FaceID@home: cycle-sharing for facial recognition

FaceID-BOINC: adaptação de algoritmos de reconhecimento facial (eigenfaces) para execução em máquinas multicore e GPUs integrado num cliente para plataforma BOINC

Nuno Miguel Abreu Teixeira - 55397

Instituto Superior Técnico

Abstract. Over the last years we experienced a revolution in the creation and dissemination of media content such as videos and photos. However, this information is not cataloged and there is no automatic system that indexes movies according to its content. To solve this problem we propose FaceID@home. This system will automatically detect and recognize faces using the Eigenfaces algorithm. To tackle the high demand of computing power from video and image processing, we will execute this system in large scale through a public distributed computing infrastructure based on BOINC. We will also make use of the volunteers' graphics processors to improve the speed of facial recognition algorithms. With this work we expect to create a free, efficient and scalable system that automatically identifies and indexes people in multimedia content.

Keywords: BOINC, Eigenfaces, GPGPU, Peer-to-Peer

1 Introduction

The evolution of technology over the last decades changed the way we access and create multimedia content. The widespread access to cameras, cellphones and computers provides means that allow anyone to create images and videos in an inexpensive way. The growing speed and availability of Internet connections is the perfect vehicle to share this information across the globe. As a result, we now live surrounded by a growing amount of multimedia data provided not only by the traditional sources, but also from anyone that publishes videos and photos in the Internet. We can confirm this trend by looking at statistics from popular websites. For example Youtube receives 24 hours of video every minute and 2 billion videos are viewed everyday[1].

The enormous of information highlights the need for multimedia indexing technologies. Manual categorization and tagging of all this data is an unfeasible task. Some details are usually provided by the publisher, such as a title and a short description, but many other informations could be obtained by using an automated system that analyses the content of the video.

While browsing through multimedia content we often want to find people. The problem is, unless this information is manually specified, we have no way of knowing in which movies and photos a certain person appears. Thus, we believe that the automatic identification of faces is an important step towards the creation of an automated multimedia indexing system.

Facial recognition has been a major research focus during the last decades. Motivated by its application in different areas such as security, law enforcement or entertainment, many approaches have been proposed. Some of these methods have shown fairly good results, however, performing a robust face detection is still difficult. The diversity of situations in which a face appears is one of the greatest difficulties for the recognition algorithms [61]. Some companies developed software and services that perform facial identification and recognition, such as Google Picasa[2] and Apple's iLife[3]. These solutions work for a private collection of photos and videos, but they don't answer the need for a global system that automatically identifies people over publicly available multimedia content.

The identification of faces in a video undergoes several stages. The frames must be extracted from the video, then analyzed to detect faces and finally these faces must be recognized. This is a potentially demanding task for a single computer. There are alternatives to speed up the process, such as supercomputers or clusters. The problem is that these resources are usually owned by specific organizations and are extremely expensive to use.

[1] http://www.website-monitoring.com/blog/2010/05/17/youtube-facts-and-figures-history-statistics/
[2] http://www.picasa.google.com
[3] http://www.apple.com/ilife/

With this work we propose a system called FaceID@home that automatically detects and indexes human faces in multimedia content such as movies and photos. However, our research efforts won't be directed towards a new facial recognition method. Instead we will create a robust computational system that can efficiently process great amounts of data to identify faces with low maintenance costs. In order to do so, we will explore the use of a public computing infrastructure based on BOINC [3] to distribute tasks across several computers.

Public computing uses personal computers from volunteers to process complex computational problems. The main attraction in such a system is that it is possible to achieve a high computational power with low maintenance costs. There are many active projects that rely on this architecture. For instance SETI@home [4] is supported by hundreds of thousands of people around the world that donate processing time from their computers.

Our system will use a similar infrastructure to tackle the high processing demands of face recognition algorithms. We will define different tasks that must be performed to identify faces in a video, such as frame extraction, face detection and face recognition. Each of these tasks will be scheduled to volunteers according to their hardware capabilities in order to maximize the efficiency of the system.

Another proposed improvement, is the use of existing graphics processors to accelerate the computation. Graphics processors were initially designed to synthesize images and help in Computer-Aided Design (CAD), but they evolved to become a powerful and flexible processor that can outperform more expensive Central Processing Units, commonly known as CPUs. It is important to bear in mind that a graphics processor has a very different architecture from common processors, which imposes serious restrictions on the tasks that it can perform efficiently. To benefit from its high computing power we must adapt our solution to a different architecture and programming model.

Since graphics processors are now commonly available on personal computers, we plan to use them to accelerate face recognition. If a volunteer possesses such hardware, he can choose to contribute to the project not only with his CPU, but also with his graphics processor.

Since automatic facial recognition is far from perfect, our system will also rely on human help to identify some faces. Faces that are recognized with a low level of confidence can be manually identified by users of the project.

The rest of the document is organized as follows. In Section 2 we will define the goals of our project. Section 3 provides a study of the State of the Art. Section 4 describes the proposed architecture of our solution, followed by the proposed methodology to its evaluation in Section 5. Finally, Section 6 offers some final remarks about our work.

2 Goals

The main goal of this work is to develop a free, efficient and scalable platform to automatically recognize and index people in multimedia content such as movies and photos. Further on, the system will be able to answer queries about the people present in each video. To achieve this, we will design a system that extracts frames from videos, detects the faces contained in each frame and then identifies those faces. To process video, extract images and detect faces, we will also use available algorithms. To recognize faces we will use a popular method named Eigenfaces.

Our system will not innovate in any face recognition, but we will develop a platform to support its computation. We will use BOINC [3], which is an existing platform for volunteer computing, and adapt it to our problem. Each stage of the facial recognition process will be distributed among volunteers. Our server will perform this scheduling according to the hardware available on each machine. To tackle the high demand for processing power from facial recognition algorithms, we will also make use of graphics processors available in the volunteers' computers. We believe that the high processing power available in these devices will greatly improve the performance of our project.

In order to reduce bandwidth and energy costs at the main server we will also create a Peer-to-Peer overlay of more capable computing nodes. This is an extension to the regular BOINC architecture and introduces a new entity in their client-server model. This overlay will be contactable by regular clients and will perform two tasks:

- aid the main server by serving data to clients and by doing a fine-grain scheduling of tasks;
- maintain a distributed database of metadata about each video.

The main server can forward queries about videos and persons to the this new network of volunteers.

3 Related Work

We will now provide a study on the related work of three different areas: face detection and recognition, graphics processors and public distributed computing. Section 3.1 will overview the algorithms used to detect and recognize faces. Section 3.2 will explore the architecture of graphics processors to understand if our solution is suitable to run on these devices. Finally, Section 3.3 will explore different systems for public distributed computing and understand their benefits and limitations.

3.1 Face Detection and Recognition

There is a diversity of practical applications for facial detection and recognition algorithms which led to several research approaches from different areas [61]. However, this is not the focus of our work. We will provide a brief overview of the existing methodologies to detect and identify faces. Our goal is to explore the possibility of running these algorithms in large scale to improve their performance.

3.1.1 Face Detection

Face detection is an important predecessor for facial recognition. Before identifying an unknown face, we have to detect and extract that face from an image. In a survey from 2002, Kriegman and Ahuja [30] classified facial detection algorithms into four categories:

Knowledge-based: these methods are based on rules that describe how a human face looks like. These rules usually involve the relation between facial features such as eyes, mouth and nose. One common problem of this approach is that it is difficult to translate human knowledge into rules that are not too strict nor too general. This method also has some difficulties to detect faces across different poses. An example is the multiresolution rule-based method proposed by Yang and Huang in 1994 [57].

Feature invariant: this is an opposite approach to knowledge-based methods that tries to find features in a face that are invariant across faces. These may include skin color, texture, facial features, among others. However, the lightning conditions and occlusion can severely corrupt the image features. Grouping of edges is an example of an approach that tries to highlight the face contour [34].

Template matching: in this method, a standard pattern is manually predefined or parametrized by a function. The correlation between this representation and the input image are computed for face detection. This approach is easy to implement, but it can not handle variations in scale, pose and shape. Examples include a predefined shape template [18] and an active model shape template [31]. The latter is a deformable template which tries to improve the flexibility in scale and shape variance of a face.

Appearance-based: Instead of using a template to match a face, appearance-based methods learn from a training set of images. The relevant features are identified in the initial set of images and these features are then used to detect new faces. This approach usually relies on a probabilistic model to identify the faces. Methods proposed within this approach achieved good results, but, like many others, sometimes fail to identify faces across different pose and illumination. The most prominent example of appearance-based methods is perhaps the method proposed in 2001 by Viola and Jones [55] which would later be considered one of the most important works of facial detection in the 2000's [59]. They achieved high success rates with high efficiency, making face detection implementation feasible in devices such as digital cameras.

3.1.2 Image preprocessing

Before facial recognition there is often a preprocessing stage in face images that reduces lightning variations and sometimes emphasizes edges or other image components. The goal of this stage is to improve the effectiveness of the recognition algorithm. [32]

One of the first preprocessing steps can be image scaling. Algorithms such as Eigenfaces require that all images have the same size. After that we can apply several different methods to reduce illumination variations. A basic approach involves converting the image to grayscale and then applying histogram equalization. This standardizes the brightness and contrast of the image and reduces lightning variations [53]. Further preprocessing could be done with edge enhancement or contour detection. Edge enhancement improves the sharpness of an image by enhancing the contrast on edges. Contour detection detects salient coarse edges that belong to the image boundaries.

Image preprocessing for recognition is also an active field of research and much more complex methods were proposed and tested with good results [8, 26].

3.1.3 Face Recognition

Facial Recognition is an attractive field for researchers from different areas namely psychology, pattern recognition, neural networks, computer vision and computer graphics. Such diversity resulted in different approaches for solving the problem. In Table 1 we categorize some of the most popular methods used to recognize faces in still two-dimensional images.

Table 1. Facial Recognition methods

Approach		**Representative Work**
Holistic methods		
	Eigenfaces	Direct application of PCA [54]
	Probabilistic eigenfaces	Two-class problem with prob. measure [42]
	Fisherfaces	FLD on eigenspace [7]
Feature-based methods		
	Pure geometry methods	Recent methods [17]
	Dynamic link architecture	Graph matching methods [56]
Hybrid methods		
	Modular eigenfaces	Eigenfaces and eigenmodules [50]

Holistic methods use the whole face as input to the recognition process. In 1991, Turk and Pentland [54] proposed an holistic method named Eigenfaces. This was the first successful demonstration of facial recognition by machines. This approach extracts relevant information from faces, encodes it efficiently and then compares it to a database of similarly encoded images. Through this comparison it is possible to infer if a new image represents a known human face.

Feature-based approaches emphasize local features such as the eyes, nose and mouth. These features are extracted and fed into the classifier. An example of this approach was proposed by Cox et al. [17]. Their approach achieves high success rates with a training database that contains only one image per person. They use a novel distance function based on statistics estimated with the whole database.

Finally, **hybrid methods** consider that both the local features and the whole face image are useful to recognize the face. One of these methods is Modular Eigenfaces [50]. This is an extension to the previous Eigenfaces method that considers not only the whole face, but also components such as the eye, nose and mouth. This increases the robustness of the algorithm and achieves high success with different head orientations.

There is also some work on three-dimensional face representations. This approach has some potential advantages over two-dimensional faces because it is viewport independent, shapes are more accurately described and lightning limitations are overcome. An overview of three dimensional facial recognition methods can be found in the surveys [10, 1].

Relevance to our work

We will employ the Eigenfaces method to recognize faces in our project. This method does not provide the highest success rates, but it is a simple and well known approach. Since this is not the main area of our study, we opted for an unsophisticated method that can still provide good results. However, we will try to improve its effectiveness by using a preprocessing step that hopefully eliminates lightning variations.

Another reason for choosing this algorithm is because it is a good candidate for achieving high speedups with graphics processors.

We will now explain in more detail the stages of the Eigenfaces algorithm.

The eigenfaces algorithm can be divided in two main segments: the creation of a database with known faces and the recognition of a new image.

The creation of the database follows these steps:

1. Acquisition of the initial set of images (training set) of the same size.

2. Transformation of each image into a vector in a two dimensional space.
3. Calculation of the eigenvectors of the covariance matrix composed by mean values of the set of training images.

This last step involves many arithmetic operations and is explained in detail in the original paper [54]. The calculated eigenvectors compose a set of features that together characterize the variation between face images. From these eigenvectors, we will chose only those with highest eigenvalues. These vectors will define the face space. By displaying the eigenvectors, we see a sort of ghostly face which is called eigenface. In Figure 1 we can see some face images and some of the corresponding eigenfaces.

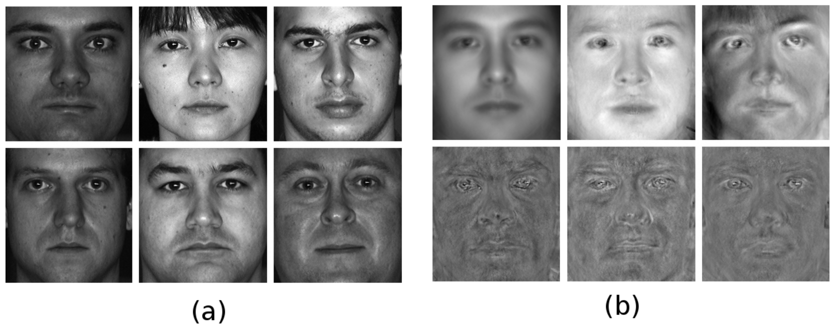


Fig. 1. (a) Examples of face images. (b) The leftmost in the first row is the average face, the others are top two eigenfaces. The second row shows eigenfaces with at least three eigenvalues.

The second part of the algorithm recognizes new faces by using our previously trained set. To do so, we must:

1. Calculate a set of weights based on the new input image and the face space by projecting the input image onto each of the eigenfaces.
2. Determine if it is a face by measuring the distance between the image and the face space.
3. If it is a face, use its weight pattern to determine if it belongs to a known person by determining if the distance among faces if under a specified threshold.

When a positive result is found, we can use the new face to update the values of our eigenfaces. If we find an unknown face several times, we can also calculate its characteristic weight and incorporate it in the list of known faces.

3.2 Graphics processing units

Fueled by the desire for realistic images, the graphics processing unit (GPU) evolved into a powerful and flexible processor that outpaces CPUs in a variety of parallel applications. Given the complexity of the Eigenfaces algorithm, we need to consider the use of widely available GPU cards to improve performance. In this section, we will talk about the architecture of a GPU and the operations it was designed to perform. We will also show related work of general purpose computation in graphical proessing units to try to understand if our problem suits the architecture of a GPU and if we can benefit from its use.

3.2.1 GPU Architecture

The Graphics Pipeline

The main task of most graphics processors is to synthesize images from given scene descriptions. Real-time graphics applications, such as video games, should produce images at a rate of about 60 times per second [39]. The processing model, used by most of today's GPUs, that transforms the input data into images is named graphics pipeline.

The graphics pipeline is designed to improve the graphics computation through parallel execution [48]. It receives a set of object vertexes as an input and it outputs an image. The data is processed through several stages which are sequentially executed. We will provide a brief overview of the classic pipeline stages to better understand the kind of operations a graphics processor is designed to deal with.

Each received vertex is expressed in its own local object coordinate system. The first stage will convert these vertexes from object space into screen space to define a common coordinate system. Each vertex is also shaded, typically through a lightning algorithm such as Phong. Finally, the vertexes are assembled to form triangles, the basic processing unit of the graphics processor. During this stage the GPU does heavy floating point operations on matrixes and vectors [39].

The second stage is rasterization. In this stage it is defined which pixels belong to each triangle and their color is calculated through interpolation. Each pixel can be treated independently from other pixels, therefore, this stage is highly parallelizable [39].

The next stage, fragment processing, calculates the final color for each pixel and adds textures. Since nearby pixels tend to access nearby texture images, the cache design helps to improve latency. This is typically the most computationally demanding stage [49].

Finally, to hide objects in the background, the fragments closer to the camera are chosen and the remaining are discarded. Figure 2 provides an overview of the mentioned stages.

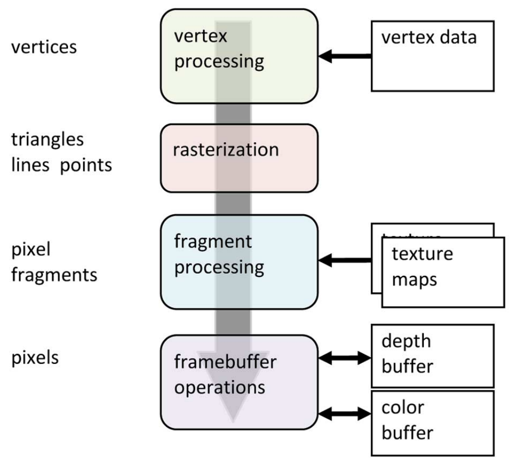


Fig. 2. The classic graphics pipeline [9].

Hardware Infrastructure

As seen in the previous section, the GPU is targeted for parallel operations that operate mostly on vectors and matrixes. Building custom hardware to exploit the high parallelism of graphical algorithms allows for significant performance improvements. Due to this fact, its architecture evolved in a very different way than common CPUs. In this section we will talk about the hardware architecture of a GPU and its programming model. At the end we will provide an example of a GPU from the latest generation of NVIDIA cards.

There are two ways to measure performance of a processing unit: task latency and throughput. Task latency is the time to complete a single task and throughput is the number of tasks done per unit of time. Common microprocessors focus mainly on improving task latency. On the other hand, GPUs must complete millions of small tasks within some unit of time. The time that each tasks takes to execute is unimportant. What matters is the time that it takes to finish all the tasks, hence the focus on throughput. With this in mind, there are three fundamental architectural characteristics present in a modern GPU: many simple processing cores, extensive hardware multi-threading and single instruction multiple data execution [23].

Many techniques were developed to improve a task's speed: out-of-order execution, speculative execution, sophisticated memory caches, among others. These techniques are extensively used by CPUs, but are usually absent in a GPU. Each processing core of a GPU is very simple: it executes each instruction sequentially avoiding branch prediction and similar techniques. This simplifications allows for savings in chip space and often improves the speed of execution of a single thread.

With abundant parallel computation, a multi-threaded approach is a natural choice. In a GPU, each thread is extremelly lightweighted and there can be thousands of threads running concurrently [29]. Such high amount of threads is used to hide latency in memory access. Unlike CPUs, GPUs usually don't have a cache hierarchy. Achieving high throughput in such conditions can only be done by running enormous amounts of parallel work at the same time.

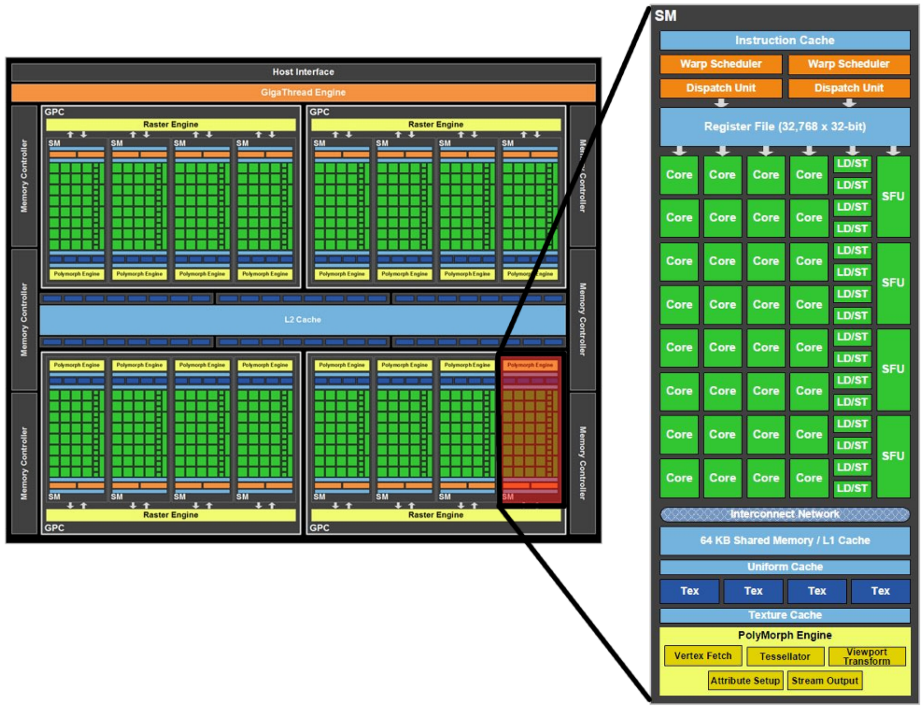


Fig. 3. The Fermi Architecture. On the right we can see a Streaming Multiprocessor in detail [46].

The Single Instruction Multiple Data (SIMD) paradigm was defined in Flynn's taxonomy. In this model, each instruction is simultaneously applied to multiple data, thus exploiting data parallelism. The GPU employs an architecture very similar to SIMD. The motivation behind this choice is that more processing units can be used if their logic is reduced. On the other hand, this could handicap performance if the threads follow different execution paths. Only one instruction can be processed at a time, therefore, threads that branch will be inactive while the others process their instruction.

The latest graphics processor architecture designed by NVIDIA[4] is named Fermi and features up to 512 processing cores organized in 16 streaming multiprocessors with 32 cores each as seen in Figure 3. The streaming multiprocessors handles all thread creation, resource allocation and scheduling. Each thread has dedicated registries, which means that there is no state to restore when switching between threads. With all this management done by hardware, the cost of using many threads is minimal.

Each streaming multiprocessor processes threads in groups of 32, called warps. This model is called Single Instruction Multiple Thread (SIMT) and it is very similar to SIMD. Each warp executes one instruction at a time across its threads. In the Fermi architecture, each SM can dispatch two warps at a time to sixteen cores. Other improvements over older GPUs include the addition of a L1 cache and ECC memory support.

3.2.2 General Purpose Programming on GPU

A GPU is an attractive device for computation that goes way beyond graphics. The high performance, low price and the constant hardware updates makes them a natural target for parallel computations and an alternative to a CPU.Figure 4 shows the performance evolution of both processors in the last decade. It is possible to observe that GPUs have evolved much faster than CPUs. In this section, we will provide an overview of general purpose computation on GPUs and some recent applications that are related to our work.

[4] http://www.nvidia.com

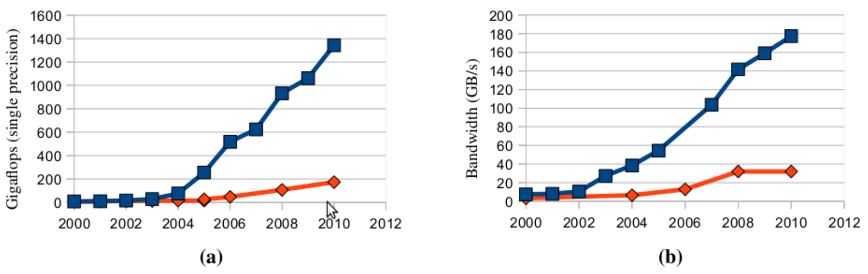


Fig. 4. Comparison between the evolution of CPUs (red dots) and GPUs (blue dots)

The use of general purpose computation on graphics processing units (GPGPU) began decades ago with machines like the Ikonas [19], the Pixel Machine [51] and Pixel Planes [22]. Also in the 80's, Pixar created Chap [36] and Flap [35] processors, which explored the SIMD paradigm. Note that the referred systems were graphics servers and not desktop workstations.

It was not until the 90's that the first applications running on desktop graphics appeared. In [33] robot motion planning was implemented using rasterization and in [27] Z-Buffer techniques were used to create Voronoi diagrams. Artificial neural networks and wavelet transformations were also implemented using GPUs.

During the 2000's, general purpose computation on graphics processing units has increased tremendously. The improvements on performance and programming flexibility combined with the wide-spread availability of low-cost accelerator cards, boosted their use by both researchers and the industry [45].

We will provide some examples of GPGPU applications in image processing and high performance computing, since these areas are related to the focus of our work. Other applications of GPGPU include physical simulations, database management, financial services and molecular biology. A detailed description of many GPGPU applications can be found in the surveys [48, 12, 9].

Image and Signal Processing

Yang et. al [58] used graphics processors to perform real-time image segmentation. Image segmentation classifies pixels as belonging to foreground or background objects and common techniques to achieve it involve pixel thresholding followed by morphological operations. Using an Nvidia Geforce 4 resulted in a 30% performance increase over an Intel MMX optimized code running on a 2.2Ghz Intel P4 CPU. Other segmentation techniques are also commonly used to process biomedical images. Hong and Wang [28] developed GPU based level set algorithms to process large images. An improvement of 12 to 13 times in processing speed was achieved by using a GPU instead of a CPU.

The Fast Fourier Transform (FFT) is one of the most important algorithms in image and signal processing. This is usually one of the first algorithms ported to new architectures [12]. There are inumerous GPU implementations with considerable gains when comparing to a CPU implementation. Moreland and Angel [43] ported the FFT to a GPU in 2003, but their implementation was only as fast as a highly optimized FFT library. Speedups could eventually be achieved if the input images were already present in the GPU. In 2008, Govindaraju et. all [25] implemented a high performance FFT in a GPU and measured speedups of 8 to 40 times over Intel implementation. Since 2007, NVIDIA also provides a FFT implementation for their hardware [47].

With such speed improvements, libraries with GPU implementations of image processing algorithms emerged. The popular image manipulation library OpenCV [11] has a GPU counterpart named GPUCV [21]. Their implementation was tested with algorithms such as the Hough transformation and histogram computing. GPUCV is up to 18 times faster than the native CPU implementation. Babenko and Shah [6] also developed a library to convert CPU code to GPU code named MinGPU. They implemented several computer vision algorithms such as the homographic 3D transformation. They achieved speedups of up to 600 times when comparing their implementation with C++ and Matlab implementations of the same algorithms.

High Performance Computing

To solve advanced computer problems, the industry and the scientific community rely on supercomputers or computer clusters. The adoption of GPUs for some tasks is also rising in this area.

Both NVIDIA and AMD [5] are producing GPUs specialized in general purpose programming. In 2006 AMD launched FireStream [2], the first dedicaded entreprise processor. A year later, NVIDIA introduced Tesla [37]. These devices have no graphical output so they can only be used as parallel computing devices. Such cards are already integrated in some of the fastest existing supercomputers. As of today, 3 of the top 5 supercomputers in the world are equiped with graphics cards[6].

In 2004, Fan et al. [20] elaborated on the advantages and difficulties of managing GPU clusters. The motivation behind this was the performance/price ratio and the speed of evolution of these devices. Nowadays, such clusters exist only in some institutions. Folding@home[7], SETI@home [4] and other public distributed compting applications also form a cluster of hybrid hardware. The users of these projects can download clients for CPUs, GPUs and other processors such as the Cell Boradband Engine [14].

Amazon Elastic Compute Cloud (EC2) is a webservice that provides computing capacity in the computer cloud. Since October 2010, they also have a GPU cluster available for renting for users that want to benefit from high throughput rates in their applications.[8] Currently, the cluster is composed by NVIDIA Tesla Fermi GPUs.

3.2.3 Software Environment

In the past, general purpose programming on a GPU was done using a graphical Application Programmer Interface (API). This was an obvious drawback since intrinsic knowledge of graphical libraries was required. Besides, it is difficult, sometimes even impossible, to express a general problem in graphical primitives. With the introduction of fully programmable shaders in the 2000's, the history of general purpose programming on GPUs took a new turn. The possibility of writing code that will be run by each processor inside a GPU gave birth to a new generation of high level languages specifically designed for GPUs.

Sh[9] and Brook [13] were two early academic projects that aimed at creating a programing language for general purpose programming on a GPU. Sh was commercialized by RapidMind [41] and in August 2009 it became a part of Intel technologies. The source code is still available under the GNU Lesser General Public License, but it is no longer actively maintained. Brook was extended by AMD in December 2007. The project was named ATI Brook+. However, after AMD joined the OpenCL working group in 2008, its development started slowly decreasing and it was discontinued by AMD after the release of ATI Stream SDK 1.4 in 2009.

Nowadays, there are two leading technologies that perform general purpose computation on graphics hardware: CUDA [29] and OpenCL [44]. Microsoft also maintains two other API's based on the DirectX runtime: Accelerator and DirectCompute. Unfortunately, they are both immature technologies with little support and documentation.

Computer Unified Device Architecture (CUDA) is a parallel computing architecture developed by NVIDIA. CUDA is tied to its manufacturer cards which is an advantage for performance and a disadvantage for portability. The programming language is an extension of C with C++ features, but there are wrappers for other languages such as Python, Ruby and Java. The GPU is programmed using kernels which are executed by all the processing cores of the GPU in parallel. The CUDA Toolkit also includes libraries such as CUBLAS, CUFFT and CUSPARSE which contain CUDA implementations of some algorithms.

OpenCL (Open Compute Language) is an open standard for parallel programming of heterogeneous hardware, maintained by the Kronos Group. It supports a wide range of hardware, from GPUs and CPUs to embed devices. The ratification board of OpenCL is composed by major hardware and software companies, such as Apple, AMD, Intel, NVIDA and IBM. This may be a sign that this technology will be as successful as OpenMP [12].

The language itself has many similarities with CUDA. It also uses kernels which are programmed in C99 with some additional extensions. Currently, all OpenCL implementations are closed source and are freely available from Apple, AMD, NVIDIA and recently Intel.

CUDA is developed since 2007 and nowadays it is a mature and highly optimized library for NVIDIA GPUs. On the other hand, OpenCL is a very recent technology. The first SDK version was released in late 2009 by NVIDIA. In spite of being a promising technology, it is still not as fast and complete as CUDA.

[5] http://www.amd.com
[6] The list is available at http://top500.org/
[7] http://folding.standford.edu
[8] Pricing and service description at http://aws.amazon.com/ec2/
[9] http://www.libsh.org

3.2.4 Relation to our work

Eigenfaces is a resource demanding algorithm. As described in section 3.1, calculating the Eigenfaces and the face space of the images involves several operations with matrixes. We believe that the natural parallelization of such algorithms will certainly benefit from the GPU hardware infrastructure. Other operations involving image and video processing are also great candidates to benefit from the GPU processing power. We provided several examples of notorious speedups achieved by using such devices.

As for the software library, the natural choice for our project will be OpenCL because it is an open standard supported by many companies and is not restricted to a particular brand of GPUs.

3.3 Public Distributed Computing

Grids emerged as a promising solution for high performance computing. A grid system aggregates a large group of heterogeneous resources from multiple institutions and tries to provide a coordinated, secure and transparent access to them. Typically, a grid has many dispersed components such as supercomputers, clusters, database storage, among others. However, it is difficult for a common user to have access to a such system. There are strict rules defined by the organizations that one must follow to access these resources. On the other hand, public computing harvests idle cycles from personal devices around the world. People can volunteer to do science with their computer just by downloading an application that will perform computation when the computer is idle. This is a more open approach, but also has its downsides. It is a much more heterogeneous and unreliable environment and the computers are not trustworthy, which raises some security issues. The main advantage of public computing are the low execution costs and flexibility.

There are several infrastructures that support public computing with different architectural approaches. We will distinguish between two types of systems based on their network model: centralized and decentralized. We will now elaborate on the differences and cite some examples.

3.3.1 Centralized Desktop Grids

In a centralized system there are three entities: a data provider, a main server and volunteers. The data provider feeds the network with work, usually through the main server. This data is divided in smaller jobs by the main server. An application that processes this data is usually also provided to the server. The main server then sends the jobs and the application to volunteers and waits for the results. When the results are received, they must be checked before sending them back to the data provider.

The role of a volunteer is simple. Before participating, they need to register themselves in the main server. After that, they just wait for available work to process and send the results back to the main server when the computation is done.

We will now explain in more detail two centralized systems: BOINC and XtremWeb [24].

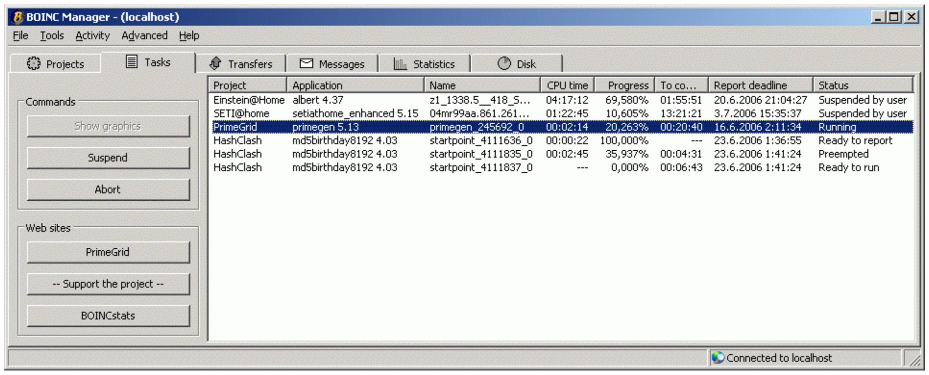
BOINC

Berkeley Open Infrastructure for Network Computing (BOINC) was originally developed to support the SETI@home [4] project, but it evolved to support other applications. BOINC has been actively developed since 2004 and has received widespread adoption. Many popular projects such as Rosetta@home[10] or Climateprediction.net[11] rely on the BOINC infrastructure.

The BOINC architecture is composed by clients, a main server and a data provider. They interact as described in the beginning of this section. In BOINC, the data to be processed received by each client receives is called a working unit. The main server schedules these working units according to the clients' hardware.

Each user that wants to donate computation time from his personal computer must download the BOINC application available at their website. After installing the application, the user can choose to join any number of projects. In Figure 5 we can see a running BOINC client with several active projects. The user can manually suspend and resume work from projects and several settings can be defined such as a threshold for disk usage, CPU usage.

[10] http://boinc.bakerlab.org/rosetta/
[11] http://climateprediction.net/



Project	Application	Name	CPU time	Progress	To co...	Report deadline	Status
Einstein@Home	albert 4.37	z1_1338.5__418_S...	04:17:12	69,580%	01:55:51	20.6.2006 21:04:27	Suspended by user
SETI@home	setiathome_enhanced 5.15	04mr99aa.861.261...	01:22:45	10,605%	13:21:21	3.7.2006 15:35:37	Suspended by user
PrimeGrid	primegen 5.13	primegen_245692_0	00:02:14	20,263%	00:20:40	16.6.2006 2:11:34	Running
HashClash	md5birthday8192 4.03	startpoint_4111636_0	00:00:22	100,000%	---	23.6.2006 1:36:55	Ready to report
HashClash	md5birthday8192 4.03	startpoint_4111835_0	00:02:45	35,937%	00:04:31	23.6.2006 1:41:24	Preempted
HashClash	md5birthday8192 4.03	startpoint_4111837_0	---	0,000%	00:06:43	23.6.2006 1:41:24	Ready to run

Fig. 5. BOINC client for Windows running PrimeGrid

XtremWeb

XtremWeb was one of the early projects to exploring the power of distributed public computing. Its architecture is close to the regular client/server model. However, it has three entities in the system: the main server/data provider, the workers and the controllers. The workers contact the main server to get data and applications. The results are sent to the controllers which manage resource availability and scheduling.

XtremWeb relies on Remote Procedure Calls (RPC) for communication. They also provide several levels of security such as data privacy protection and result validation. The system is also fault-tolerant and uses checkpointing.

There are two deployments of XtremWeb [12]: XtremWeb for High Energy Physics and XtremWeb-CH.

3.3.2 Decentralized Desktop Grids

Before explaining the details of decentralized public computing, we will elaborate on the definition of Peer-to-Peer since this is a common architectural model in these systems.

Peer-to-Peer (P2P) is a distributed computing architecture that partitions work between clients, which are called peers. P2P is more directed towards direct sharing between nodes instead of relying on intermediate or central servers [5]. This is one of the greatest advantage of these systems. The fact that there is no dependence on central servers avoids problems such as bottlenecking, single point of failure, among others. P2P is becoming very popular in the Internet, mainly due to file sharing services that use this protocol, such as BitTorrent. Many research has also been devoted to P2P, particularly with the goal of creating distributed hash tables such as Chord [52] or Tapestry [60].

Decentralized public computing is based in Peer-to-Peer. Since there are no central servers, volunteers also called nodes communicate directly among each other. For a volunteer to register in the network he must contact one of the nodes. The jobs are submitted to a node by a data provider and then the nodes distribute the jobs among themselves. Each volunteer then begins the computation. The results are submitted to a trusted peer which verifies the results before returning them to the data provider.

We will now analyze a decentralized system proposed in 2004 named Cluster Computing On the Fly (CCOF) [38].

Cluster Computing On the Fly

Cluster Computing On the Fly was developed at the University of Oregon and it seeks to harvest cycles in an open-access non institutional environment. In CCOF the nodes are loosely coupled and organized in a P2P network. Volunteers join a chosen network overlay depending on how they want to spend their cycles. The data provider then forms clusters on the fly by discovering and scheduling data to machines from these overlays. The scheduling is specialized in harvesting night cycles. This is done through a distributed hash table that migrates nodes in different timezones. To deal with the insecurity of public computing, CCOF uses a reputation system. Result verification is done by sending dummy tasks to volunteers with known results. If these tasks are answered correctly, it is assumed that the node works properly.

[12] http://www.xtremweb.net/deployment.html

3.3.3 BOINC extensions

BOINC has become one of the most popular platforms of cycle sharing. However, its centralized approach may cause a bottleneck in some situations. This is also an area of research and some extensions to the BOINC client-server model have proposed. In this section we will analyze some of the existing options to decentralize the BOINC architecture.

Integrating P2P capabilities in the BOINC network poses a number of challenges that must be solved [16]. Most personal computers are hidden behind a router. This means that it is necessary to bypass a Network Address Translation (NAT) protection to establish a connection between two clients. Solving this problem may involve complex mechanisms, such as hole punching or, alternatively, each user could manually configure his router to allow incoming connections.

Data integrity is another problem of a decentralized untrusted network. Malicious users could exploit the system by corrupting data. This is an important problem, but it goes beyond the scope of our project. Depending on the network layout, some security measures must be implemented by the main server to ensure the validity of the data.

Finally, the integration with the BOINC client must be done through a C++ implementation or binding, which also poses some implementation problems.

In 2007, Costa et. al [16] studied two approaches to integrate a P2P network in BOINC.

The first scenario describes the application of a BitTorrent [15] protocol in BOINC. The Bittorrent protocol uses a central server to control the list of files available in the network. Anyone who wants to download a certain file must first contact the central server. After that, the user will be able to download the file he wants from anyone that has it. When the download completes, he can also serve that file to others. Applying this protocol in BOINC would mean that each input data file could be downloaded from all the other clients. Obviously, the BOINC server would be the first computer to serve the file. This could be useful if the same input file is used several times to perform calculations from different clients.

The other studied approach is a platform named P2P-ADICS [40]. This project was created by researchers at Cardiff University as part of the European CoreGRID project. Their purpose is the introduction of an additional category of nodes in BOINC, besides the main server and the clients, named data caching peers. These are trusted datacenters that store data provided by the main BOINC server. As seen in Figure 6, the data caching peers send data to clients and also communicate among each other to replicate data.

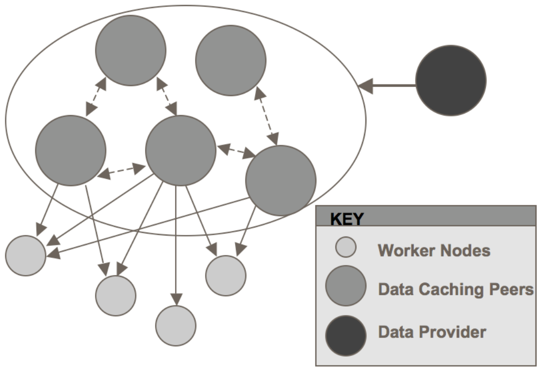


Fig. 6. P2P-ADICS Architecture [16].

This project creates a clear separation between trusted storage nodes and worker nodes that perform computation. P2P-ADICS is currently implemented as a BOINC extension in the Java language.

3.3.4 Public Computing Projects

To end this section of our related work we will provide some examples of active public computing projects and their goals. From the mentioned systems, BOINC is the most used. It encompasses a great number of projects, perhaps the most famous being SETI@home [4]. SETI@home is a project from the University of California, Berkeley. SETI is an acronym for the Search for Extra-Terrestrial Intelligence. Its purpose is to analyze radio emissions from outer space and search for signs of extra-terrestrial life.

There are other popular projects, but they are usually based in their own middleware such as GIMPS, Distributed.net and Folding@home. GIMPS[13] and Distributed.net[14] were among the earliest public computing projects in the 1990's. The Great Internet Mersenne Prime (GIMPS) aims at finding Mersenne prime numbers. As of today, 13 primes were already found and 11 of them are the largest known Mersenne Primes. Distributed.net solves general purpose problems and their current project is trying to break RC5 with a 72-bit key.

Folding@home [15] is a project from the Standford University and their goal is to understand protein folding, misfolding and related diseases such as Alzheimer's, ALS and Huntington's. According to their website, they have about 500 thousand active clients providing more than 5 PetaFLOPS of power. This is two times better than the performance of the fastest supercomputer[16].

3.3.5 Relation to our work

We will use BOINC to support the underlying communication infrastructure of FaceID@home. BOINC is the most popular public distributed computing project with hundreds of thousands of active users. Other systems often lack software maturity and documentation and don't have the support of a large community.

However, the high demands of our application in terms of data transmission has led us to propose a new extension to BOINC. This extensions aims at improving the scalability and performance of the project by decentralizing part of the architecture. This extension will be described in detail in Section 4.

4 Architecture

We will now describe the proposed architecture for FaceID@home. In Section 4.1 we will begin with a general overview of the tasks performed. Section 4.2 describes the entities of our system and the changes made to the regular BOINC model. Section 4.3 focuses on the network structure and the interconnection between computers. Finally, section 4.4 details the protocol used for scheduling and resource management.

4.1 Workflow

The main goal of our system is to recognize faces in photos and videos. We consider that a photo is treated like a video frame, so our description will focus on videos. Figure 7 shows an overview of the identification process, which we will now describe.

1. **Split the video:** if a video is too large, it must be split in several parts. This is not a requirement, but we may need to do it due to bandwidth constraints from the users. The number of parts and its size depends on the video size and the available clients' bandwidth.
2. **Extract frames:** now we must extract still images from the movie, but not all frames need to be extracted. We can extract only a few frames per second for example. Other techniques that detect scene changes and movement can also be used to try to minimize the number of extracted images.
3. **Detect faces:** in each frame, we will detect faces. To do this, we will use the algorithm proposed by Viola and Jones [55].
4. **Extract faces:** the previously detected faces will now be extracted from the frames.
5. **Normalize and preprocess:** as described in Section 3.1.2 , we will do a preprocessing stage before performing the actual face recognition using the Eigenfaces method [54]. This will involve image scaling to adjust size and other techniques to reduce light variations.
6. **Calculate the new eigenface:** as described in Section 3.1.3, we need to calculate the eigenface of the new image before performing the match against the database.
7. **Match against the face database:** finally, we will try to identify the person using the database of known faces.

Our system will perform three additional tasks. One of them is the possibility to manually confirm identified faces. Eigenfaces is not a perfect method and not all faces will be identified with a high level of confidence. To solve this problem we will provide the possibility for an administrator to confirm the identification of some faces. Users could also contribute to this manual confirmation through the project

[13] http://mersenne.org/default.php
[14] http://www.distributed.net
[15] http://folding.standford.edu
[16] http://top500.org

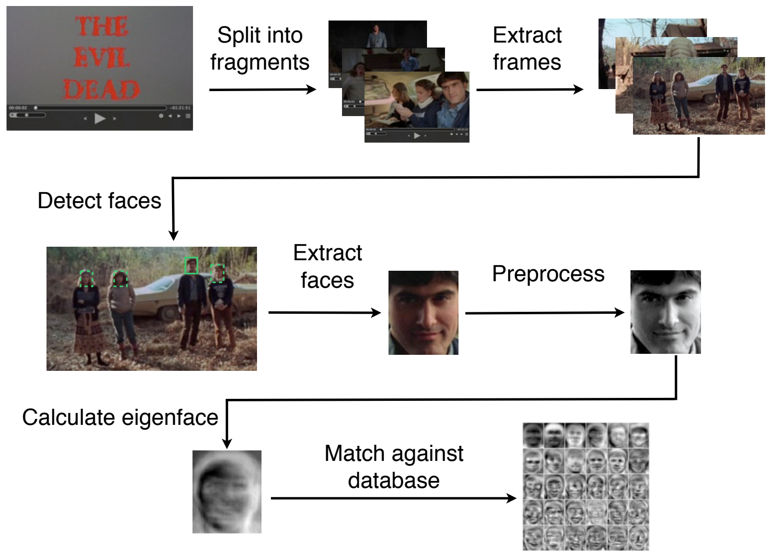


Fig. 7. Required steps to perform facial recognition in a video.

website. However, our goal is to maximize the automation of the process and try to reduce this manual interaction by improving the effectiveness of the recognition algorithm.

The second additional task is recalculating the face space when new faces are recognized. This will improve the accuracy of the algorithm and allow it to expand the database of known faces.

Finally, our system will do video indexing, which is unrelated to the identification process, but benefits from its results. This allows to answer queries related to the people present in the movies. Two example are:

– In which movies does a certain person appear?
– Who appears in a certain movie?

To do this, the results of the facial recognition process must be integrated in a database that stores metadata from movies and persons. This metadata must contain additional informations such as the person name and in which scenes he or she appears. This data could also be cross referenced to answer more complex queries.

4.2 System entities

We propose an architecture based on the BOINC model in which a volunteer pulls tasks from a main server. Each task is one of the stages explained in Section 4.1 and will be scheduled according to the available hardware in volunteers.

Some stages of the facial recognition workflow described in Section 4.1, specially the first ones, are data intensive. Sending movies and receiving parts of movies from clients could cause a bandwidth bottleneck in the main server.

We should also note two other things about the workflow. One of them is that the server is only interested in the final results and, perhaps, other small intermediate results about the detected faces. The other is that there are some precedences among tasks because the eigenfaces calculation requires preprocessed faces, which requires extracted faces and so forth. With this in mind, we reasoned that if clients could transmit data between them, we would reduce bandwidth usage and maintenance costs in the main server and improve the overall scalability of the system.

As discussed in Section 3.3.3, decentralizing the BOINC model by allowing direct communication between clients poses several difficulties, mainly due to network connectivity issues. Instead of a fully decentralized approach, the solution we propose creates a small group of trusted nodes that are well connected. Each node of this new group is called a controller. A controller is a computer from a volunteer that gathers the necessary requirements. The most important requirements in a controller are:

– **High connectivity**: a controller must have a properly configured network to be able to receive requests from other computers.

– **Availability**: a controller must be online most of the time.
– **High Bandwidth**: sending and receiving data from a controller should be fast.
– **Available storage**: there should be a fair amount of free storage that can be used by the system

4.3 Network Architecture

There are three entities in our system: the main server, the controller and the client or volunteer. In Figure 8 we show the communication between the controller and the main server. They interact using a pull approach. The controller asks the server for work and the server returns available working units. The controller will usually receive data intensive tasks, which relate to the initial stages of facial recognition: split a video into fragments or extract frames from video fragments. When new faces are identified, the controller reports the results back to the main server.

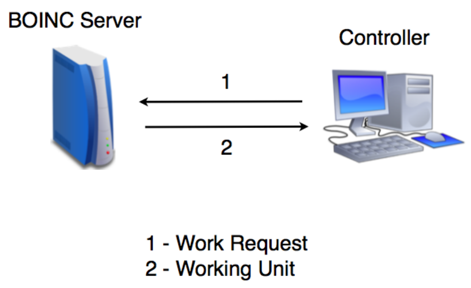


Fig. 8. A controller contacts the main server and receives work.

In Figure 9 we show the role of a client in our system. The first request for work from the client goes to the main server. The server may send a working unit directly to the client or instruct him to contact a specific controller. The first case follows the regular BOINC approach, so we will analyze the second case with more detail.

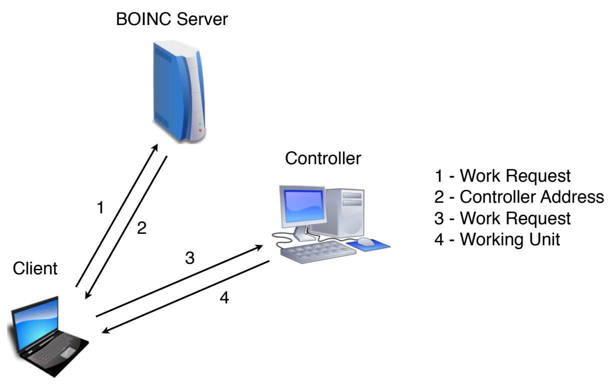


Fig. 9. A client contacts the BOINC server to receive work and is redirected to a controller.

When the server returns to a client an address of a controller, that client will now ask the controller for available work. The controller becomes responsible for assigning work to that client and for receiving the results of its computation. When a client finishes the work, it informs the main server before sending the results back to the controller. This means that despite the introduction of controllers, the main server continues to control the network and to aggregate all final results. The association between a client and a controller can be terminated at any time by the main server.

We should note that a node running the controller extension is also running a client. Therefore a controller can run work acquired from the server, from another controller or from itself. The most common case is that it will run work from itself to avoid data transfer.

Another situation where we can benefit from the use of controllers is to answer search queries. Our database of metadata with information about people and videos is stored in the main server but will also be distributed among the controllers. They will form a Peer-to-Peer network that can be used to search video information. Such network could be based on distributed hash tables systems, such as Chord [52] or Tapestry [60]. In Figure 10, we can see an overview of this system. When a new request arrives, the main server can redirect it to the controllers' network. This will also lighten the load on the main server by using volunteer's resources.

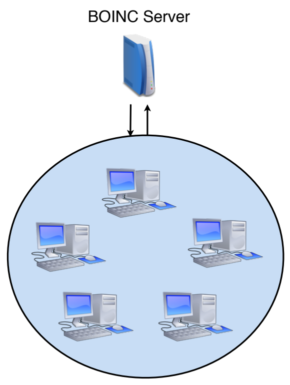


Fig. 10. The main server uses the controllers' network to answer queries.

4.4 Scheduling Protocol

As described in the previous section, the main server can assign work to a client directly or indirectly through a controller. We will now elaborate on how these decisions are done.

When the BOINC client connects to a server, a description of the host's hardware is sent. The server also has information on the requirements for each kind of task. For example, the requirements for the Eigenfaces calculation are: high processing, low memory, and the GPU is preferred. The chosen clients to execute this code should have a powerful GPU or CPU and there are little memory and bandwidth constraints. Based on the hardware description of the client and on the tasks' requirements, the most suited task for the client is chosen. Now we can send tasks directly to the client or assign a controller to this client. Work available on controllers has priority over work available directly from the main server.

Choosing an appropriate controller follows a similar strategy. The main server has global knowledge of the system. Therefore it can choose an appropriate controller based on the type and amount of pending work that each controller has. The preferred tasks sent to controllers are mostly most data intensive. Not only due to their high bandwidth requirements, but also because these tasks are the predecessors to other tasks that can be done by clients.

4.5 GPU client

As described in Section 3.2, the GPU has become a powerful parallel processor. Its architecture and programming model have some limitations, therefore not all tasks are adequate to run on a GPU. However, we believe that facial recognition is a good candidate to achieve a considerable speedup by using a GPU. The Eigenfaces algorithm heavily relies on matricial operations that are by nature highly parallelizable. Even the other stages of our workflow, that involve heavy video and image processing, are certainly tailored to such architecture. We identified several cases of image and signal processing algorithms that achieved great speedups. This is no surprise because a GPU was designed to process images.

Given the wide availability of GPUs on consumers' computers, we will also develop a GPU application to increase the performance of facial recognition for FaceID@home.

5 Evaluation

We will evaluate our work using three distinct assessment processes: qualitative, quantitative and comparative. Qualitative measures evaluate the correct implementation of the proposed functionalities. Quantitative measures are related to performance. Finally, the comparative approach will compare our system with the traditional BOINC and other Peer-to-Peer alternatives.

Qualitative:

- Correct recognition of faces in videos.
- Correct processing of queries about the contents of videos.
- Correct calculation of Eigenfaces by the GPU application.
- Automatic scheduling of tasks and handling of queries in a controller.
- Adequate scheduling of tasks to the clients and to the controllers from the main server.

Quantitative:

- Processing time of each stage of the workflow.
- Response time to a query.
- Speedup of the GPU application.
- Speedup of the distributed execution against the local execution of facial recognition.

Comparative: the comparison of our work with other systems will be done in terms of:

- Scalability of the system under heavy load.
- Adequate task scheduling.
- Ratio of volunteers' resources usage.
- Speedup of the distributed execution of facial recognition.

6 Conclusion

Over the last years we experienced a revolution in the creation and dissemination of media content such as videos and photos. However, this information is only indexed by manually defined tags and descriptions which are often insufficient to fully describe the contents of a video.

We believe that finding people is one of the most desired features for a multimedia search engine. Due to the enormous amounts of available content, manually tagging each person on a video would be an unfeasible task.

The main goal of this work is to develop a free, efficient and scalable platform to automatically recognize and index people in multimedia content such as videos and photos.

There are several algorithms available for face detection. In this work we will use the appearance based method proposed by Viola and Jones which has achieved high success rates. For face recognition, we will use the Eigenfaces algorithm, which is a simple, but effective approach. To deal with the high computational demands from video processing and recognition algorithms, our system will be based on BOINC, which is a public distributed infrastructure. We will adapt its centralized model to a more decentralized approach based on Peer-to-Peer. To improve the performance of the application running on each volunteer's computer, we will implement facial recognition in the GPU, using the OpenCL library. We believe that facial recognition is a suited task for a graphics processor and could greatly benefit from its high throughput architecture.

To evaluate our solution we will use qualitative, quantitative and comparative methods.

We this work we hope to contribute to answer the need for a global system that automatically identifies people over publicly available multimedia content.

References

1. A. Abate, M. Nappi, D. Riccio, and G. Sabatino. 2D and 3D face recognition: A survey. *Pattern Recognition Letters*, 28(14):1885–1906, Oct. 2007.
2. AMD. AMD Introduces World's First Dedicated Enterprise Stream Processor, 2006.
3. D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

4. D. Anderson, J. Cobb, E. Korpela, and M. Lebofsky. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
5. S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, Dec. 2004.
6. P. Babenko and M. Shah. MinGPU: a minimum GPU library for computer vision. *Journal of Real-Time Image Processing*, 3(4):255–268, May 2008.
7. P. Belhumeur, J. Hespanha, and D. Kriegman. Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, July 1997.
8. D. Beymer. Face recognition under varying pose. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 756–761, 1994.
9. D. Blythe. Rise of the Graphics Processor. *Proceedings of the IEEE*, 96(5):761–778, 2008.
10. K. Bowyer, K. Chang, and P. Flynn. A survey of approaches and challenges in 3D and multi-modal 3D+2D face recognition. *Computer Vision and Image Understanding*, 101(1):1–15, Jan. 2006.
11. G. Bradski and A. Kaehler. *Learning OpenCV: computer vision with the OpenCV library.* O'Reilly Media, 2008.
12. A. Brodtkorb, C. Dyken, T. Hagen, and J. Hjelmervik. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33, 2010.
13. I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. In *ACM SIGGRAPH 2004 Papers*, pages 777–786. ACM, 2004.
14. T. Chen, R. Raghavan, J. Dale, and E. Iwata. Cell Broadband Engine Architecture and its first implementation: A performance view. *IBM Journal of Research and Development*, 51(5):559–572, 2007.
15. B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer systems*, 6:68–72, 2003.
16. F. Costa, L. Silva, I. Kelley, I. Taylor, P. Ii, and P. Marrocos. Peer-To-Peer Techniques for Data Distribution in Desktop Grid Computing Platforms, 2008.
17. I. Cox, J. Ghosn, and P. Yianilos. Feature-based face recognition using mixture-distance. In *Proceedings CVPR'96 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 209–216, 1996.
18. I. Craw, D. Tock, and A. Bennett. Finding face features. In *Computer Vision—ECCV'92*, pages 92–96, 1992.
19. J. England. A system for interactive modeling of physical curved surface objects. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 336–340. ACM, 1978.
20. Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. GPU Cluster for High Performance Computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, number 1, page 47. IEEE Computer Society, 2004.
21. J. Farrugia, P. Horain, E. Guehenneux, and Y. Alusse. GPUCV: A Framework for Image Processing Acceleration with Graphics Processors. In *2006 IEEE International Conference on Multimedia and Expo*, pages 585–588. IEEE Computer Society, July 2006.
22. H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. *ACM SIGGRAPH Computer Graphics*, 23(3):79–88, 1989.
23. M. Garland and D. Kirk. Understanding Throughput Oriented Architectures. *Communications of the ACM*, 53(11):58–66, 2010.
24. C. Germain, V. Neri, G. Fedak, and F. Cappello. XtremWeb: building an experimental platform for Global Computing, 2000.
25. N. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli. High performance discrete Fourier transforms on graphics processors. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, number November, pages 1–12. IEEE Computer Society, Nov. 2009.
26. R. Gross and V. Brajovic. An image preprocessing algorithm for illumination invariant face recognition, 2003.
27. Hoff III, Kenneth E, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM/Addison-Wesley Publishing Co., 1999.
28. J. Hong and M. Wang. High speed processing of biomedical images using programmable GPU. In *ICIP'04. 2004 International Conference on Image Processing*, volume 4, pages 2455–2458. IEEE Computer Society, 2004.
29. D. Kirk. NVIDIA cuda software and gpu parallel computing architecture. In *Proceedings of the 6th international symposium on Memory management - ISMM '07*, pages 103–104. ACM Press, 2007.
30. D. Kriegman and N. Ahuja. Detecting faces in images: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58, 2002.

31. A. Lanitis, C. Taylor, and T. Cootes. Automatic face identification system using flexible appearance models. *Image and Vision Computing*, 13(5):393–401, 1995.
32. A. Lemieux and M. Parizeau. Experiments on eigenfaces robustness. In *International Conference on Pattern Recognition*, pages 421–424. IEEE Computer Society, 2002.
33. J. Lengyel, M. Reichert, B. Donald, and D. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *ACM SIGGRAPH Computer Graphics*, 24(4):327–335, Sept. 1990.
34. T. Leung, M. Burl, and P. Perona. Finding faces in cluttered scenes using random labeled graph matching. In *5th IEEE International Conference on Computer Vision*, page 637. IEEE Computer Society, 1995.
35. A. Levinthal, P. Hanrahan, M. Paquette, and J. Lawson. Parallel Computers for Graphics Applications. *ACM SIGARCH Computer Architecture News*, 15(5):193–198, 1987.
36. A. Levinthal and T. Porter. CHAP - A SIMD Graphics Processor. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, volume 18, pages 77–82. ACM, 1984.
37. E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro*, 28(2):39–55, Mar. 2008.
38. V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster Computing on the Fly : P2P Scheduling of Idle Cycles in the Internet, 2004.
39. D. Luebke and G. Humphreys. How GPUs Work. *Computer*, 40(2):96–100, Feb. 2007.
40. C. Mastroianni, P. Cozza, D. Talia, I. Kelley, and I. Taylor. A scalable super-peer approach for public scientific computation. *Future Generation Computer Systems*, 25(3):213–223, Mar. 2009.
41. M. Mccool. Data-Parallel Programming on the Cell BE and the GPU using the RapidMind Development Platform. In *GSPx Multicore Applications Conference*, 2006.
42. B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):696–710, 2002.
43. K. Moreland and E. Angel. The FFT on a GPU. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 112–119. Eurographics Association, 2003.
44. A. Munshi. OpenCL: Parallel Computing on the GPU and CPU, 2008.
45. B. Neelima and P. Raghavendra. Recent Trends in Software and Hardware for GPGPU Computing: A Comprehensive Survey. In *5th International Conference on Industrial and Information Systems, ICIIS 2010*, pages 319–324. ACM, 2010.
46. NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Fermi, 2009.
47. NVIDIA. CUDA CUFFT Library, Dec. 2010.
48. J. Owens, D. Luebke, and N. Govindaraju. A Survey of General Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.
49. J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.
50. A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proceedings CVPR'94., 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 84–91. IEEE Computer Society, 2002.
51. M. Potmesil and E. Hoffert. The Pixel Machine: A Parallel Image Computer. *ACM SIGGRAPH Computer Graphics*, 23(3):69–78, 1989.
52. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM, 2001.
53. M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, Nov. 1991.
54. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.
55. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, pages 511–518. IEEE Computer Society, 2001.
56. L. Wiskott, J. Fellous, N. Kuiger, and C. Von der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775–779, 2002.
57. G. Yang and T. Huang. Human face detection in a complex background. *Yang, G. and Huang, T.S.*, 27(1):53–63, 1994.
58. R. Yang and G. Welch. Fast Image Segmentation and Smoothing Using Commodity Graphics Hardware. *Journal of graphics tools*, 7(4):91–100, 2002.
59. C. Zhang and Z. Zhang. A Survey of Recent Advances in Face Detection, 2010.
60. B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer*, 74:11–20, 2001.
61. W. Zhao, R. Chellappa, P. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys (CSUR)*, 35(4):399–458, 2003.

Appendix A: Planning

Month	**Planning**	
February	Setup and configuration of a BOINC server	Writing the document
	Creation of a modular application to implement the workflow described in 4.1	
March	Integration of the CPU client in BOINC	
April	Implementation and integration of a GPU client in BOINC	
May	Implementation and integration of the controller extension in the BOINC architecture	
June	Validation and Debug	
	Performance analysis	
July	Revision and delivery	