 2007.
[35] E. Kilgariff and R. Fernando, "The GeForce 6 Series GPU Architecture," *Proceeding SIGGRAPH '05 ACM SIGGRAPH 2005 Courses* 2005.
[36] M. Harris. *GPGPU: General General-Purpose Computation on GPUs*. Accessed 22 June 2011. Available: http://http.download.nvidia.com/developer/presentations/2005/GDC/OpenGL_Day/OpenGL_GPGPU.pdf

[37] D. M. Chitty, "A Data Parallel Approach to Genetic Programming Using Programmable Graphics Hardware," *Proceeding GECCO '07 Proceedings of the 9th annual conference on Genetic and evolutionary computation* 2007.

[38] R. Buyya, *High Performance Cluster Computing: Programming and Applications Vol 2*. NJ, USA: Prentice Hall PTR, 1999.

[39] Khronos. *OpenCL Overview: The open standard for parallel programming of heterogeneous systems*. Accessed 02 May 2011. Available: http://www.khronos.org/opencl/

[40] NVIDIA. *Developer Zone*. Accessed 17 July 2011. Available: http://developer.nvidia.com/

[41] Khronos. *The Industry's Foundation for High Performance Graphics*. Accessed 24 June 2011. Available: http://www.opengl.org/

[42] A. M. Devices. *ATI Stream Technology* Accessed 02 May 2011. Available: http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx

[43] NVIDIA. *NVIDIA CUDA Architecture: Introduction and Overview*. Accessed 06 June 2011. Available: http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf

[44] Shujia Zhou, *et al.*, "Accelerating Climate and Weather Simulations Through Hybrid Computing," *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* 2010.

[45] N. Hinitt and T. Kocak, "GPU-based FFT computation for multi-gigabit wirelessHD baseband processing," *EURASIP Journal onWireless Communications and Networking,* 2010.

[46] Tobias Preis, *et al.*, "GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model," *Journal of Computational Physics,* vol. 228, pp. 4468–4477, 2009.

[47] N. Neophytou, *et al.*, "Hardware acceleration vs. algorithmic acceleration: Can GPU-based processing beat complexity optimization for CT?," *SPIE Medical Imaging,* 2007.

[48] S. N. University. *GPU based Medical Imaging*. Accessed 05 May 2011. Available: http://www.nvidia.co.kr/content/cudazone/download/showcase/kr/Seoul-univ-GPU-based-Medical-Imaging.pdf

[49] Shuqian Luo and J. Han, "Filtering Medical Image Using Adaptive Filter," *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE* vol. 3, pp. 2727 - 2729, 2002.

[50] J.-P. Guédon and Y. Bizais, "Bandlimited and Haar filtered back-projection reconstructions," *Medical Imaging, IEEE Transactions on* vol. 13, p. 1, 1994.

[51] Philippe Thévenaz, *et al.*, "Interpolation Revisited," *IEEE TRANSACTIONS ON MEDICAL IMAGING,* vol. 19, p. 1, 2002.

[52] A. Kumar, *et al.*, "Automatic Image Alignment and Stitching of Medical Images with Seam Blending," *World Academy of Science,* p. 1, 2010.

[53] R. L. Cook, "Shade Trees," *Computer Graphics,* vol. 18, pp. 1-9, 1984.

[54] Timothy J. Cullip and U. Neumann, "Accelerating Volume Reconstruction with 3D Texture Hardware," *World Academy of Science,* pp. 1-6, 1993.

[55] P. Taylor. *Programmable Shaders for DirectX 8.0*. Accessed 05 May 2011. Available: http://msdn.microsoft.com/en-us/library/ms810459.aspx

[56] B. Cabral, *et al.*, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Proceeding VVS 94 Proceedings of the 1994*

symposium on Volume visualization, pp. 1-9, 1994.

[57] P. B. Noël, *et al.*, "Clinical Evaluation of GPU-Based Cone Beam Computed Tomography," *Computer Methods and Programs in Biomedicine (2010)*, pp. 1-12, 2011.

[58] X.-L. Wu, *et al.*, "Advanced MRI Reconstruction Toolbox with Accelerating on GPU," *Proceedings of the IS&T/SPIE Electronic Imaging 2011 Conference on "Parallel Processing for Imaging Applications"*, pp. 1-10, 2011.

[59] B. Bai and A. M. Smith, "Fast 3D Iterative Reconstruction of PET Images Using PC Graphics Hardware," *Nuclear Science Symposium Conference Record, 2006. IEEE*, pp. 2787 - 2790 2007.

[60] C. Han-Yong, *et al.*, "Application of Segmentation and Measurement in the Treatment of Developmental Dysplasia of the Hip," *Bioinformatics and Biomedical Engineering, 2007. ICBBE 2007. The 1st International Conference on* pp. 989 - 991 2007.

[61] W. Wein, *et al.*, "Automatic CT-ultrasound registration for diagnostic imaging and image-guided intervention," *Medical Image Analysis*, 2008.

[62] Suren Chilingaryan, *et al.*, "A GPU-based Architecture for Real-Time Data Assessment at Synchrotron Experiments," *Real Time Conference (RT), 2010 17th IEEE-NPSS* 2010.

[63] E. Vuçini, *et al.*, "Enhancing Visualization with Real-Time Frequency-based Transfer Functions," *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, 2001.

[64] Santhanam, *et al.*, "Real-Time Simulation and Visualization of Subject-Specific 3D Lung Dynamics," *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on* pp. 629 - 634 2006.

[65] Takehiro Tawara and K. Ono, "A framework for volume segmentation and visualization using Augmented Reality," *3D User Interfaces (3DUI), 2010 IEEE Symposium on* 2010.

[66] F. Yuan, "An interactive virtual endoscopy system based on consumer level GPUs," *Digital Media and its Application in Museum & Heritages, Second Workshop on* 2007.

[67] A. Krüger, *et al.*, "Sinus Endoscopy - Application of Advanced GPU Volume Rendering for Virtual Endoscopy," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, 2008.

[68] NVIDIA. *CUDA GPUs*. Accessed 03 May 2011. Available: http://www.nvidia.com/object/cuda_gpus.html

[69] S. Tomov. *Discussion on NVIDIA's Compute Unified Device Architecture (CUDA)*. Accessed 23 June 2011. Available: http://web.eecs.utk.edu/~dongarra/WEB-PAGES/SPRING-2011/Lect13_CUDA_Discussion.pdf

[70] NVIDIA. *The CUDA Compiler Driver NVCC*. Accessed 06 June 2011. Available: http://sbel.wisc.edu/Courses/ME964/2008/Documents/nvccCompilerInfo.pdf

[71] E. Balagurusamy, *Programming in ANSI C 4E*: McGraw-Hill, 2007.

[72] D. Kirk and W. Hwu. *ECE 498 AL : Applied Parallel Programming*. Accessed 17 July 2011. Available: http://courses.engr.illinois.edu/ece498/al/

[73] S. Ryoo, *et al.*, "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA," *13th ACM SIGPLAN Symposium on Principles and practice of parallel programming* 2008.

[74] F. Bellard. *FFmpeg* [GNU Lesser General Public License (LGPL) version 2.1]. Accessed 30 March 2011. Available: http://www.ffmpeg.org/

[75] J. M. Martínez. *MPEG-7 Overview*. Accessed 30 March 2011. Available:

http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm#E12E23

[76] N. Sematech. *Engineering Statistics Handbook (6/01/2003 ed.)*. Accessed 03 May 2011. Available: http://itl.nist.gov/div898/handbook/eda/section3/histogra.htm

[77] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Second Edition ed. USA: Springer, 1995.

[78] K.-R. Müller, *et al.*, "An Introduction to Kernel-Based Learning Algorithms," *Neural Networks, IEEE Transactions on* vol. 12, p. 21, 2001.

[79] M. Schmidt and H. Gish, "Speaker identification via support vector classifiers," *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on* p. 4, 2002.

[80] Y. Li, *et al.*, "Support Vector Regression and Classification Based Multi-view Face Detection and Recognition," *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on* p. 6, 2000.

[81] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," vol. 1398/1998, pp. 137-142, 1998.

[82] ClopiNet. *SVM Application List*. Accessed 07 April 2011. Available: http://clopinet.com/isabelle/Projects/SVM/applist.html

[83] DTREG. *Introduction to Support Vector Machine (SVM) Models*. Available: http://www.dtreg.com/svm.htm

[84] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. United Kingdom: Cambridge University Press, 2000.

[85] B. Schölkopf, *et al.*, "Input Space Versus Feature Space in Kernel-Based Methods," *IEEE Transactions on Neural Networks,* vol. 10, pp. 1000 - 1017 1999.

[86] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery,* vol. 2, pp. 121–167, 1998.

[87] I. Foster, *et al.*, "The Anatomy of the Grid," *International Journal of High Performance Computing Applications,* vol. 15, 2001.

[88] M. Baker, *et al.*, "Cluster Computing and Applications," 2000.

[89] M. Harris. *Optimizing Parallel Reduction in CUDA* [NVIDIA Developer Technology]. Accessed 24 June 2011. Available: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf

[90] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *IEEE Computer Society,* 2008.

[91] NVIDIA. *CUDA CUBLAS Library*. Accessed 16 July 2011. Available: http://developer.download.nvidia.com/compute/cuda/2_0/docs/CUBLAS_Library_2.0.pdf

[92] netlib_maintainers. *BLAS (Basic Linear Algebra Subprograms)*. 2005, Accessed 21 March 2011. Available: http://www.netlib.org/blas/

[93] V. Podlozhnyuk, "Histogram calculation in CUDA," November 2007.

[94] Kashyap. *CUDA multi-gpu textures* [Blogue]. Accessed 22 April 2011. Available: http://gpuray.blogspot.com/2010/01/cuda-multi-gpu-textures.html

[95] Intel. *Intel Core i7-950 Processor*. Accessed 24 June 2011. Available: http://ark.intel.com/Product.aspx?id=37150

[96] J. Wang, *et al.*, "Architectural Support for Reducing Parallel Processing Overhead in an Embedded Multiprocessor," *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 2010.

[97] I. C. Oliveira, *et al.*, "Extending a desktop endoscopic capsule video analysis tool used by doctors with advanced computing resources," *presented at the IberGrid, Santander, Spain*, 2011.

[98] J. Brodkin. *100 iPads Set for Deployment at Californian Hospital*. Accessed 10 May 2011. Available: http://www.pcworld.com/article/194655/100_ipads_set_for_deployment_at_californian_hospital.html

[99] C. L. Incorporated. *Multithreading in Windows NT*. Accessed 07 June 2011. Available: http://www.calsoftlabs.com/whitepapers/multithreading.html

[100] Intel. *Programming with Windows Threads*. Accessed 07 June 2011. Available: http://zeus.lci.ulsa.mx/divulgacion/Material/pdf/Windows%20Threads.pdf

[101] B. Barney. *POSIX Threads Programming*. Accessed 07 June 2011. Available: https://computing.llnl.gov/tutorials/pthreads/

[102] Silberschatz. *Chapter 4: Multithreaded Programming*. Accessed 08 June 2011. Available: http://www.inf.uni-konstanz.de/dbis/teaching/ss06/os/ch4.pdf

[103] M. E. Russinovich, *et al.*, *Windows Internals 5th Edition*. Washington: Microsoft Press, 2009.

[104] FFmpeg. *Main Page FFmpeg on Windows* Accessed 19 May 2011. Available: http://ffmpeg.arrozcru.org/wiki/index.php?title=Main_Page

[105] FFmpeg. *FFmpeg on Windows*. Accessed 19 May 2011. Available: http://www.ffmpegwindows.org/

[106] e-Eighteen.com. *Power Grid Corporation of India*. Accessed 24 May 2011. Available: http://www.moneycontrol.com/stocks/company_info/pricechart.php?sc_did=PGC

[107] i. Technologies. *Web Cluster*. Accessed 24 May 2011. Available: http://iweb.com/managed-hosting/web-cluster

8. Appendix

The topographic segmentation algorithm execution times were measured in two different computers. Machine 1 is a common laptop used for development. Machine 2 is a remote, high-end computer available in the lab, dedicated to GPU programming exercises. Both computers run Linux. For the integration with CapView application, a third computer was need, a desktop running Windows. The specifications of all three machines are summarized in Table 8-1.

	Machine 1	Machine 2	Machine 3
Architecture	32 bit	64 bit	32 bit
CPU (Intel)	Core 2 Duo T6400	i7 CPU 950	Dual-Core E5200
RAM	4 GB	12 GB	4 GB
Frequency	2.00 GHz	3.07 GHz	2.50 GHz
GPUs			
N.º GPU's	1	4	1
Model	GeForce 9300M GS	Tesla T10 Processor	GeForce GTX 275
Driver ver.	3.20	3.20	4.0
Capability	1.1	1.3	1.3
Cores	8	240	240
Memory	251.06 MB	4.29 GB	848 MB
Clock Rate	1.45 GHz	1.44 GHz	1.46 GHz

Table 8-1 Specifications