Speeding Up Computer Vision Applications on Mobile Computing Platforms

Luna Backes

Supervisor:
Dr. Björn Franke
School of Informatics,
The University of Edinburgh

Academic tutor:
Dr. Xavier Martorell
Departament d'Arquitectura de Computadors

Bachelor Degree in Informatics Engineering

Specialisation in Computer Engineering

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

January 28th, 2015

A mi familia

Abstract

Computer vision (CV) is widely expected to be the next "Big Thing" in mobile computing. For example, Google has recently announced their project "Tango", a 5-inch Android phone containing highly customized hardware and software designed to track the full 3-dimensional motion of the device as you hold it while simultaneously creating a map of the environment.

One of the problems yet to solve is how to transfer demanding state-of-the-art computer vision algorithms —designed to run on powerful desktop computers with several graphics processing units (GPUs)— onto energy-efficient, but slow embedded GPUs found in mobile devices.

This project investigates ways of speeding up computer vision kernels and applications through optimisation and parallelisation. We took a representative example of a CV application, the KinectFusion, and we ported it to a mobile platform using OpenCL. Then, we conducted a performance evaluation, identifying performance bottlenecks and further optimise performance. We finally broaden our focus and studied its performance on a different platform to evaluate the performance portability of our optimisations.

KEYWORDS: OpenCL; Embedded GPUs; Computer Vision; KinectFusion

Resumen

La visión por computador (CV) se espera que sea la próxima ”revolución”, en el campo de la computación móvil. Por ejemplo, Google ha anunciado recientemente su proyecto ”Tango”, un teléfono de 5 pulgadas Android que contiene hardware y software diseñado para realizar el seguimiento del movimiento en 3 dimensiones del dispositivo que, mientras se sostiene, crea simultáneamente un mapa del entorno.

Uno de los problemas que queda por resolver es cómo utilizar lo último sobre algoritmos de visión por ordenador —diseñados para ejecutarse en ordenadores de escritorio potentes con varias unidades de procesamiento gráfico (GPUs)— en GPUs integradas de dispositivos móviles, que son eficientes energéticamente pero más lentas.

Este proyecto investiga la manera de acelerar núcleos y aplicaciones de visión por computador a través de diversas técnicas de optimización y paralelización. Hemos seleccionado una aplicación representativa de CV, KinectFusion, para portarla a una plataforma móvil usando OpenCL. A continuación, hemos realizado una evaluación de rendimiento, hemos identificado los cuellos de botella y hemos optimizado aún más el rendimiento. Por último, hemos estudiado brevemente su rendimiento en otra plataforma móvil diferente para evaluar la portabilidad del rendimiento de nuestras optimizaciones.

KEYWORDS: OpenCL; GPUs integradas; visión por computador; KinectFusion

Resum

La visió per computador (CV) s'espera que sigui la propera "revolució" en el camp de la computació mòbil. Per exemple, Google ha anunciat recentment el seu projecte "Tango", un telèfon de 5 polzades Android que conté maquinari i programari dissenyat per fer el seguiment del moviment en 3 dimensions del dispositiu que, mentre es sosté, crea simultàniament un mapa de l'entorn.

Un dels problemes que queda per resoldre és com fer servir els últims algoritmes de visió per ordinador —dissenyats per executar-se en ordinadors d'escriptori potents amb diverses unitats de processament gràfic (GPUs)— en GPUs integrades de dispositius mòbils, que són energèticament eficients però més lentes.

Aquest projecte investiga la manera d'accelerar nuclis i aplicacions de visió per computador a través de diferents tècniques d'optimització i paral·lelització. Inicialment hem seleccionat una aplicació representativa de CV, KinectFusion, i l'hem portada a una plataforma mòbil utilitzant OpenCL. Després, hem realitzat una avaluació de rendiment, hem identificat els colls d'ampolla i hem optimitzat encara més el rendiment. Finalment, hem estudiat el seu rendiment a una plataforma mòbil diferent per avaluar la portabilitat del rendiment de les nostres optimitzacions.

KEYWORDS: OpenCL; GPUs integrades; visió per computador; KinectFusion

Acknowledgements

First of all, I want to thank my advisor, Björn Franke, for believing in me and giving me the chance to live this experience at the University of Edinburgh. I enjoyed living there, the topic of the work, the work environment and the freedom he gave me. Not only he introduced me to the field of computer vision but also to research. He also gave me support with the project and reviewing the thesis after the end of the internship when I was in Barcelona.

I also want to thank my academic tutor, Xavier Martorell, for his help and advice during all these months. From the distance, when I was in Edinburgh, until the last day in Barcelona.

Filippo Mantovani, for giving me good career and personal advice, for his time and support with my goals and his enthusiasm.

Oriol and Dani, for giving me good advice and tips, and for their friendliness since the very first day. Specially Oriol, for his help and cooperation and for making the office a better place to work.

Thanks to many other people from BSC, such as Nikola Rajovic for his help with power measurements; FIB professors such as David Lopez, who made me enjoy preparing and giving presentations thanks to his motivational classes and tips, and Victor Muntés and Agustín Fernández for making me love computer architecture; and friends from different stages of my life, such as Alba, Laura (ITO), Ainoa, Francesc, Khaoula, Jing, Jane and Mathi.

Special thanks to Alex, for his love, trust and support during all the degree and during this thesis. For his patience while I was working on holidays and several times till the small hours.

And, last but not least, all my family. My parents: Maria Teresa and Paulo, for supporting me with the several trips I did, summer schools, conferences and internships; for their patience; and, for helping me grow this passion for computer science since I was a little girl. My 24 cousins: Ceci, Laura, Maria Emilia, July, Josefina, Tomy, Martin, Guada, Fran, Majo, Juan, Edu, Luciana, Nacho, Santy, Sebi, Juampy, Lucas, Agustín, Betty, Katia, Yaundé, Kayawe, Kitwe, and their wives/husbands and children; for cheering me up from the distance and caring about me. My 16 aunts and uncles: *Mary*, Titi, Lucy, Cecilia, Miguel, Monica, Eduardo, Alicia, Emilio, José Maria, Rosi, Tito, Marcela, Ghi, Julius and Denise; for their love. My grandmothers: Lilia, for always sharing her knowledge and wisdom from her 99 years, and my *vó* Erica. My grandfathers, that live in my memory. Most of them are far away but I remember them every day.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

In the last decade, computer architecture changed the way of designing chips: increasing the number of cores per chip instead of building faster single-core chips. This happened because the technique being used to increase single-core performance was to increase clock frequency by reducing the size of transistors and increasingly pipelining the central processing unit (CPU) logic. This was not sustainable as power density reached a point at which the heat produced by the chip reached the limit that common dissipation equipment could get out from the package. The size of transistors kept decreasing thanks to technology improvements and this led to add more complexity to the core to improve performance (e.g., out-of-order execution) and more cache space. After some time, computer architects were experiencing diminishing returns. At this point, the power density was too high due to the concentration of most of the logic in a small space. The multi-core era started. It was better to integrate more cores even if they were simpler. This helped to better distribute the power and improved performance mainly in multiprogrammed environments running several software applications concurrently. The problem was for compute-intense applications that had to be parallelised to get performance out of all the available resources. In this case, the problem was left to the programmers.

Mobile devices tend to have heterogeneous system-on-a-chip (SoC) with specialised processing elements or accelerators for faster and more energy-efficient computation of certain operations. The current approach to accelerator-based systems [1] is to couple general-purpose CPUs together with compute-capable graphics processing units (GPUs) to accelerate compute-intensive workloads. This approach is the same for desktop computers and mobile devices. Desktop computers have powerful CPUs and GPUs that provide high performance in a few hundred watts of power. Embedded ones, on the other hand, are more energy-efficient but less powerful as they have to fit in a power envelope of a few watts. Power consumption and heat dissipation are major issues when designing mobile devices.

Mobile devices are the new era in computing and the market grows fast [2]. At the same time, their performance has increased by 25× in 3 years [3]. The challenge now is power: the power envelope is going to remain fixed. This is due to the need for cooling, which is limited in compact mobile devices. This thermal limit equally determines the power limit that stays constant. The problem is how much power these devices can handle. This also affects performance: a mobile device working at maximum speed for a long time heats up quickly, which forces to slow down operating frequency.

Smartphones and other mobile devices integrate more sensors in each generation. Some examples are GPS, accelerometer, gyroscope, proximity, light and camera (front and back). These hardware components make mobile devices an appealing option for computer vision (CV). The possibilities of CV in mobile devices are endless [4]: games, improve blind people's life, new learning techniques, easy planning of decoration, and so on.

Demanding state-of-the-art computer vision algorithms require powerful GPUs able to provide high performance. They are still not prepared to run in mobile devices and that is the reason why there is ongoing research in this topic.

1.1 The PAMELA project

This project is part of a larger project called PAMELA [5]. PAMELA is a project which aims to optimise the hardware and software configurations together to address the important application domain of 3D scene understanding. This will enable a future smartphone fitted with a camera to scan a scene and not only to store the picture it sees, but also to understand that the scene includes a house, a tree, and a moving car. In the course of addressing this application, they expect to learn about optimising many-core systems that will have a wider applicability too, and the prospect of making future electronic products more efficient, more capable, and more useful.

1.2 My project

The project consists of porting the C++ KinectFusion (KF) code (more information in Section 2.2), used as a representative of CV applications, to an embedded GPU, the ARM Mali-T604, using the OpenCL programming language [6]. Then, optimise it as much as possible.

We aim to run the KF algorithm at an interactive rate in the embedded GPU, which is a challenge and we did not know if it is possible beforehand. We used the Arndale board, a development board which contains the ARM Mali-T604 GPU. There are already two GPU versions of the KF algorithm (in two different languages, OpenCL and CUDA). However, these existing implementations were translated from other versions: one was aimed at powerful GPUs and the other one is not optimised. The latter was not implemented for performance but to experiment with the code. Therefore, it is not optimised for any specific hardware, even less for this specific board. This is why we decided to write a new version from scratch. This way we can focus on performance from the beginning, while keeping in mind potential optimisations specific to the Arndale Board.

First, we wrote a clean version to have a decent starting point and baseline for comparison. Then, we ported the code to OpenCL and we optimised it for this SoC. After this, we measured the performance of our optimised KF code and compare it to the original version. We also measured and compared power and temperature. Last, we ran the optimised code in a more powerful SoC to validate the functional and performance portability of our optimised version.

The outcome of this project will serve for the PAMELA project (explained in the next section) and the researchers working on it. Apart from the code I developed, they will also benefit from the lessons learnt chapter about the challenges and solutions I faced when working on an experimental development board and the potential performance of this embedded GPU running the KF application, and this thesis in general. My code will be uploaded to the repository so the researchers involved in the PAMELA project will benefit from it and use it as starting point of other research. Also, as it is open source, other people can take a look to it and reuse it. My thesis will be public so the results can be seen by anyone interested.

1.2.1 Objectives

The main objectives of this project are:

- To evaluate the potential for speeding up compute-intensive CV apps on mobile devices
- To investigate how far CV is from running in a mobile device in terms of performance (frames per second), feasibility and reliability

1.2.2 Contributions

The contributions of this project to fulfil the aforementioned objectives are:

- A clean C++ version of KF
- An OpenCL version of KF
- Performance, power and temperature evaluation of both C++ and OpenCL versions in the Samsung Exynos 5 Dual (two ARM Cortex-A15 and an ARM Mali-T604 GPU)
- A list with the lessons learnt from using a development board and OpenCL in this kind of algorithm

All the software will be updated to a repository of the KinectFusion to allow easy access to all the researchers working on it.

1.2.3 Integration of knowledge

In this project we integrated knowledge from several disciplines: GPU programming; architecture-aware programming; computer architecture; microprocessor implementation constraints including: thermal constraints and power; and system administration.

1.3 Document structure

Chapter 2 introduces some background related to this project. Chapter 3 explains the state of the art of computer vision and embedded GPUs. Chapter 4 presents the initial estimation of the project, both planning and cost evaluation, and the deviations. I also discuss the sustainability and the viability of the project. Chapter 5 introduces and explains the code and the optimisations performed, and discusses the challenges that came up. Chapter 6 describes the methodology used in the project. This includes the tools, the evaluation of the final result and the risk management. Finally, we conclude showing and discussing the results, the lessons learnt and the conclusions of this work. At the end, we propose different lines of future work.

Chapter 2

Background

In this chapter I introduce the technologies involved in this project.

2.1 Computer Vision

Computer vision (CV) is a field related to artificial intelligence. It aims to acquire, process, analyse and understand images and high-dimensional data from the real world in order to produce numerical or symbolic information [7].

CV is used, for example, for scene scanning, object recognition and augmented reality (AR). For this purpose, we need either a camera or input files, such as videos or images. To use it in mobile devices we will assume the use of a camera as the input.

In scene recognition, the camera may move around the scene. This can be used for driverless cars, where the camera captures the environment of the car and recognises the objects separately. In object scanning, the camera is fixed and the object is the one that may move in front of it to create the complete 3D model. It can be used, for example, to scan objects that can be printed in a 3D printer: if a piece of an object breaks, you could scan it and print a new one. In AR, the screen shows what is being captured by the camera and overlays an image on top of it providing some virtual objects in the real image. Some examples of AR are: helping you decorate a room by placing the furniture adding it to the real-time image captured by the camera of a mobile device [8]; replacing a building or an area for how it was some years ago; replacing the words that appear in the screen for the translation to other language [9]; and helping brain surgeons visualise 3D brain scans (Microsoft Research Cambridge). Figure 2.1 shows some of those examples.

These are just a few examples of what the possibilities of such CV algorithms are capable of. As the technology and the algorithms improve, the possible usages are endless: "Imagination is the only barrier to the next era of mobile applications" stated T.H. Kim, vice president of System LSI marketing, Device Solutions, Samsung Electronics. Most of them require a mobile device to be useful, to use them in any place at any time.

Figure 2.1: Examples of current computer vision applications. Image 1 is an example of augmented reality: the IKEA app. Image 2 is an example of object scanning: a 3D model of a person created by a Kinect. Image 3 is an example of scene recognition: the Google driverless car. Image 4 is an example of object scanning: face recognition. Image 5 is an example of augmented reality: a prototype to visualise 3D brain scans built by the Microsoft Research Cambridge team. Image 6 is an example of augmented reality: the Word Lens app.

2.2 KinectFusion

KinectFusion (KF) provides 3D object scanning and model creation using a Kinect for Windows sensor [10]. The user can scan a scene with the Kinect camera and simultaneously see and interact with a detailed 3D model of the scene. With a powerful GPU, this can run at interactive rates, but in a smartphone would run so slow that it will probably not provide a good enough user experience.

We chose this algorithm for this project as it is one of the CV algorithms that many researchers in the PAMELA project were working at the moment. Not only I had support but also I was working with their latest work and updates.

2.3 Development boards

A development board is a printed circuit board containing a microprocessor and the minimal support logic needed to use it and to learn programming it. It also serves for prototyping applications for the target microprocessor.

In this project we are interested in development boards with mobile chips. These boards are mainly targeted to Android programmers, so they can build applications before releasing phones with that chip. In our case, we want to use Linux to be able to compile other languages and not only Java.

We chose the development board among the ones available in the department: Odroid-XU, Jetson TK1, Arndale board and Odroid-XU3.

Odroid-XU is the first development board with a big.LITTLE[1] architecture SoC [11]. This board is equipped with Samsung Exynos 5 Octa including a Cortex-A15 1.6GHz quad core (the "big" cluster) and a Cortex-A7 quad core (the "LITTLE" cluster) CPUs. However, the GPU is not from ARM, but from Imagination Technologies: PowerVR SGX544MP3 GPU.

Jetson TK1 features the NVIDIA Tegra K1, the NVIDIA's strongest bet for embedded GPUs [12]. NVIDIA has two GPU architectures among others: Tegra and Kepler. Tegra is the line of mobile GPUs and Kepler, the line of energy-efficient GPUs for high performance. The TK1 aims to be the high-performance GPU for mobile devices. It contains a 4-Plus-1 quad-core ARM Cortex-A15 CPU and a Kepler GPU with 192 CUDA cores.

Arndale board is a development board equipped with the Samsung Exynos 5250, including two ARM Cortex-A15 and an ARM Mali-T604 GPU [13]. This board was the first one to integrate this chip, specifically, this embedded GPU: the ARM Mali-T604.

Odroid-XU3 is the first development board with a big.LITTLE architecture SoC which allows to use all eight cores at the same time in Linux [14]. This board is equipped with Samsung Exynos 5422 including a Cortex-A15 2GHz quad core (the "big" cluster) and a Cortex-A7 quad core (the "LITTLE" cluster) CPUs and a Mali-T628 GPU. This GPU supports OpenCL and it is more powerful than the one in the Arndale board (T604).

We made the selection of the board considering its support, both hardware and software, for OpenCL. The Odroid XU has OpenCL support only in Android. We need to use Linux, therefore we discarded it. The Jetson TK1 has hardware support for OpenCL but there is no software support available yet, so we also discarded it. The Mali-T604 is compute-capable and has OpenCL support. In fact, the Arndale board is the first development board including an embedded GPU with OpenCL support. We decided that this was the best choice for this project. In August, the Odroid-XU3 was released and we decided to use this one to validate functional and performance portability of our optimised version.

2.4 GPU architecture

A GPU is a heterogeneous chip multi-processor highly tuned for graphics. GPUs are mainly composed of several cores, a memory hierarchy, a job manager and a memory management unit (MMU). The idea of GPUs is to compute a single instruction in all the cores, this paradigm is called single instruction multiple threads (SIMT).

Lately, GPUs are used not only for graphics but also for computation, helping to accelerate compute-intensive algorithms. These GPUs are called general purpose graphic processing units

[1]big.LITTLE is an ARM's power-optimisation technology. It consists of combining a powerful CPU with a enegy-efficient one: the powerful one is used when more performance is needed, and the energy-efficient one, otherwise. This technology removes the trade-off of deciding whether we prefer more performance or consume less power, making the most of each one.

(GPGPUs) and they are programmed with different languages than the ones for the CPU (see Section 2.5). Each core (so called shader cores in ARM and streaming multiprocessors in NVIDIA) commonly include also vectorial instructions. The work is divided in threads —also called work-items— which are the basic unit of work in a GPU. The data is divided among the cores, and the threads execute the same operation on the data they have.

Depending on the architecture, GPUs have different memory types. The important ones regarding GPGPUs are the global and local memory. The global memory is the one that can be accesses from other devices to copy from/to it. The local memory is private to the device, and usually provides faster access than global.

Furthermore, there are different memories among devices, host memory and device memory. The host memory is the one from the CPU that sends the kernels to the GPUs or other devices —CPUs can also be a device—. The device memory is private to a device. Data transfer between host and device memories require explicit data copies. When shared memory is available, host and device memory are the same. In this case, we can avoid transferring data from the CPU to the GPU by asking the MMU to map data from the CPU's address space into the GPU's address space without copies. This allows to decrease the overhead of a kernel call. Shared memory is commonly found in embedded GPUs.

2.4.1 Embedded GPUs

Mobile SoCs include embedded GPUs for processing graphics. However, in recent years these GPUs are also programmable for general-purpose programming (GPGPU, see Section 2.5). These embedded GPUs can be used to run compute-intensive codes faster and more efficiently than in general-purpose CPUs.

In Table 2.1, we show a set of desktop and smartphone GPU models and their peak single-precision floating point performance (FLOPS[2]). The performance of mobile SoCs is increasing rapidly [15] as they have a large demanding market, but there is still a long way to go to reach the performance of a desktop GPU. For this reason, if we want to have CV algorithms running properly in our phones in the future, we need two things: optimise the software to run as fast as possible with the available resources and improve the hardware for this purpose.

Usage	GPU	GFLOPS	Frequency	TDP
Desktop/HPC	AMD FirePro S10000	5910	825 MHz	375 W
Desktop/HPC	NVIDIA Kepler K40	4290	745 MHz	235 W
Mobile	Mali-T678	378	625 MHz	2-5 W
Mobile	NVIDIA K1	365	950 MHz	5 W
Mobile	Mali-T628	136	533 MHz	2-5 W
Mobile	Mali-T604	68	533 MHz	2-5 W

Table 2.1: Performance of different types of GPUs in GFLOPS. The values are obtained from diverse web sources as wikipedia, blog posts, presentations of products and estimations.

[2]FLOPS (FLoating-point Operations Per Second) is a measure of computer performance used in fields of scientific calculations that make heavy use of floating-point calculations. GFPLOPS is a billion of FLOPS.

2.4.2 ARM Mali-T604

For this project, we chose an SoC with the ARM Mali-T604 embedded GPU. This description is for the implementation of the Samsung Exynos 5 Dual, the one in the Arndale board. Figure 2.2 shows the architectural details of this GPU. It consists of four shader cores, each of which contains two arithmetic pipes, one texturing pipeline, and one load/store unit. The shader cores share a coherent L2 cache, an MMU, a tiler, and a Job Manager. The Job Manager is the one that dynamically move threads among the shader cores, which are multi-threaded. This GPU complies the IEEE-754-2008 precision requirements for single and double precision floating point. It also has shared memory managed by the MMU that allows to map data from the CPU's address space into the GPU's address space without copies.

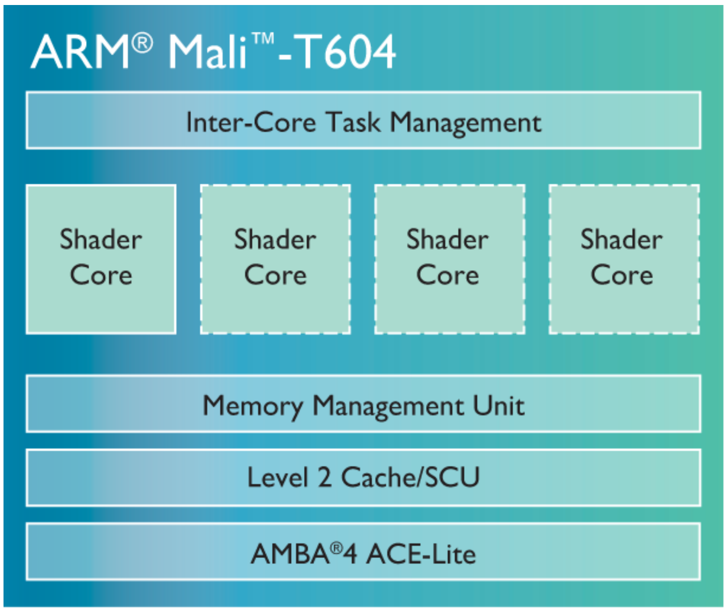


Figure 2.2: ARM Mali-T604 high-level architecture.

2.5 GPGPU Languages

Nowadays, GPU devices are not programmable with general purpose parallel programming models used for CPUs (such as Pthreads, OpenMP and MPI). GPU vendors have introduced a set of programming models for general-purpose (non-graphics) computing on GPUs to exploit their massively parallel architectures on compute-intensive codes. With this scheme, GPUs are seen as accelerators to which the CPU can off-load some compute-intensive work that can be executed faster and more efficiently on the GPU. In this section, I cover the most popular programming models for GPUs: CUDA, OpenCL, OpenACC and C++AMP. I also cover OmpSs, which also supports programming GPUs.

2.5.1 CUDA

Compute Unified Device Architecture (CUDA) [16] is a parallel programming model developed by NVIDIA for its GPUs. It is implemented as extensions to C, C++ and Fortran that provide an API for explicit data transfer between the *host* (CPU) and the *device* (GPU) memories and for off-loading *kernels* (parallel work) to be run on the GPU.

The CUDA kernel is specified inside a function labelled with a specific keyword. The code inside a kernel works on a subset of the total data to be processed. The dimensions of the data and the *blocks* in which it is partitioned to be computed in parallel is specified in the kernel call.

The computation in a kernel is split in thread blocks (each one processing a data block) and each thread block is composed of a set of threads, each one typically processing a chunk of the output data.

A typical CUDA program follows these steps:

1. Allocate and initialize data in host memory.
2. Allocate data in device memory.
3. Copy input data from host to device memory.
4. Launch kernel to run on the GPU.
5. Wait for the kernel to finish.
6. Copy output data from device to host memory.

2.5.2 OpenCL

Open Computing Language (OpenCL) [6] is a programming model for multiple types of processors such as CPUs, GPUs, DSPs and FPGAs. OpenCL is developed and maintained by the consortium Khronos Group, which includes Apple, Intel, Qualcomm, AMD, NVIDIA, Imagination Technologies and ARM, among others.

OpenCL is implemented as a library for C and C++ for the host code, and a specific compiler for the target device. The main idea of OpenCL code is to be portable across different devices. However, although OpenCL codes are actually functionally portable, performance is not portable across different types of devices [17].

The way computation is split and the steps a typical program follows are equivalent to CUDA. A remarkable difference is that kernel code is compiled at run-time and not at compile time. Programmers have to use a specific API call to compile the kernel at run-time. This and other features added for portability, make programming in OpenCL to be, although equivalent, more tedious than programming in CUDA.

We chose to port it to OpenCL because of the portability of OpenCL compared to CUDA, which is tightly related to NVIDIA GPUs.

2.5.3 OpenACC

Open Accelerators (OpenACC) [18] is a programming model by Cray, CAPS, NVIDIA and PGI for accelerators. It is implemented as a set of compiler directives for C, C++ and Fortran. The OpenACC compiler then converts the code to an intermediate language such as CUDA or OpenCL, that is then compiled for the GPU.

Similar to OpenMP, it allows to specify parallel sections, such as parallel loops, and also kernels to be run on the device.

The objective of OpenACC is to simplify device programming with respect to CUDA and OpenCL. One of the major advantages is that data transfers between host and device memory are now implicit, so the programmer does not need to deal with them.

2.5.4 C++AMP

C++ Accelerate Massive Parallelism (C++AMP) [19] is a programming model by Microsoft for accelerators. It is implemented as a C++ library that allows to specify kernels as functions (including lamdas). It also defines a set of data structures for specifying data to be processed by the accelerator.

As in the case of OpenACC, data transfer are implicit, so not exposed to the programmer.

2.5.5 OmpSs

OmpSs [20] is a programming model by the Barcelona Supercomputing Center for multiple types of processors including CPUs, GPUs, and clusters. It is implemented as a set of compiler directives for C, C++ and Fortran. The OmpSs compiler translates the OmpSs code including accelerator directives to intermediate code that includes CUDA or OpenCL.

OmpSs also hides explicit data transfers to the programmer and allows the specification of multiple implementations of the same kernel for different devices, such as an implementation for the CPU and another for the GPU. This way, the OmpSs runtime library automatically makes simultaneous use of both the CPUs and GPUs in the system and schedules the kernel to the most suitable processor type at run-time.

Chapter 3

Related work

CV is a trending topic nowadays. But, actually, back in the early 1900s, there was research on this topic. For example, Prof. Lippmann came up with a new photographic method called *integral photography* [21] which consists of using small glass globules placed in a way to reflect the view of an object from different places to obtain a more complete view of the object. The pictures taken by this camera reflected the view of the object from different perspectives, so looking at them seemed to "reconstruct" the original geometry of the scene. It took a hundred years to achieve enough performance to compute CV algorithms in a mobile device.

3.1 GPGPU on Mobile SoCs

Interest in mobile SoCs is growing in diverse areas since the market keeps growing fast. For example, in 2011 there was already an article showing this interest in low-power processors for high-performance computing [22].

There are several research projects ongoing with the Arndale board to use the embedded GPU. For example, in the Mont-Blanc project, a supercomputer is being built out of the same chip as the one in the Arndale board (Samsung Exynos 5250) [23]. There is research to evaluate if these mobile chips are ready for supercomputing, which is a field where the performance of the CPUs and the GPUs is essential[15]. Moreover, there is research about OpenCL programming for the same GPU [24] where some optimisation techniques are evaluated with benchmarks.

A similar work [25] also evaluates low-power GPUs for non-graphic workloads. It benchmarks the embedded GPU with different benchmarks to find utility in different application domains. They conclude that embedded GPUs are not only promising for their performance, but for their energy efficiency. They also propose techniques to program these mobile SoCs using heterogeneous programming.

3.2 Computer Vision in Mobile Devices

Computer vision for mobile devices is becoming a trending topic since the performance of the chips is high enough to run CV applications. There are specialised companies like Irida labs and Movidius working on software and hardware for CV in mobile devices.

Irida Labs [26] is a company that develops high throughput applications such as video stabilisation with rolling shutter correction, face detection and recognition, low-light image/video enhancement, and car plate detection and recognition, addressing various markets ranging from consumer electronics to mobile appliances and automotive applications. For example, in video stabilisation, the mobile device processes the scene in real time and recognises the objects in order to understand how the camera moves relative to the scene and eliminate the trembling. This is specially difficult when it is dark, there are plenty of similar objects together or the resolution of the camera is low.

On the other hand, Movidius [27] created a processor specialised for computer vision: the Myriad 2. The Myriad 2 architecture comprises a complete set of interfaces, a set of enhanced imaging/vision accelerators, a group of 12 specialized vector VLIW processors called SHAVEs, and an intelligent memory fabric that pulls together the processing resources to enable power efficient processing. It is targeted to mobile devices like smartphones, tablets, wearables and embedded devices.

Google has recently announced their project *Tango*, a 5" Android phone containing highly customized hardware and software designed to track the full 3-dimensional motion of the device as you hold it while simultaneously creating a map of the environment [28]. They also release a 7" tablet including the NVIDIA Tegra K1 processor, 4GB of RAM, 128GB of storage, motion tracking camera, integrated depth sensing and the usual wireless interfaces (WiFi, 4G LTE).

There are also other attempts of CV in mobile devices, like many Android apps. Some of them are: the IKEA app [8] and the Word Lens app [9], but there are many others. I tested some of them myself and I think they are not as fast as necessary to be interactive and you can easily notice the battery discharging.

3.3 Similar projects

A previous study [29] tests a face recognition algorithm in a smartphone GPU. It revealed that a significant speed-up was achieved in performance as well as substantial reduction in total energy consumption, in comparison with the CPU version.

A similar work [30] explains the development of an OpenCL-based heterogeneous implementation of a CV algorithm -image inpainting-based object removal algorithm- on mobile devices. Their experimental results show that heterogeneous computing based on GPGPU co-processing can significantly speed up the computer vision algorithms and makes them practical on real-world mobile devices.

Although there are similar works, to the best of our knowledge, this project is the first one to port and optimise this particular CV algorithm, the KF algorithm, for an embedded GPU.

Chapter 4

Project management

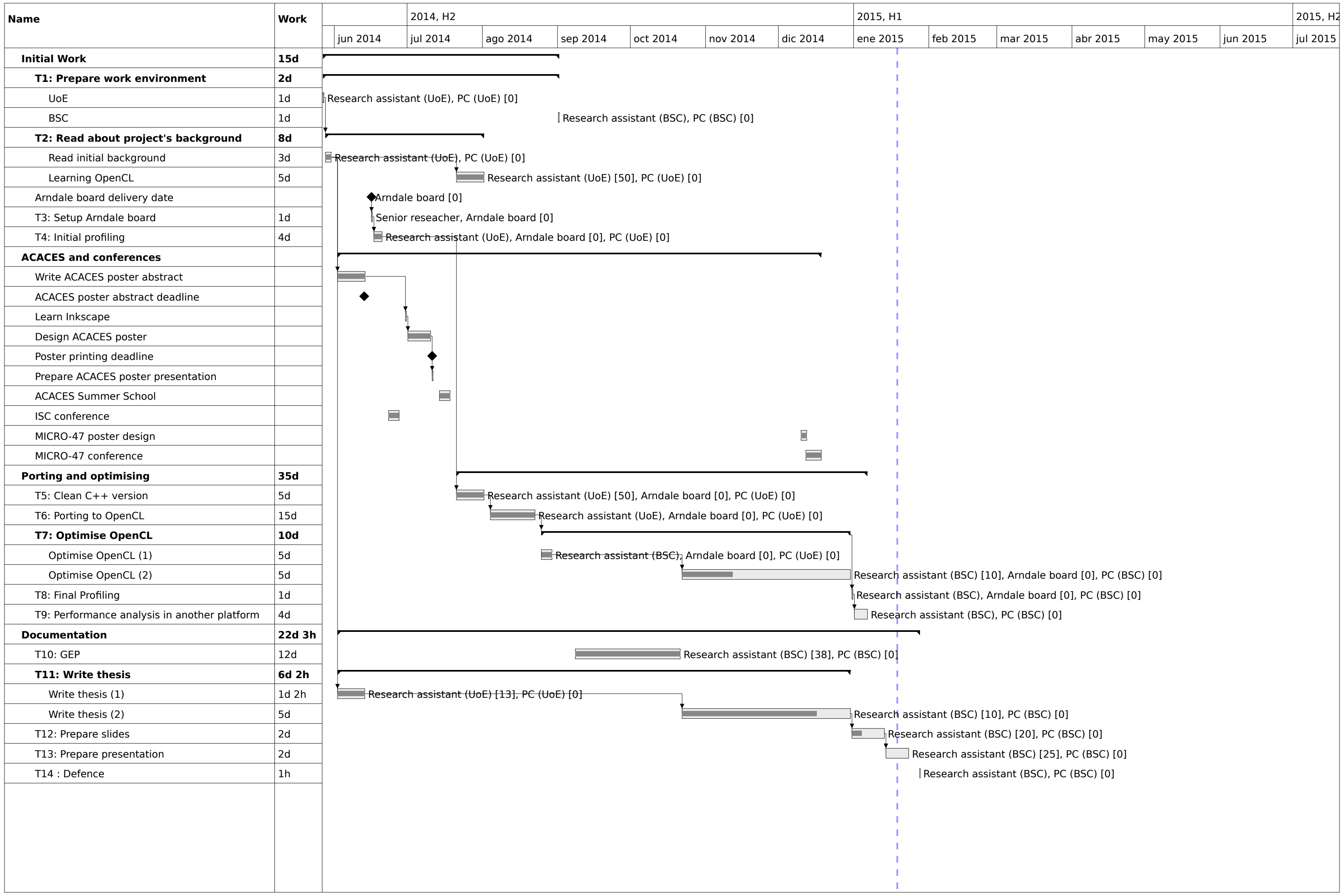
In this chapter we present the plan for this thesis. First, we present an overview of the planning which includes the Gantt chart and the resources needed to carry out the work. We also list all the tasks and a brief description of each of them. Second, we present a cost evaluation and the budget monitoring methodology. Third, we summarise the laws and regulations that apply to this work. Last, we discuss the sustainability and viability of the project.

4.1 Planning

The initial date of the project is May 27th and the final date must be January 26th. This makes for approximately 571 hours of work. I worked from May 27th to August 29th full-time (8 hours a day), and from September 1st to January 26th part-time (3 hours a day). Both periods of time from Monday to Friday. We planned the duration of the tasks considering potential problems that can delay each task. If that is not enough, I can extend the working days to the whole week including the weekends.

This thesis is the result of two internships: the first (and largest) part is at the University of Edinburgh (UoE) and the second part is at the Barcelona Supercomputing Center (BSC).

Figure 4.1 shows the Gantt chart of my plan. For each task, there is information about the resources and the duration. The milestones are also shown in the Gantt chart but not explained in the document as they are just deadlines of some tasks. The weekly meetings with my supervisor are not shown as he told me that his availability depends from week to week and the duration of each meeting is also variable. Nevertheless, I am considering them on the general duration of each task.



Name	Work
Initial Work	**15d**
T1: Prepare work environment	**2d**
UoE	1d
BSC	1d
T2: Read about project's background	**8d**
Read initial background	3d
Learning OpenCL	5d
Arndale board delivery date	
T3: Setup Arndale board	1d
T4: Initial profiling	4d
ACACES and conferences	
Write ACACES poster abstract	
ACACES poster abstract deadline	
Learn Inkscape	
Design ACACES poster	
Poster printing deadline	
Prepare ACACES poster presentation	
ACACES Summer School	
ISC conference	
MICRO-47 poster design	
MICRO-47 conference	
Porting and optimising	**35d**
T5: Clean C++ version	5d
T6: Porting to OpenCL	15d
T7: Optimise OpenCL	**10d**
Optimise OpenCL (1)	5d
Optimise OpenCL (2)	5d
T8: Final Profiling	1d
T9: Performance analysis in another platform	4d
Documentation	**22d 3h**
T10: GEP	12d
T11: Write thesis	**6d 2h**
Write thesis (1)	1d 2h
Write thesis (2)	5d
T12: Prepare slides	2d
T13: Prepare presentation	2d
T14 : Defence	1h

2014, H2
2015, H1
2015, H2
jun 2014
jul 2014
ago 2014
sep 2014
oct 2014
nov 2014
dic 2014
ene 2015
feb 2015
mar 2015
abr 2015
may 2015
jun 2015
jul 2015
Research assistant (UoE), PC (UoE) [0]
Research assistant (BSC), PC (BSC) [0]
Research assistant (UoE), PC (UoE) [0]
Research assistant (UoE) [50], PC (UoE) [0]
Arndale board [0]
Senior reseacher, Arndale board [0]
Research assistant (UoE), Arndale board [0], PC (UoE) [0]
Research assistant (UoE) [50], Arndale board [0], PC (UoE) [0]
Research assistant (UoE), Arndale board [0], PC (UoE) [0]
Research assistant (BSC), Arndale board [0], PC (UoE) [0]
Research assistant (BSC) [10], Arndale board [0], PC (BSC) [0]
Research assistant (BSC), Arndale board [0], PC (BSC) [0]
Research assistant (BSC), PC (BSC) [0]
Research assistant (BSC) [38], PC (BSC) [0]
Research assistant (UoE) [13], PC (UoE) [0]
Research assistant (BSC) [10], PC (BSC) [0]
Research assistant (BSC) [20], PC (BSC) [0]
Research assistant (BSC) [25], PC (BSC) [0]
Research assistant (BSC), PC (BSC) [0]

Figure 4.1: Gantt chart with the time duration and resources for each task.

4.1.1 Plan deviation

The work plan has changed a bit due to unexpected business trips and extra work. We decided that I could attend a research conference (MICRO-47 held in Cambridge, UK). I wanted to take advantage from this experience by presenting a poster in a workshop of the conference as I want to pursue a research career and this was a good opportunity of learning new skills and interacting with the attendees. So, we did not finished the optimisations in OpenCL on time, but we reserved the holidays and vacations from work to recover the schedule. This way I worked more hours a day instead of only three. Also, the initial plan already included a margin so this was not a big issue.

4.1.2 Resources

The resources needed in this project are human and material.

Human: A research assistant (me, divided in UoE and BSC), an advisor (Björn Franke, my supervisor at UoE) and a Senior researcher (Bruno, a postdoc at UoE working on a similar topic).

Material: Arndale board, desktop computer (UoE), laptop (BSC) and another platform similar to Arndale board.

4.1.3 Tasks

The tasks are divided in four general groups: Initial work, ACACES and conferences, porting and optimisation and documentation.

Initial work

Task 1: Prepare work environment This task consists of setting up all my environment in the new office at the University of Edinburgh. This includes: going to human resources and get the key for the office, get the login information to access the computer and email, configure the new email and other services in the computer, sign the contract, apply for the card to access the building. Also, set up the desk environment which includes: regulating the height of the chair, footrest, taking pens and paper and preparing a water bottle. And, of course, presenting myself to the people working in the same office. The resources needed are the research assistant and the PC.

Also, I set up the environment in my new office at BSC where I was doing the second part of my thesis. Similar tasks and resources apply.

Task 2: Read about project's background This task is to read about the background of the main parts of the project to have a first picture of the state of the art. This includes reading about CV, the KF algorithm and OpenCL. Also, to write a poster abstract for a summer school (which I refer in Section 4.1.3), I had to know about these topics to write a proper background. A part from this, I reserve 5 days to exclusively for OpenCL, as I think I would need time to practise and not just reading. The resources needed are the research assistant and the PC.

Task 3: Install software This task consists of installing all the software in the Arndale board needed by the KF application. Bruno, the senior researcher working also in the PAMELA project, helped me wit it. The resources needed are the senior researcher and the Arndale board.

Task 4: Initial profiling I did an initial profiling to evaluate the performance and bottlenecks of the application. I used 4 different inputs which last around 25 minutes each. As I did 5 executions for each input file to have an average time and the standard deviation to mitigate noise. This adds up to approximately 8.5 hours to execute. This task also includes preparing scripts to automatise the performance analysis. Also, to run other profiling tools such as GNU gprof. The resources needed are the research assistant, the Arndale board and the PC.

ACACES and conferences

ACACES stands for Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems. The ACACES Summer School is a one week summer school for computer architects and tool builders working in the field of high performance computer architecture and compilation for computing systems. The school aims at the dissemination of advanced scientific knowledge and the promotion of international contacts among scientists from academia and industry. I decided to present a poster with the preliminary work so I can get feedback from other participants and gain experience in writing articles and presenting posters as I want to pursue a research career.

This part is not counted as part of the project, but as it is related to it and it impacts in the schedule of the project, I include the tasks below.

Write ACACES poster abstract I wrote the four-page poster abstract for the ACACES summer school. It was the first time I write a research dissemination document so I planned more days than I would have thought at first. Also because of my little knowledge on the project's background at the submission deadline, which was shortly after my starting date, and possible problems using LaTeX.

Learn Inkscape We decided to design the ACACES poster with the Inkscape tool. As I had no experience using it, I reserved one day to learn and experiment with it. This was also useful to create figures for this document.

Design ACACES poster I used Inkscape to design the poster and the text from the poster abstract to rewrite it.

Prepare ACACES poster presentation I presented my poster to the other students of the summer school, so I reserve one day to prepare what to say. This was useful for having a first experience presenting this project, before the GEP presentation and the final defence.

ISC conference I went to the International Supercomputing Conference held in Leipzig (Germany) as a student volunteer. This is not entirely part of the project, although it can be very beneficial for my knowledge and for the project indirectly.

MICRO-47 poster design I designed another poster with my current work at BSC. I used all the knowledge and skills learnt from the previous poster.

MICRO-47 conference I attended the 47th International Symposium on Microarchitecture held in Cambridge (United Kingdom). I also did a 3-minute presentation of my poster in a workshop of the conference. This is not part of the project but it was also beneficial to improve several skills.

Porting and optimising

Task 5: Clean C++ version This task consists of cleaning the current version of the C++ KF. This is because the existing version is done by translating a CUDA version previously translated from OpenMP. Also, we expect to clean it in a way that can be easier to port to OpenCL and potentially even optimise the plain C++ version as well. The resources needed were the research assistant, the Arndale board and the PC.

Task 6: Porting to OpenCL This task consists of porting the clean C++ version to a simple version in OpenCL. I approximated 15 days to do the porting as I had no previous experience with OpenCL and I was learning along (see Section 6.3). The resources needed were the research assistant, the Arndale board and the PC.

Task 7: Optimise OpenCL I optimised the simple OpenCL version. This task is iterative, I applied each optimisation separately and this includes doing a profiling to check that the optimisation it is actually faster than the previous one. If it is worse I have to rollback to the previous version. I divided this task in two as I did one part at UoE and the second part at BSC. The resources needed were the research assistant, the Arndale board and the PC.

Task 8: Final profiling This task consists of doing the final profiling of the applications to compare it with the original version and discuss the results. The resources needed were the research assistant, the Arndale board and the PC.

Task 9: Performance analysis in Odroid-XU3 We installed all the software and my OpenCL optimised version in another platform to evaluate performance portability. The resources needed were the research assistant, the Odroid-XU3 and the PC.

Documentation

Task 10: GEP This task includes 7 deliverables about the management of this project and a presentation. The resources needed are the research assistant and the PC.

Task 11: Write thesis This task consists on writing a document from the background and motivation to the evaluation and results of the project. Part of this task is done in the GEP task, where part of the deliverables can be adapted to fit in the thesis. Also, I reused text from the ACACES poster abstract, so I started this task at the same time as that task. The resources needed are the research assistant and the PC.

Task 12: Prepare slides This task consists of designing the slides for the defence and a possible demonstration of an execution of my optimisation with the board. The resources needed are the research assistant and the PC.

Task 13: Prepare presentation This task includes to modify and adjust the slides done in the previous task to fit better on the available time for the defence and to prepare the speech. The resources needed are the research assistant and the PC.

Task 14: Defence The defence will last 30 minutes plus another 30 minutes to set up the material needed for the presentation and make sure everything works fine. The resources needed are the research assistant and the PC.

Task dependences

We briefly mention the tasks dependences. We refer them with the task number. The symbol "<" separates that order. The symbol ";" denotes different dependency chains. The critical path is the path that needs more hours which is the one of the porting and optimising the code.

T1<T2<T3<T5<T6<T7<T9;
T10<T11<T12<T13<T14;
T4,T8<T11;

4.2 Cost evaluation

In this section we present the budget for this project and study its sustainability and viability. I divided the expenses in four groups: human resources, hardware, software and other expenses. We chose to evaluate this in pounds (£) as the project was done in the United Kingdom. Table 4.1 shows the costs of each part and the total.

Category	Cost (£)
Human resources	5 821.63
Hardware resources	68.33
Software resources	0.00
Other	36.67
Total	**£5 926.63**

Table 4.1: Total costs.

4.2.1 Human resources

Table 4.2 shows the costs of the human resources needed in this project. Figure 4.1 also shows the human resources per task. However, there are other not related to a specific task that are considered apart as explained below.

The cost per hour is not an estimation but the real salary calculated from the annual salary at the University of Edinburgh (UoE) and at the Barcelona Supercomputing Center (BSC). The salary of the research assistant is considered separately (UoE and BSC) as the salary is different in the different companies.

The time estimation for the advisor comes from: 10 meetings of an hour, and 10 hours for revising text (Task 11) and answering questions (unplanned). The time estimation for the senior researcher comes from: 1 hour of installing and setting up the Arndale board (Task 3) and 9 hours of various support. The rest of the hours are for the research assistant as specified in the previous section (Section 4.1) as a sum of all the other tasks (including task 10).

Human resources	Cost (£/h)	Hours	Cost (£)
Advisor	29.15	20	583.00
Senior researcher	18.71	10	187.10
Research assistant UoE	12.38	314	3 887.32
Research assistant BSC	4.53	257	1 164.21
Total		**601**	**5 821.63**

Table 4.2: Costs of human resources.

4.2.2 Hardware

Table 4.3 shows the actual expenses of each hardware component needed in this project. Everything was bought just before the start of the project except the desktop computer (Intel Core Duo 2.5GHz, 2GB RAM), the laptop (Dell Latitude E7440) and the other platform, which are estimations. The other platform may be either an Odroid XU3 (£109.90) or a Nvidia Jetson K1 (£117.78), if we get OpenCL software support. As the other platform was not decided at the beginning of the project, we added a margin in case we decide to use a more expensive one. We chose the Odroid-XU3 which costs £120, as it is under the initial budget there was no problem.

The mouse, the keyboard and the screen are estimated for both locations, as they have similar characteristics and price. Other common office-related objects, like a chair, a footrest, a desk, pens, are not considered as they are not bought specifically for this project and other people will use them afterwards.

In the plan of Section 4.1, the Arndale board and the MicroSD card are used 26 days (Tasks 3-8), but as it was non-stop during those days to be accessible remotely, I estimate a total of 624 hours on (24 hours a day), instead of 208 hours (8 hours a day). The depreciation (last column of Table 4.3) is calculated with Equation 4.1. The other number of hours are extracted from the planning of the previous section (Section 4.1). I consider the lifetime of the hardware as 3 years, as usually it is replaced every 3 years and the development boards probably do not last longer or become useless.

We added in Table 4.3 the cost of the heat sink and the fan which were not considered in the initial plan.

$$depreciation(\pounds) = \frac{cost(\pounds)}{3\,years} \cdot \frac{1\,year}{365.25\,days} \cdot \frac{1\,day}{\#\,hours} \cdot \#hours \tag{4.1}$$

Hardware resources	Power (W)	Cost (£)	Time (h)	Total cost (£)
Arndale board	15	146.74	624	3.48
Heat sink		2.59	440	0.06
Fan	2	3.43	430	0.08
MicroSD card 8GB		4.49	624	0.11
Desktop computer UoE	400	500.00	314	17.91
Laptop BSC	90	1 100.00	257	32.25
Another platform	15	118.43	32	0.14
Keyboard		10	571	0.65
Mouse		8	571	0.52
Screen	20	200	571	13.03
Total				**£68.33**

Table 4.3: Costs of hardware resources.

4.2.3 Software

The software used in this project is free. So there is no cost related to software.

4.2.4 General expenses

This section includes other expenses not considered in the previous sections. It contains indirect costs (electricity) and other expenses related to the internship.

The cost of power consumption is calculated, as rated by *ScottishPower* (£0.12kWh), using the Equation 4.2. Equation 4.3 shows the power cost calculation for the Arndale board as an example. I used the same for the whole project as in Spain is quite similar (£0.11kWh), to simplify calculations.

$$Power\ consumption(\pounds) = cost(\frac{\pounds}{kWh}) \cdot power(kW) \cdot time(h) \tag{4.2}$$

$$Power\ consumption(\pounds) = \frac{\pounds 0.12}{kWh} \cdot 0.015\,kW \cdot 624\,h = \pounds 1.12 \tag{4.3}$$

I do not take into account the expenses of using Internet as I can use Eduroam in both locations. In any case, it might not work or may be slow, so we considered this as unforeseen cost. Also, we used free software and free repositories but we first thought that we might have to pay some service for this, so it was considered in the unforeseen costs.

Concept	Cost (£)
Power consumption	36.67
Total	**£36.67**

Table 4.4: Costs of other resources.

4.3 Budget monitoring

Table 4.5 shows the cost per task, done by considering the human and material cost of each task, but not the power consumption. I use this table to compare, at the end of each task, the possible budget deviation from this initial estimation. The cost per task does not include other costs and contingency as they are equally shared among all.

Task	Human cost (£)	Material cost (£)	Total (£)
Task 1	226.28	1.86	228.14
Task 2	883.32	5.24	888.56
Task 3	239.68	0.05	240.73
Task 4	487.16	2.81	489.97
Task 5	586.20	3.51	589.71
Task 6	1576.60	10.52	1587.12
Task 7	544.40	9.75	463.15
Task 8	36.24	1.25	37.49
Task 9	235.96	5.05	241.01
Task 10	525.89	14.43	540.32
Task 11	396.03	6.83	402.86
Task 12	72.48	2.41	74.89
Task 13	72.48	2.41	74.89
Task 14	4.53	0.15	4.68
Total	£5 821.63	£68.33	**£5 889.33**

Table 4.5: Costs per task considering human and material resources.

4.4 Cost deviation

The final costs of this project, shown in Table 4.1, were lower than expected in the initial plan, shown in Table 4.6. The additional board used to evaluate performance portability costed less than we estimated, £118.43 instead of £200. However, this represents only £0.10 less when calculating the depreciation.

The only unexpected costs were that we needed to buy a heat sink and a fan, which were very cheap. Also, the fan broke in the middle of the project, but after that, we used one that was available in the department. The Yokogawa power meter was not bought for this project, so the only impact on its use was on power.

The cost of the power grew by the use of the fan and by having the Yokogawa power meter turned on many hours. This is shown in Table 4.7, as we estimated at the beginning of the project.

All these deviations were covered by the over estimation of the additional board and a little part of the unforeseen costs. The final cost of the project is lower than the initial estimation: £5 926.63 instead of £6 414.31.

Category	Cost (£)
Human resources	5 821.63
Hardware resources	68.19
Software resources	0.00
Other	219.05
Contingency	305.44
Total	**£6 414.31**

Table 4.6: Total estimation costs of the initial plan.

Concept	Cost (£)
Power consumption	19.05
Unforeseen costs	200.00
Total	**£219.05**

Table 4.7: Costs of other resources estimated in the initial plan.

4.5 Laws and regulations

We identified 3 laws and regulations that could apply to this project: the law of intellectual property, as we are modifying existing code; a non-disclosure agreement (NDA) with ARM, as we are using their OpenCL drivers; and the regulations related to recycling, as we are using different hardware in this project.

In the case of the law of intellectual property, all the source code is open source as well as the benchmarks. I have the rights to use and modify the code, so in that sense this project complies the law.

The OpenCL drivers are under an NDA, this means that the results have to be reported to ARM before publishing them (if they agree).

The regulations of recycling electronic components does not apply to this project. We will return the board to the University of Edinburgh and they will continue using it, after that it is outside the project. The same happens with all the other hardware used, for example the laptop, everything will be still in use after this project.

4.6 Sustainability and Viability

In this section we cover the project's sustainability and viability. Sustainability is discussed from the economic, social and environmental point of views.

4.6.1 Sustainability

Economic The estimated cost of the project is £6 414.31. For monitoring the budget of the project we decided to update it with any deviation of the estimation. I checked the possible deviations at the end of each task as well as at the end of each internship. The deviations were minimum and lower than the estimations so there was no inconvenient in that sense.

The hardware platform and the time to carry out the project are fixed so it is not possible to do it cheaper given these requirements. However, given this fixed cost, we target to maximise the output of the project by completing multiple optimisations and thorough evaluations.

A large part of the coding (port and optimise) could be reused from the existing OpenCL version. However, the project requires a new original and better version without the techniques used in the previous version. Therefore, having a lower development cost by reusing previous code is not an option either.

Although it is not considered in the costs, this project could be cheaper by removing the attendance to the ACACES summer school, but it is a research project that requires original ideas and expertise in OpenCL. For this purpose, the summer school is very beneficial as I have no experience in OpenCL and it is usually beneficial to interact with other people, explain them your challenges, getting feedback and disseminate this work. We consider it essential.

Social One of the goals of the project is to improve mobile devices' hardware to run CV applications. The possibility of running CV applications on an affordable device can help multiple social groups such as blind people, people with disabilities and the elder people. As an example, a blind person can use his mobile phone to create a model of his home by recognising the objects and then help this person to move around and reach the objects. Another example is the IKEA app, which overlays their furniture in the image through augmented reality technology, helping people to decorate their home easily and faster. There are endless applications that can be developed to improve the life quality of the society in diverse ways. We need to understand CV applications and to improve hardware and software, and make CV applications on mobile devices a reality. My project is one step to that goal. Also, I will try to publish this work, so further research can be done based on my results.

This technology can also improve scene recognition, useful for driverless cars, which will reduce the accidents up to a 90% [31] and lower the car insurances [32].

Environmental After this project, all the hardware bought and used will be reused by other people. As it is part of a research project, the board is still useful to do further testing. The other hardware resources used to develop the project (computer, keyboard, mouse, screen) will be used by the next worker allocated in my desk. Furthermore, at the end of the hardware's lifespan, the components will be brought to a recycling center so the harmful components will be treated appropriately.

The main component used in this project is the GPU. Mainly, I use it to get more performance but, it is also more energy efficient, so it will consume less power to execute the algorithm in OpenCL in the GPU than in the CPU. Power will be measured and compared between the original version and my optimised version. We expect better energy efficiency, so future platforms featuring our design would be greener.

Regarding the brands of the hardware resources used, the more expensive one is the Dell laptop. Dell ranks in the 5th position of the *Greenpeace Guide to Greener Electronics*. But, Dell still has not removed polyvinyl chloride plastic (PVC) and brominated frame retardants (BFRs), and has no phase-out date for hazardous substances. This substances are toxic, highly resistant to degradation in the environment and able to bioaccumulate (build up in animals and humans)[33].

Most of the power consumption will be at UoE, using energy from ScottishPower. This company has excellent green credentials: besides having a large number of windfarms and a pipeline including 10,000 MW of offshore wind, they also created a support community benefit funds, empowering communities to control how this money is spent to best serve the needs of the local area. To date they have given more than £8.5 million to communities across the UK.

4.6.2 Viability

This project is part of a larger project called PAMELA. Within this project, they had a budget for an internship to do the task of porting and optimising a CV algorithm to a specific GPU and to evaluate performance and power. After evaluating the costs, they approved the project because it is cheaper than their expected budget for the internship. Apart from my estimation of the costs, they are also keeping a margin in case of deviation.

It is a research project, so there is no need to evaluate the potential gains from the product as we do not plan commercialisation of the final product and my results will be open to the public.

Chapter 5

Implementation challenges and solutions

In this chapter, we first explain the organisation of the code. Then, we describe the steps followed from the original version to the optimised OpenCL one. In the last section, we summarise the main challenges found during this part of the project.

5.1 Original version

Listing 5.1 shows a simplified file tree of the KF working directory to help the explanation of the code modifications we did. The common code for all versions is the one in the *include* folder and the main file. The *kernels.cpp* inside the *cpp* folder is exclusive for the C++ version. The OpenCL version would be in another folder under *src/*.

```
kfusion/
  include/             < Folder with the common files >
    kernels.h          < File that includes the data structures and common functions >
    vector_types.h     < File that includes the definition of the vector types >
  src/                 < Folder with the source code of all versions of the code >
    main.cpp           < File that contains the main >
    cpp/               < Folder that contain<s the files of the C++ version >
      kernels.cpp      < File that contains the kernel functions of the application >
```

Listing 5.1: Simplified file tree of the original KF working directory.

The main function implements a while loop that processes all the frames, one per iteration. There are six steps in each iteration: the *acquisition* of the frame, *preprocessing*, *tracking*, *integration*, *rendering* and *drawing*. Listing 5.2 shows a simplified version of the main function. The most important header files are *kernels.h* and *vector_types.h*. The former includes the definition and implementation of the data structures and short inline functions (mainly related to the data structures) common to the kernels. The latter defines the vector data types and their functions.

```
int main (int argc, char ** argv) {  [...]
  while (reader->readNextDepthFrame(inputDepth)) {               // Acquisition
        float4 k = reader->getK() / 2;
        kfusion.preprocessing(inputDepth, inputSize, k, frame); // Preprocessing
        kfusion.tracking(k, frame);                             // Tracking
        kfusion.integration(k, frame);                          // Integration
        kfusion.renderDepth( depthRender, computationSize);     // Render
        kfusion.renderTrack( trackRender, computationSize);     // Render
        kfusion.renderVolumeByPass( volumeRender,               // Render
                   computationSize, reader->getK() /2);
        drawthem(depthRender, trackRender,                      // Drawing
                    volumeRender ,computationSize);
        frame++;
  } [...]
}
```

Listing 5.2: Simplified main code

Then, the file *kernels.cpp* is a different one for each implementation. The functions called from the main file are implemented in this file.

5.2 Starting point

I have started getting a quick idea of the application by printing the time spent in each of the six stages of a frame's processing. Listing 5.3 shows the percentage of the time spent in each stage. Clearly, the Integration stage was the larger one.

```
Acquisition:     2.70 %
Preprocessing: 23.08 %
Tracking:        5.50 %
Integration:   49.87 %
Rendering:     18.64 %
Drawing:         0.14 %
```

Listing 5.3: Percentage of time spent in each stage of a frame's processing.

Then, I did a profiling with *gprof* in the whole application to find the specific functions where the application was spending more time. I found the functions *integrate* and *raycast* to be the kernels that more time was spent in, in that order. Both of them are called in the Integration stage shown above.

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 47.47    647.12   647.12     1260   513.59   513.59  integrate()
 19.49    912.85   265.73     1260   210.90   222.48  raycast()
 18.95   1171.23   258.38     1260   205.06   216.65  cppRenderVolume()
  4.89   1237.86    66.63                             Kfusion::tracking
   ()
  3.64   1287.43    49.57    18981     2.61     2.61  reduce()
  2.45   1320.80    33.37     1260    26.48    26.48  bilateral_filter()
                             [...]
```

Listing 5.4: Profiling with gprof of the C++ original version.

Amdahl's law is used to find the maximum expected improvement to an overall system when only part of the system is improved. It is also used in parallel computing to predict the theoretical maximum speed up using multiple processors [34]. We applied Amdahl's Law with Equation 5.1, to calculate the maximum theoretical speed up we can achieve optimising only the *integrate* function.

$$S(B) = \frac{1}{(1-B) + \frac{B}{M}} \tag{5.1}$$

Where S is the speed up achieved, B the portion of the application which is optimized and M the speed up achieved in B portion. Using the percentage of the execution time spent in the Integration stage, which is a 49.87%, the maximum speed up is the one shown in Equation 5.2. We assume maximum speed up in the B portion to compute the maximum reachable speed up by improving the Integration stage.

$$S(0.4747) = \frac{1}{(1-0.4987) + \frac{0.4987}{\infty}} = \frac{1}{(1-0.4987)} = 1.99 \tag{5.2}$$

5.3 Refactoring

To create a faster OpenCL version, I first had to clean the C++ version through several steps:

- Const qualifiers.
  First, I added some missing "const" qualifiers, as suggested by the C++ coding guide from Scott Meyers [35]. Not only for helping the compiler and getting more performance, but also for readability. When I started, I had to learn about the whole application. Having the "const" qualifiers was very helpful to quickly identify which parameters were inputs and which were outputs. After understanding the code, I was able to add the missing qualifiers.

- Parameters by reference.
  I changed some functions to pass their parameters by reference instead of value, to reduce the overhead of copying data structures.

- Porting common code to C99 to avoid code duplication.
  OpenCL does not support C++ code, only C99. A common file that includes data structures and functions needed by the kernels was implemented in C++. I created a separated common file (from the original *kernels.h*) named *types.h* to incorporate in it all the data structures and functions which were going to be used by OpenCL. In this first version, I just added the ones needed by the first kernel I was going to port, and the following ones I decided to move them on demand for each kernel.

  Another way of fixing this issue would be to put them directly in the kernels file (*kernels.cl*) and not include any file in it. It would have been much easier as the C++ code do not need to be modified, but we decided to spend more time to have a version without duplicated code. As those data structures were used from the kernels, I split them in a structure with the attributes and I put the functions apart, as C99 does not support the previous C++ version and for cleanliness. Listings 5.5 and 5.6 show this with the example of the Volume data structure.

- Reduce useless function calls.
  We also realised, after the profiling, that many of the function calls were done to create vector types, for example *make_int3*. Many of which were not necessary. I reduced the ones I found that were avoidable. For example, many were creating a pair of values for the position of a pixel (even inside long loops) from two other existing variables and then accessing the pair to read the values. None of the variables were modified during the period of the readings so there was no point to create an extra structure duplicating the values.

```
struct Volume {
  uint3 size;
  float3 dim;
  short2 * data;

  Volume() {
      size = make_uint3(0);
      dim = make_float3(1);
      data = NULL;
  }
  float2 operator[](const uint3 & pos)
      const { [...] }
  [ ...functions... ]
  void release(){ [...] }
};
```

Listing 5.5: Original C++ volume data structure implementation.

```
typedef struct {
    uint3 size;
    float3 dim;
    short2 *data;
} Volume;
[ ...functions... ]
inline void releaseVolume(Volume* v)
  { [...] }
```

Listing 5.6: C99 volume data structure implementation.

5.4 Simple OpenCL version

The first step towards the optimised OpenCL version of KF is to create an initial simple functional version. We decided to first port the *integrate* kernel because it was the function that took more execution time. I did this porting aware that was not efficient, but as it was going to be the first one, I wanted to do it as simple as possible.

1. Create a singleton class to include all initialisation and cleaning of OpenCL: OpenCLState

   - Initialisation

     (a) Get the platform IDs and select one (clGetPlatformIDs)

     (b) Connect to a compute device (clGetDeviceIDs)

     (c) Create a compute context (clCreateContext)

     (d) Create a command queue (clCreateCommandQueue)

     (e) Open the kernel file and pass it to OpenCL (clCreateProgramWithSource)

     (f) Build the executable program (clBuildProgram)

    (g) Create kernel objects (clCreateKernel)

- Cleaning
    (a) Release kernel objects (clReleaseKernel)
    (b) Release program (clReleaseProgram)
    (c) Release command queue (clReleaseCommandQueue)
    (d) Release context (clReleaseContext)

2. Create kernel in OpenCL: kernels.cl

3. Create a wrapper function that includes all the necessary to execute that kernel

    (a) Create memory objects (cl_mem)
    (b) Create buffers (clCreateBuffer)
    (c) Prepare arguments (clSetKernelArg)
    (d) Call the kernel (clEnqueueNDRangeKernel)
    (e) Wait for the kernel execution to finish (clFinish)
    (f) Get the results and mapping them to the CPU (clEnqueueMapBuffer)
    (g) Release memory created by the buffers (clReleaseMemObject)

4. Change the original call to the kernel to the one of the wrapper, with the same parameters

The working directory changed as shown in Listing 5.7. The OpenCL code is on a new folder called "opencl".

```
kfusion/
  include/              < Folder with the common files >
    kernels.h           < File that includes the data structures and common functions >
    vector_types.h      < File that includes the definition of the vector types >
  src/                  < Folder with the source code of all versions of the code >
    main.cpp            < File that contains the main >
    cpp/                < Folder that contains the files of the C++ version >
      kernels.cpp       < File with the kernel functions of the application >
    opencl/             < Folder that contains the files of the OpenCL version >
      kernels.cpp       < File with the C++ kernel functions and OpenCL wrappers >
      kernels.cl        < File with the OpenCL kernel functions of the application >
      openclstate.hpp       < File with the declaration of the OpenCL class >
      openclstate-impl.hpp < File with the implementation of the in-line methods >
      openclstate.cpp       < File with the larger methods of the class >
```

Listing 5.7: Simplified file tree of the final working directory.

5.5 Optimisations

Afterwards, the optimisations I applied to the resulting code were the following ones. Some of them did not had a speed up or even had a slow down. We explain them below.

- Optimisation 1: architecture-dependant flags (ARM-flags).
  The code was being compiled only with the "-O3" flag. I added some other flags that may produce more optimised code for this platform. I used the following: -mcpu=cortex-a15, -mtune=cortex-a15, -mfloat-abi=hard and -mfpu=neon-vfpv4.

  The "-O3" flag specifies the level of optimisation to be done in the code by the compiler. Flags "-mcpu" and "-mtune" specify the name of the target processor to derive the name of the target architecture and the processor type for which to tune for performance. The flag "-mfloat-abi" specifies which floating-point application binary interface (ABI) [36] to use, the "hard" value allows generation of floating-point instructions and uses FPU-specific calling conventions. And, last, the flag "-mfpu" specifies what floating point hardware is available on the target.

- Optimisation 2: Use shared memory (OCL-SharedMem).
  The Samsung Exynos 5 Dual has a unified memory shared between the CPU and the GPU. The method done in the original version was to create memory objects before calling the kernel. This means to allocate a buffer with the same size (so it will use double the memory than necessary) and copying the data before and after the call. In the case of shared memory, which is not common in powerful chips, we can create the buffers once with the OpenCL call (clCreateBuffer), instead of *malloc*, with a parameter (CL_MEM_ALLOC_HOST_PTR) that allows the memory to be created and accessed by CPUs and GPUs. In OpenCL, if you create a buffer you cannot access it from the CPU, it is necessary to map it to the CPU before using it (clEnqueueMapBuffer). Also, before calling a kernel it is also necessary to map it to the GPU again (clEnqueueUnmapMemObject). I created the buffer and immediately mapped it to the CPU not to interfere with other uses of that buffer from different parts of the code. In the OpenCL wrapper I map it to the GPU before calling the kernel and map back to the CPU afterwards.

- Optimisation 3: OpenCL flags (OCL-CLFlags).
  I also added flags for the compilation of the OpenCL kernels. Some optimisations make the floating point operation in the GPU not to be compliant with the IEEE754 standard. Since floating point arithmetic precision is not paramount in visual computing, we added flags that will potentially improve performance at the expense of floating point precision. They were the following: -cl-fast-relaxed-math, -cl-mad-enable, -cl-no-signed-zeros, -cl-denorms-are-zero and -cl-single-precision-constant.

  The "-cl-fast-relaxed-math" flag allows optimisations for floating-point arithmetic but they may violate the IEEE 754 standard and the OpenCL numerical compliance requirements. The "-cl-mad-enable" flag allows multiply and add operations with reduced accuracy. The "-cl-no-signed-zeros" flag allows optimisations for floating-point arithmetic that ignore the signedness of zero. The "-cl-denorms-are-zero" flag controls how single and double precision denormalised numbers are handled. The "-cl-single-precision-constant" flag allows to treat double-precision floating point constants as single precision constants.

- Optimisation 4: Port more kernels, *raycast* (OCL-Raycast).
  I ported the *raycast* kernel including all the previous optimisations from the beginning.

- Optimisation 5: Port more kernels, *track* (OCL-Track).
  I ported the *track* kernel including all the previous optimisations from the beginning.

- Optimisation 6: OpenMP (OCL-OpenMP).
  I ported the application to OpenMP. I added a "pragma omp parallel for" in the main loops (most of which were two or three nested loops). This optimises the part running in the CPU to take advantage of the second core which was not being used.

- Optimisation 7: Barriers and Maps (OCL-Barriers).
  I removed a barrier and the mapping of some data structures from/to the GPU from the OpenCL implementation that were not necessary. I did this between the call of *integrate* and *raycast*. Those two functions are called consecutively, so all the data mapped to the GPU that *integrate* uses, does not need to be mapped back to the CPU after *integrate* and mapped again to the GPU before *raycast*.

  Also, the call "clFinish()" blocks until all previously queued OpenCL commands are issued. We are using a single command queue for all the commands, and those are run in order, so we do not need to wait them to finish before enqueuing the new kernel. Moreover, as we are using shared memory, there is no need to wait for the arrays to be copied to the CPU, as there is no copy.

- Optimisation 8: Port more kernels, *bilateral_filter* (OCL-Bilateral).
  I ported the bilateral_filter kernel, which is a large part of the Preprocessing stage, also including all previous optimisations.

5.6 Porting more kernels

The most challenging part was to port the first kernel due to all the OpenCL code preparation. After porting one, porting more is relatively easy following these steps:

1. Create kernel in the kernels' file (kernels.cl)
2. Create the wrapper for that kernel
3. Replace malloc calls of the data structures related to the kernel by OpenCL buffers
4. Replace original call to the new wrapper
5. Call OpenCL to create the kernel (clCreateKernel in OpenCLState and increase the size of the array of kernels)

5.7 Challenges

Porting those kernels implied challenges beyond just learning OpenCL. Because of the complex original code, I faced the following difficulties:

5.7.1 Vector data types

The vector data types in the C++ code were a different size from the OpenCL code. Many data structures are of the size of three elements, for example ”float3”. In OpenCL, size-3 data structures are actually implemented as structures with four elements due to memory alignment issues. Moreover, you cannot pass a three-element structure as an argument of a kernel. Finding out this issue was difficult. Some structures had arrays of float3 elements. The host allocated these arrays with three-element data, and then the kernel accessed these data assuming that it had four elements each. This led to undefined behaviour. The compiler did not complain (because the kernel argument was the structure including the array, and not the array directly) and the application finished with wrong results.

5.7.2 OpenCL debugging

Debugging in OpenCL is tedious. Not only for running in a GPU (impossible to print values as with *printf*) but also for the large size of vectors and matrices used. This made the previous issue even harder to fix. I came up with my own techniques for debugging, mainly two. The first one is to have a general idea of where in the code something is making the program to produce a wrong output. This is done by modifying or removing some lines of the kernel file. For example, removing the code inside an *if* statement. This method was very comfortable as you do not even need to compile the program, the kernel file is compile in execution time (see Section 2.5.2 for more details). The second one is to create another parameter in the kernel to return the values we want to read and print them to a file from the host code.

There are proprietary tools for performance evaluation and debugging from different companies including ARM, Intel and AMD. But we did not have access to them.

5.7.3 OpenCL is C99

When porting the kernel, it is necessary to change any C++ sentence to C99. But this also means that the header files included by the OpenCL kernel code also must be C99, not C++. As previously explained in Section 5.3, I created a separated file for the code needed by both devices.

5.7.4 Original data structures

In the original version, some data structures had functions inside, treated like C++ classes. I had to modify them as well as all the calls of all those data structures and functions.

5.7.5 Function names in C++ and OpenCL libraries

Some of the common math functions (e.g. max()) were named differently in the C++ host library and the OpenCL GPU library. I replaced them by the equivalent built-in functions in the kernel side, and added *define* compiler directives to use either the host or OpenCL version of the function for code that was executed by both sides.

Chapter 6

Methodology

In this chapter we explain the methodology used to evaluate the outcome of the project: the tools and the evaluation methods. We also include, at the end, a risk management section.

6.1 Tools

I briefly describe below both the hardware and software tools I have used in this project.

6.1.1 Hardware

I used hardware for three different purposes: to develop and optimise the code; to run the code; and to measure the power consumption.

The hardware to develop the code itself were a desktop computer at UoE and a laptop at BSC. To run the code I used the Arndale board (described in section 2.3) and another similar platform, the Odroid-XU3. To measure the power consumption I used the Yokogawa power meter.

6.1.2 Software

I used software for five different purposes: to use the computer, to develop the code, to profile the application, to do the documentation, and to manage the project.

To use the computer, I used different Linux distributions. In the Arndale board I used Ubuntu 12.04; in the desktop computer at UoE, Scientific Linux; and in the laptop at BSC, Ubuntu 14.10. To develop the code, I used *Vim* and *GCC*. To profile the application, I used *GNU gprof*. To do the documentation, I used *Texmaker* for writing and *Inkscape* for creating figures. To manage the project I used *Planner* as Gantt chart creator, *git* as a version control software and *Redmine* as a general project management tool to share with my director and academic tutor.

6.1.3 Benchmarks

I used four video input files provided by the Imperial College in format ".raw" as benchmarks of the KF application. The four benchmark are: *chairs*, *desktop*, *person* and *weird*, named by the objects in the video. These videos were recorded by moving a camera around them to get the most angles possible from them. This is because it needs to get the depth from the camera to the whole object to create the 3D model. They take between 15 and 45 minutes each to run in the original version.

Figure 6.1 shows a frame of the output video from an execution of the four benchmarks. Each benchmark creates a window with three images: the left image is the depth map, the middle one is the tracking, and the right one is a raycast of the volume. The colours of the tracking mean the following.

- Grey: it is ok
- Black: no input
- Red: not in the image
- Green: no correspondence
- Blue: too far away
- Yellow: wrong normal

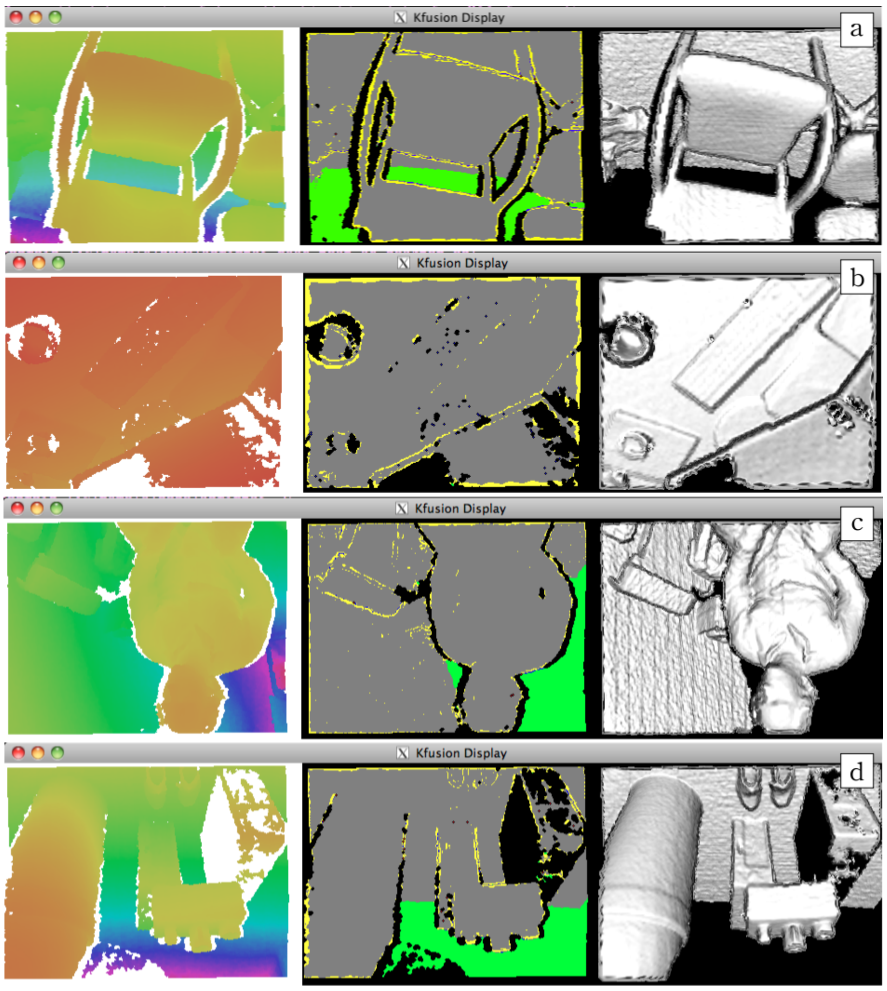


Figure 6.1: Images of the output video obtained by executing KF. The images are the output of the following inputs: (a) chairs, (b) desktop, (c) person, and (d) weird.

6.2 Evaluation

The workflow we followed for porting and accelerating the KF code, as shown in Figure 6.2, was to start with a plain C++ version and port it to OpenCL. Then, having checked its functional correctness, optimise it through performance analysis in the ARM Mali-T604 of the Arndale board.

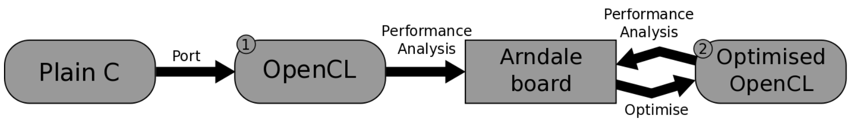


Figure 6.2: Project workflow. The coding steps are: 1, implement the base OpenCL code and 2, optimise the OpenCL code for the Mali-T604.

We also measure the temperature and the power of the chip. The temperature is measured with the system call *sensors* in Celsius degrees (see Section 6.2.2). The power is measured using the Yokogawa power meter (see Section 6.2.3).

For benchmarking I used 4 videos as input files, shown in Figure 6.1. I ran them 5 times each to calculate the average performance, temperature and power. This is a valid method because the input files are long enough to be representative.

The methodology for all different metrics is to get together all the measurements from a single script, so for each run it generates the performance, temperature of the chip and the power consumed by the board. We did the measurements for the original version of the code in three steps: the standalone chip (without cooling), the chip with the heat sink, and the chip with the heat sink plus the fan. After that, I did the measurements with the compiler flags using the heat sink and the fan. From then on, all the executions are done with the heat sink plus the fan to assure stable results.

To validate the results, the program prints the output matrices in each execution and compare it to the matrices obtained in the original code. We validated the results in each optimisation stage. Nevertheless, we allow close enough results, if there is a small difference but it runs faster is also valid, so we can also validate it by watching the output.

6.2.1 Performance measurement

To measure the progress of the project, I profiled the application at different stages: the final clean C++ version, and at each step of the optimisation process of the OpenCL version. All the executions were done in exclusive: no other software running at the same time. The profiling was done with *clock_gettime()* between the main steps for processing each frame. There are six: acquisition of the frame, preprocessing, tracking, integration, rendering and drawing. I also used the profiling tool *gprof* to have an idea of the time spent in each function. I used execution time in seconds as a metric to measure each stage of the computation of a single frame and frames per second (FPS) as a metric to measure the performance of the whole application.

All the executions are done compiled with the following flags: -O3, -mcpu=cortex-a15, -mtune=cortex-a15, -mfloat-abi=hard and -mfpu=neon-vfpv4. Also, we set the frequency governor in "performance".

I implemented a script that runs one benchmark for testing that receives as parameters the name of the benchmark and the version (C++ or OpenCL). I also implemented another script that runs 5 times each benchmark (using the previous) to get three different measurements for each: performance, temperature and power.

For benchmarking I used the scripts to get the three measurements.

6.2.2 Temperature measurement

The temperature was measured by the sensors on the chip. By calling *sensors* of the package "lm-sensors" in the terminal, you get the heat of the chip in Celsius degrees. I implemented a script which waits two seconds between calls to read the sensors and stores the output in a file. We chose two seconds as it is a good compromise to have enough time so the heat changes but without losing important data. There is no important overhead associated with these calls, as it is just a read from a register every two seconds.

Figure 6.3 shows the Arndale board with and without the heat sink during the process of adding it.

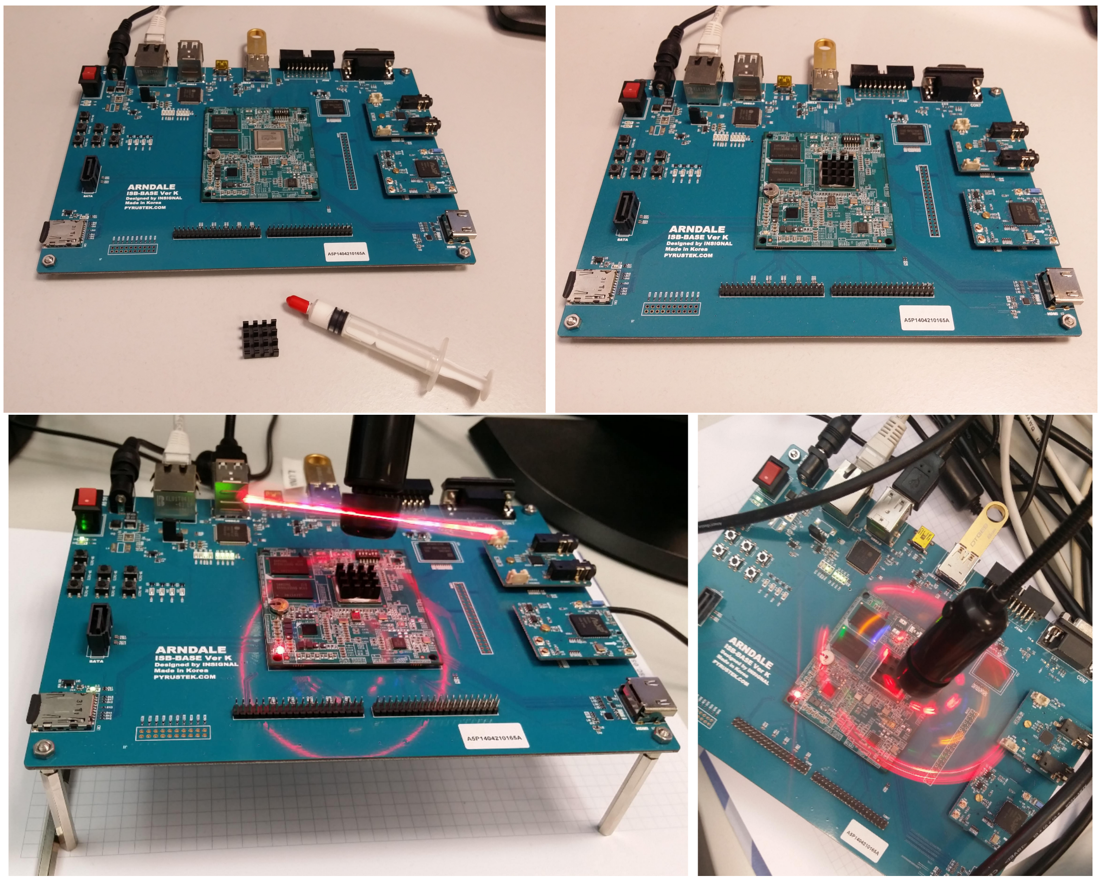


Figure 6.3: On the top left: Arndale board without cooling, before adding the heatsink. On the top right: Arndale board with the heat sink on the top of the SoC. On the bottom: The Arndale board with a fan (with LEDs).

We measured the temperature of the chip for the original version without any cooling, with a heat sink and with a fan. After that, all the measurements are done with a fan except the last one. However, towards the end of the project, the fan broke as it was many hours on and it was not designed for this purpose. So, I took an available fan I found, but it was too big for this board. I solved this by using two toothpicks to hold the fan in around 45° from the SoC, allowing to easily move the heat from the chip to outside. Figure 6.4 shows a picture of the new fan.

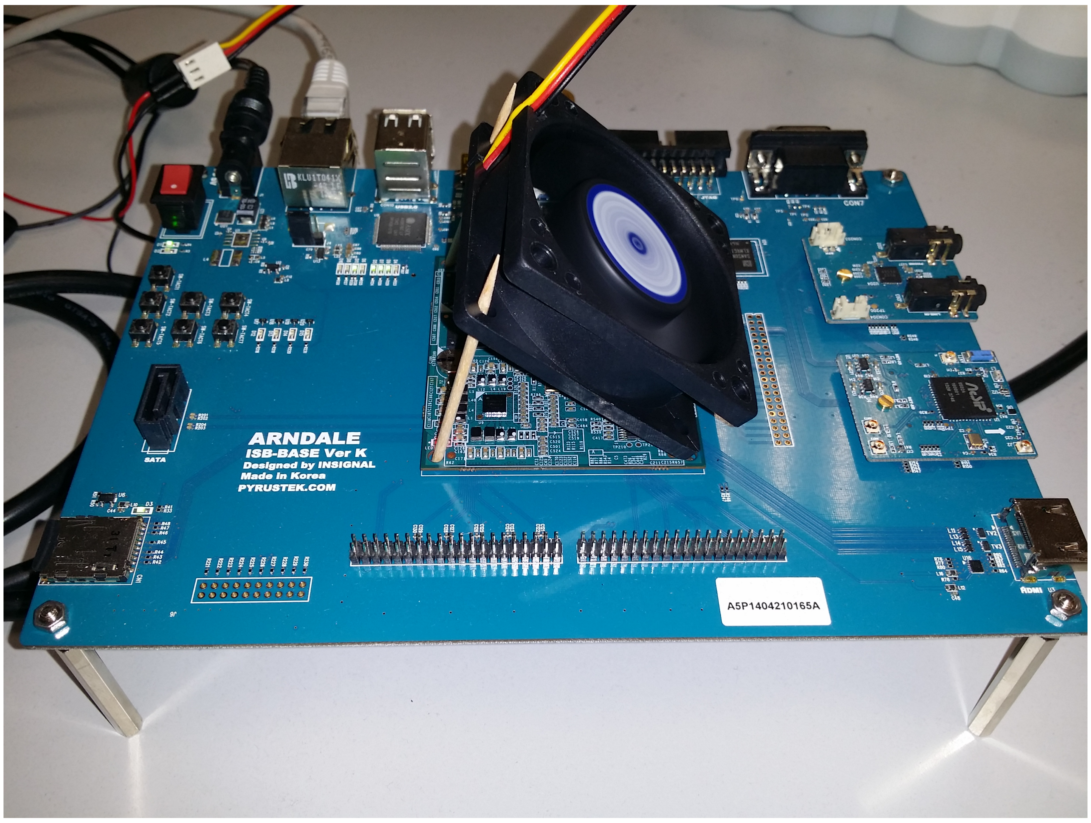


Figure 6.4: Arndale board with the new fan held by two toothpicks in a 45° angle.

As mobile devices have little space, not enough to include any cooling, the last measure is without cooling to get a real insight of the performance that can be achieved in a real device.

6.2.3 Power measurement

We decided to measure the power with the Yokogawa power meter. Figure 6.5 shows the connection scheme used in this project. I connected the power plug of the Arndale board to the Yokogawa, and my laptop to the Yokogawa with a serial cable. I used a script (made by Nikola Rajovic, a colleague at BSC) from the board, which connects to my laptop, reads from the power meter and stores the reading in a file. The frequency of the readings is 0.1s/read. There is no overhead, as all the readings are made from the laptop to the power meter.

The fan is directly connected to the board. Therefore, all the power consumption measurements include the power of the fan.

As power consumption is not a valuable unit to compare optimisations, as we are not taking into account the performance. A better metric is the energy-to-solution [37]. Equation 6.1 shows how to calculate the energy; where E is energy, P is power and t is execution time. Apart from the power consumption, we calculate the energy-to-solution to compare versions. Other

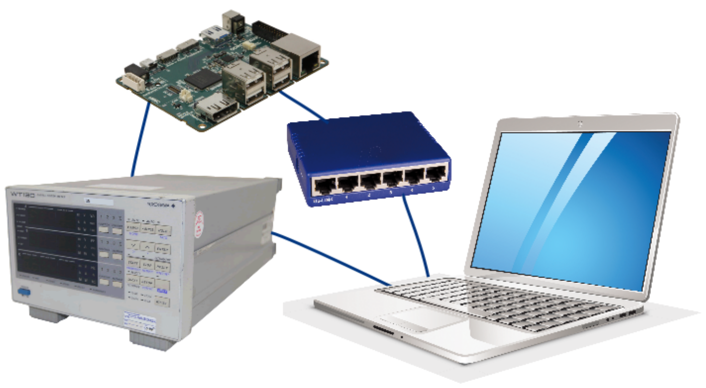


Figure 6.5: Power meter connection scheme.

metrics that give more importance to performance are energy-delay (ED, Equation 6.2) and energy-delay-square (ED^2, Equation 6.3) products.

$$E = P \times t \tag{6.1}$$

$$E = P \times t \times t \tag{6.2}$$

$$E = P \times t \times t \times t \tag{6.3}$$

6.2.4 Methodology deviation

To measure temperature and power was not in our initial plan. We decided to measure the temperature and the power of the board after realising that the standard deviation of each execution of the same benchmark were large. Listings 6.1 and 6.2 show the differences between the standard deviation for the execution time of each stage before and after adding the cooling. I guessed that it was due to dynamic voltage and frequency downscaling to avoid overheating. I checked the temperature of the chip by hand (in a not automated way) a few times and saw that the temperature was changing from 75 to 85 degrees Celsius (being 85 the maximum allowed) during all the execution. The solution we took first was to put a heat sink: it improved the performance and stability, but not enough. Then we decided to put a fan on top of the chip (including the heat sink) which eliminated the thermal throttling. See Section 7.1 for more details on these experiments.

```
Acquisition avg stdev  : 18501.4
Preprocessing avg stdev: 38219.8
Tracking avg stdev     : 9168.27
Integration avg stdev  : 88840
Rendering avg stdev    : 32168.2
Drawing avg stdev      : 137.552
Speed avg stdev        : 0.0365131
Total avg stdev        : 122.969
```

Listing 6.1: Example of the large standard deviation for each processing step after the execution of five times the same benchmark in the chip without cooling.

```
Acquisition avg stdev  : 21.0402
Preprocessing avg stdev: 11343.7
Tracking avg stdev     : 52.7774
Integration avg stdev  : 382.774
Rendering avg stdev    : 120.774
Drawing avg stdev      : 34.0171
Speed avg stdev        : 0.0091228
Total avg stdev        : 21.3441
```

Listing 6.2: Example of the standard deviation for each processing step after the execution of five times the same benchmark in the chip with heat sink and fan.

Also, I found compiler flags specific for the SoC of the development board we use. They were not considered in the initial plan but we thought that they might be interesting to try, to see if they will improve performance. It is wise to use the help of the compiler instead of trying to do optimisations that are already done with no effort. The flags we found are the following: -mcpu=cortex-a15, -mtune=cortex-a15, -mfloat-abi=hard and -mfpu=neon-vfpv4. They are explained in Section 5.5.

6.3 Risk management

There were some possible setbacks in this project.

One of them is, as I was brand new to GPU programming and computer vision and I was not sure how long it was going to take to learn about the algorithm per se and OpenCL, the time to finalise the work was hardly predictable. If running out of time, I would have had discarded running the code in another platform. Also, I left a month to prepare the presentation, which is more than the necessary, for possible delays.

Also, we used the Arndale board to run the tests. As it is a development kit aimed at Android programmers, there is limited documentation and support for Linux. If there are problems with the board, they might be difficult to solve and I would have been unable to use it. This could have been solved by using another board with better support featuring a similar GPU to reuse as much as possible the optimisations done so far.

Chapter 7

Results

In this chapter we present the results obtained during this project.

7.1 Preliminary results

In this section we present the results of the original C++ version in terms of temperature, performance and power. We decided to evaluate the temperature of the chip because we were experiencing variable results between executions (see explanation in Section 6.2.4). We found that there was thermal throttling causing frequency downscaling. Then, we evaluated different cooling solutions trying to achieve consistent performance results: without cooling, with a heat sink, and a heat sink plus a fan.

7.1.1 Temperature

Figure 7.1.a shows the temperature of the chip throughout time. The maximum temperature allowed in the chip is 85°C. From this plot, we understand what the operating system does: when the chip reaches 85°C, the frequency scales down so the chip works slower until lowering the temperature to 75°C as soon as possible not to burn the chip. The temperature grows back fast to 85°C because it is still performing intense workloads. This degrades performance due to the overhead of changing the frequency all the time. Also, the time working at the maximum speed is very low.

Figure 7.1.b shows the same measurements but with a cooling element: a heat sink. In this case, there is still thermal throttling, but less frequently. The time to reach 85°C is longer, and it consequently lowers the number of times that it is necessary to change the frequency. In the first chart, the frequency is changed **ten times** in the interval highlighted by the arrows; while in the second it changes **two times**. So, the chip works more time at a high speed and the overhead of changing frequency is paid less times.

Figure 7.1.c also shows the same measurements but adding a fan on the top of the heat sink. This finally avoids thermal throttling: the temperature never reaches the maximum and stays almost constant. Here, the change of temperature is probably due to the algorithm itself or noise. Working at the maximum speed all the execution time means that there is no overhead associated to frequency scaling.

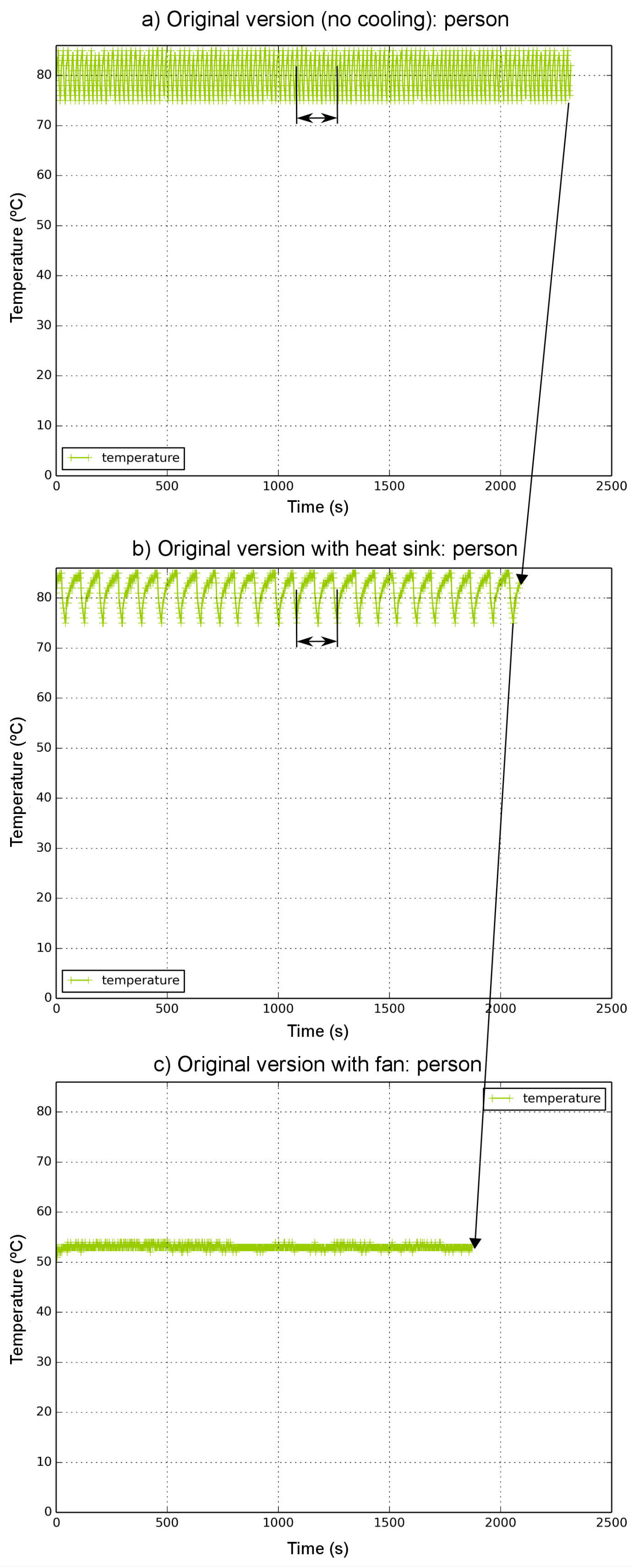


Figure 7.1: Temperature in Celsius degrees comparing the original version (person benchmark) without cooling, with heat sink and with fan throughout time. a) Execution of the original version without cooling elements in the board. b) Execution of the original version with a heat sink as a cooling element. c) Execution of the original version with a heat sink and a fan as cooling elements.

7.1.2 Performance

In the last section, we showed temperature measurements for the three configurations of the Arndale board: without cooling, with a heat sink, and with a heat sink and a fan.

First, comparing the versions without cooling and with a heat sink, we saw that the thermal throttling was not as frequent as without cooling. This has an impact in performance due to being more time in high speed, as shown in Figure 7.2. On average, the heat sink version represents an **11% speed up** over the version without cooling. In the heat sink plus fan version, this is even more clear, performance grows to a 12%-speed up over the heat sink version, and a **24% speed up** over the version without cooling.

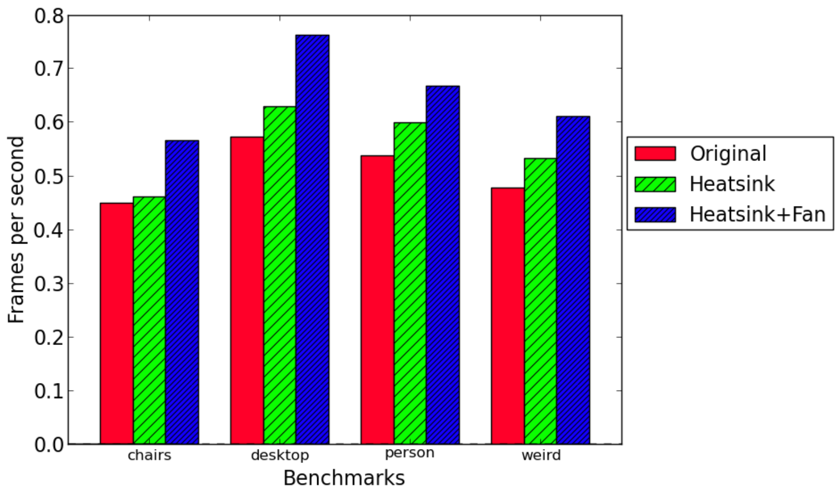


Figure 7.2: Performance in frames per second (FPSs) of the original version for the different benchmarks and cooling configurations.

7.1.3 Power

In the previous sections, we explained the performance difference between executing a program with and without cooling. The best performance was achieved by using two elements: a heat sink and a fan. However, this is not for free. Apart of the price of the cooling elements, we spend more power for each execution. We measured the idle power by connecting a power meter to the board. We obtained that the board was consuming around 7.3W with the fan instead of 6W with only the heat sink, which represents a **22% increase** of idle power consumption. It is not feasible to put these cooling elements in a mobile device and the speed up is approximately the same as the power increase. However, it was an off-the-shelf solution to have stable measurements and a way to investigate the impact of these elements in the board.

Figure 7.3 shows the static and dynamic power consumption of the original version with the different cooling elements for each benchmark. We consider the static power as the idle power. The static power consumption is the same for the original and heat sink versions, but the dynamic grows in the heat sink measurement. This is because there are fewer changes of frequency so it remains more time at high speed. The version of the fan consumes more power for two reasons: the fan adds extra power so the static power consumption is higher; and the dynamic is also higher because it maintains a high frequency during all the execution. The increase of dynamic power of the fan version over the heat sink version is: a **23%** for the chairs benchmark; a **20%** for the desktop; **4%** for the person; and **8%** for the weird.

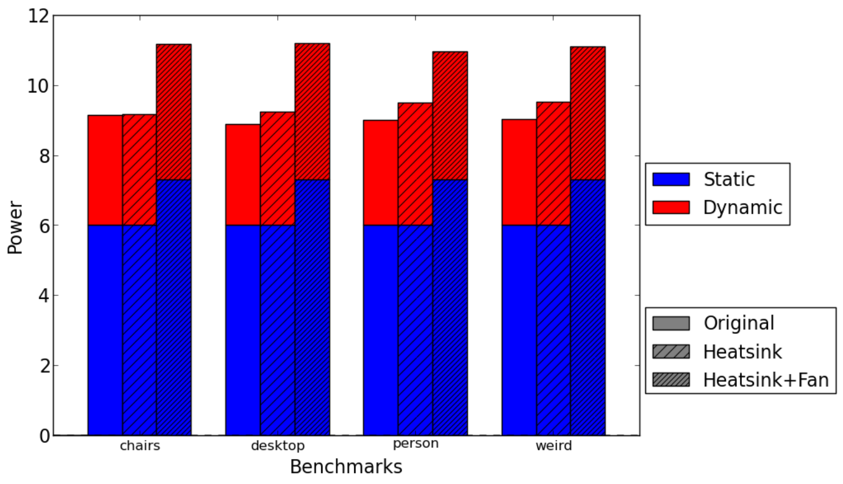


Figure 7.3: Average power consumption in Watts comparing the original version (for all benchmarks) without cooling, with heat sink and with fan.

7.2 Optimisation results

7.2.1 Performance

First, to get the overall picture of each optimisation's impact, Figure 7.4 shows the execution time for all benchmarks and optimisations, and the breakdown per stages of the program.

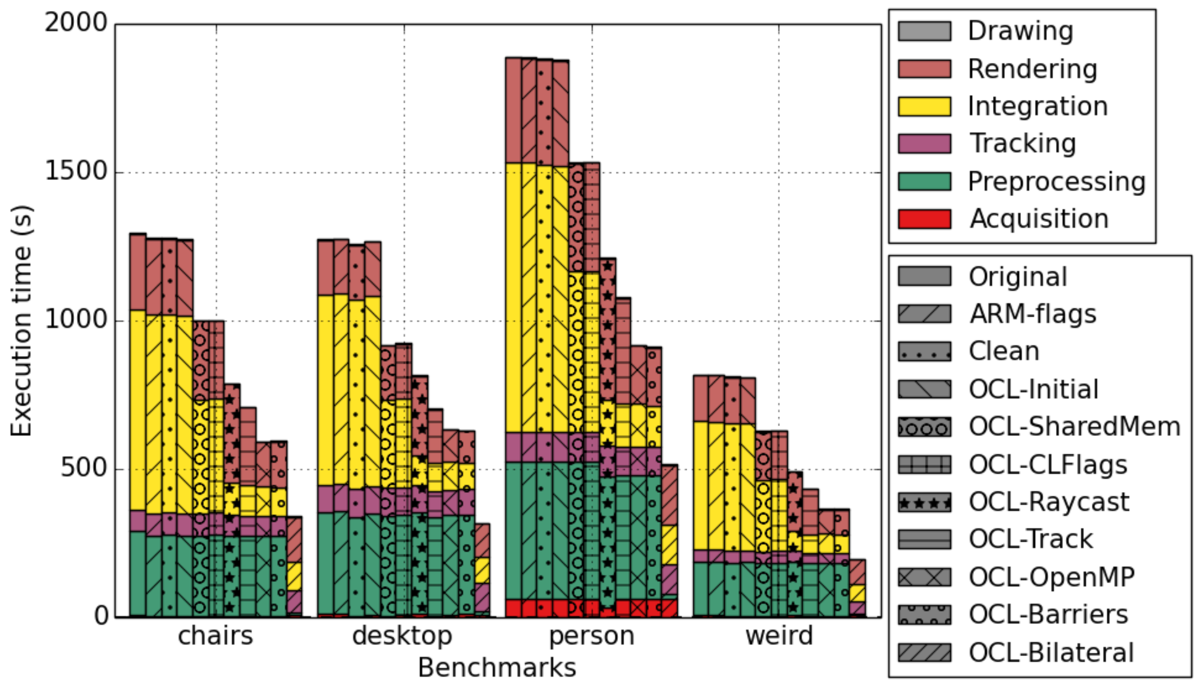


Figure 7.4: Execution time breakdown of each stage in microseconds (us) of all the optimisations for the different benchmarks.

The first two optimisations, ARM-flags and Clean, did not have an impact on performance, probably because the compiler was already doing a good job. Anyhow, the clean version helped to develop the following optimisations. The same happens with the OCL-CLFlags optimisation.

The next two optimisations, OCL-Initial and OCL-SharedMem, were about porting the *integrate* kernel to OpenCL. The former is the first OpenCL-working version, which was inefficiently copying the data to the GPU. The latter makes use of shared memory to avoid copies to the GPU. These optimisations plus OCL-Raycast affect the Integration stage. OCL-SharedMem represented a **45%** execution time reduction of that stage, and OCL-Raycast a **71%** over the former. In total, both together represented an **84%** reduction over the Integration stage. This reduced the overall execution time of the program by **30%**.

The next optimisation, OCL-Track, was about porting to OpenCL the *track* function. The OCL-Track optimisation reduced the execution time in the Rendering stage by **9%**.

Next optimisation, OCL-OpenMP, is applying OpenMP in the CPU-side code to use the second core. This represented a further execution time reduction of **14%** versus OCL-Track.

OCL-Barriers is the version without barriers and CPU/GPU mapping between the kernels of the Integrate stage. This had a negligible impact. Probably, this optimisation would have more impact in a GPU without shared memory, where the overhead of these calls are larger.

Last, OCL-Bilateral is about porting one more kernel, the *bilateral_filter*. This dramatically reduced the execution time of the Preprocessing stage by **96%**. This represented an execution time reduction of **46%** versus the previous version.

Comparing the original version to the last version with all optimisations, we achieved a **74%** execution time reduction, which translates to a speed up of **3.93**×.

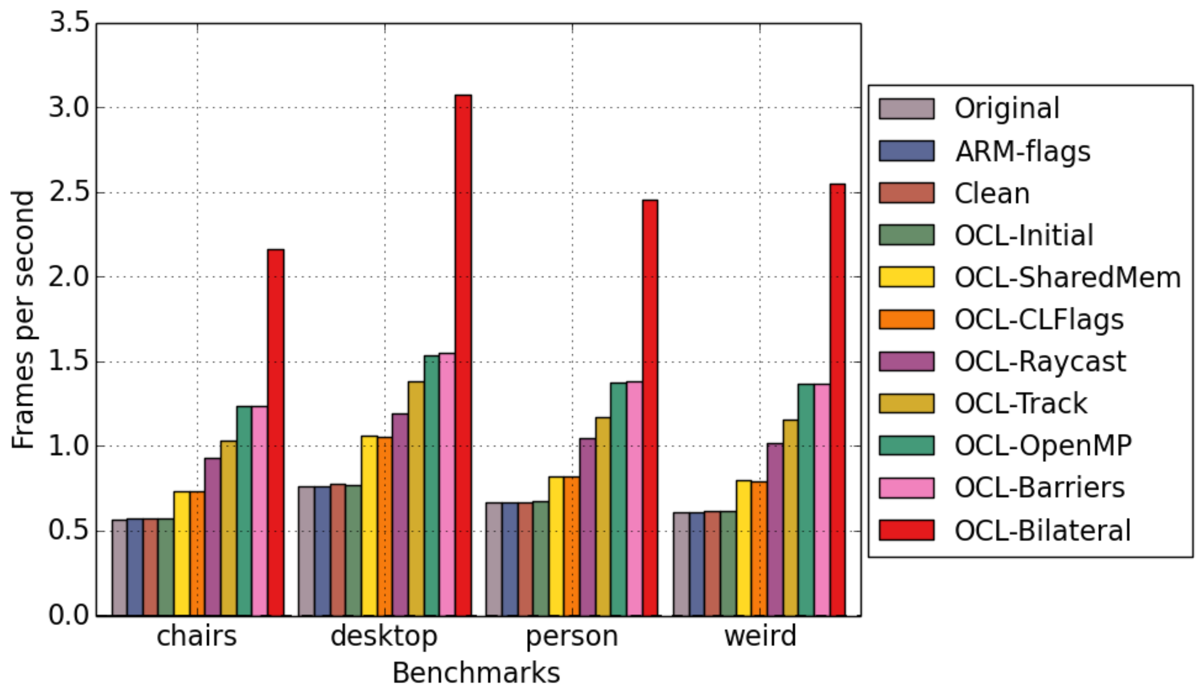


Figure 7.5: Performance in FPS of all the optimisations for the different benchmarks.

Figure 7.5 shows the achieved FPS for each version. From the original 0.6–0.7 FPS, combining all optimisations we achieved between 2.2–3.1 FPS.

7.2.2 Power

Figure 7.6 compares the power consumption for each optimisation. As we had to change the fan for a more powerful one, the static power is higher than in the previous section. The power consumption of the first four versions, from Original to OCL-Initial is similar, which makes sense because performance was also similar. Moving the integrate kernel to the GPU in OCL-Initial did not have an impact neither in performance nor in power. The OCL-SharedMem version actually improved performance at no power cost. This is because this optimisation is about ”doing less stuff” by avoiding CPU-GPU memory copies.

The first optimisation with a significant impact in power is OCL-OpenMP. This is because not only the GPU is more in use, but also both cores in the CPU side are working in parallel. Then, OCL-Bilateral also implies an increase in power consumption, but in a much lower proportion than the reduction in execution time of almost half mentioned before.

Comparing the original version with the last optimisation (OCL-Bilateral), the power consumption increased in **38%**. If we compare the power consumption increase with the performance increase, we see a clear benefit: power increased by 38% while performance grew by 293%. This large difference is mainly due to the portion of static power, which is paid regardless of the performance achieved.

In terms of energy-to-solution for all benchmarks, we get that the original version consumes 58409J and the optimised version 19850J, which represents a **66%** reduction in energy. For other metrics such as energy-delay (ED) and energy-delay-square (ED^2) products, the optimised version reduces them in 91% and 98%, respectively. This means 3× better energy-to-solution, 11× better ED and 44× better ED^2.

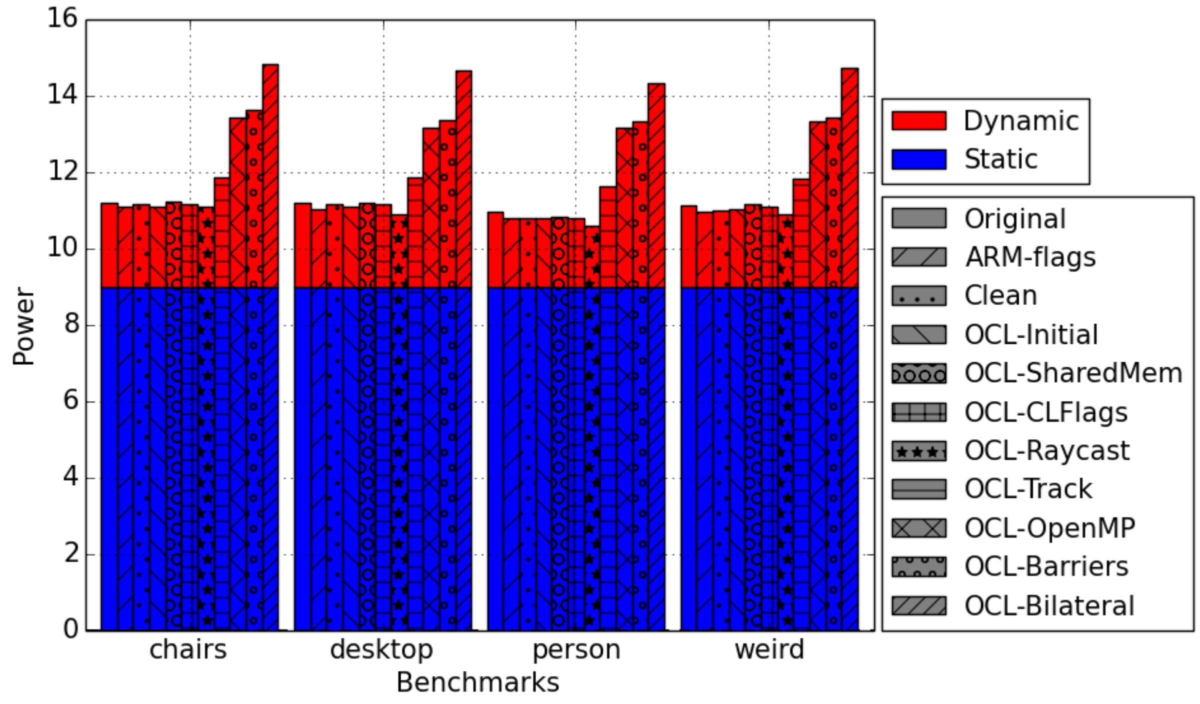


Figure 7.6: Average power consumption in Watts comparing all optimisations and benchmarks.

7.3 Final results

Considering only the optimisations in the Integration stage, the speed up for that stage is **7.13**×. For the whole application, this means a speed up of **1.60**×. In Section 5.2 we calculated with Amdahl's law in Equation 5.2, the maximum theoretical speed up assuming infinite resources and the result was 1.99×. With our optimisations and considering the limited resources of an embedded platform, achieving 1.60× over 1.99× is a very good result.

We achieved a **5**× speed up over the original C++ version without cooling. Comparing the fan versions (original and optimised), the speed up is **4**×.

We also ran our optimised version in the board without any cooling. In this case, we achieved a **3**× speed up over the original version without cooling. This is shown in Figure 7.7 for all benchmarks.

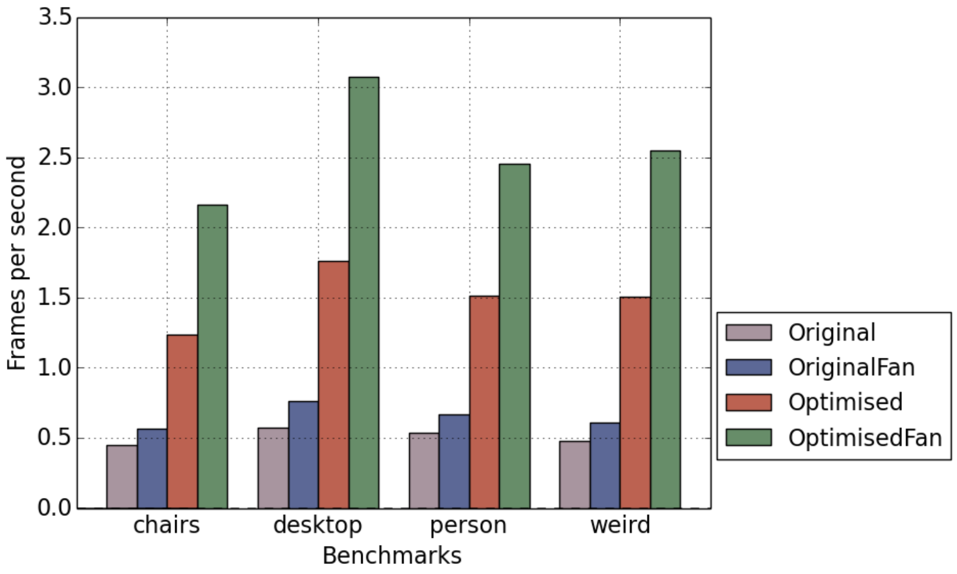
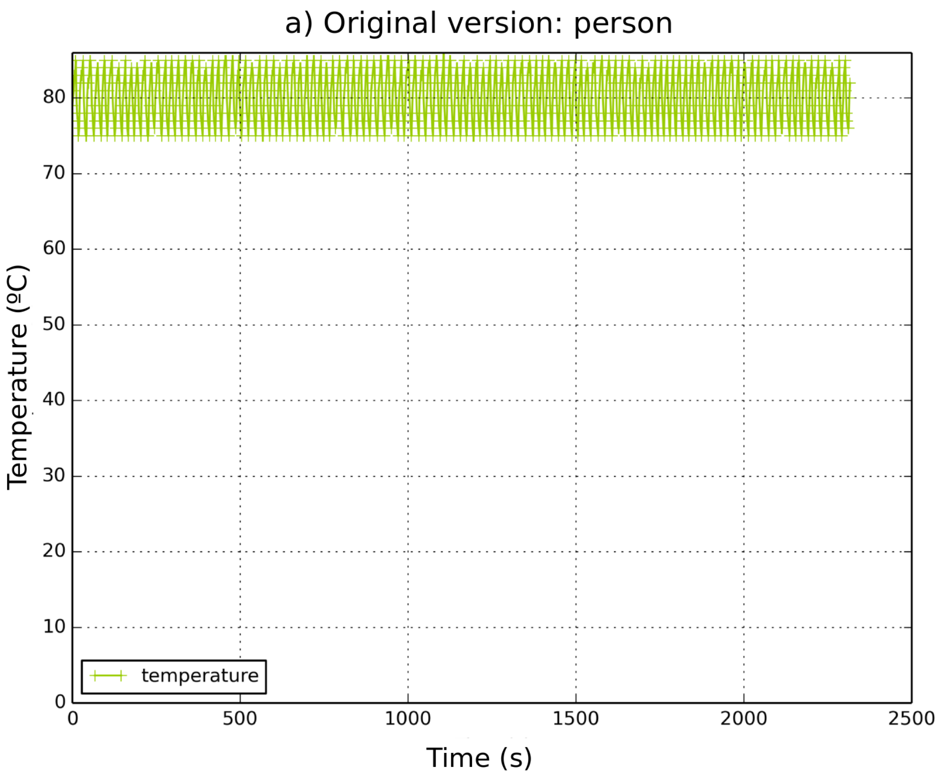


Figure 7.7: Performance comparison in FPS of the original version and the optimised version for the different benchmarks and cooling configurations.

With our optimisations, not only we optimised performance, but also we reduced the frequency of thermal throttling. Figure 7.8.a shows the temperature for the original version, and Figure 7.8.b shows the temperature for the optimised one. In the optimised version the GPU is active for long periods of time leading the CPUs to be idle waiting the GPU to finish. Our hypothesis is that the heating produced by the CPUs is lower in the optimised version than in the original due to this lower activity of the CPUs. Although the GPU is active in the optimised version and not in the original, it seems that this activity does not generate an amount of heat that compensates the lower utilisation of the CPUs. This would result in overall lower heating in the optimised version and therefore explain the lower thermal throttling.

In terms of power, as Figure 7.9 shows, the power consumption is practically the same. So, for the same power, we achieve better performance and more stable results as the frequency of thermal throttling is lower. This shows the benefits of the larger energy efficiency of the GPU.



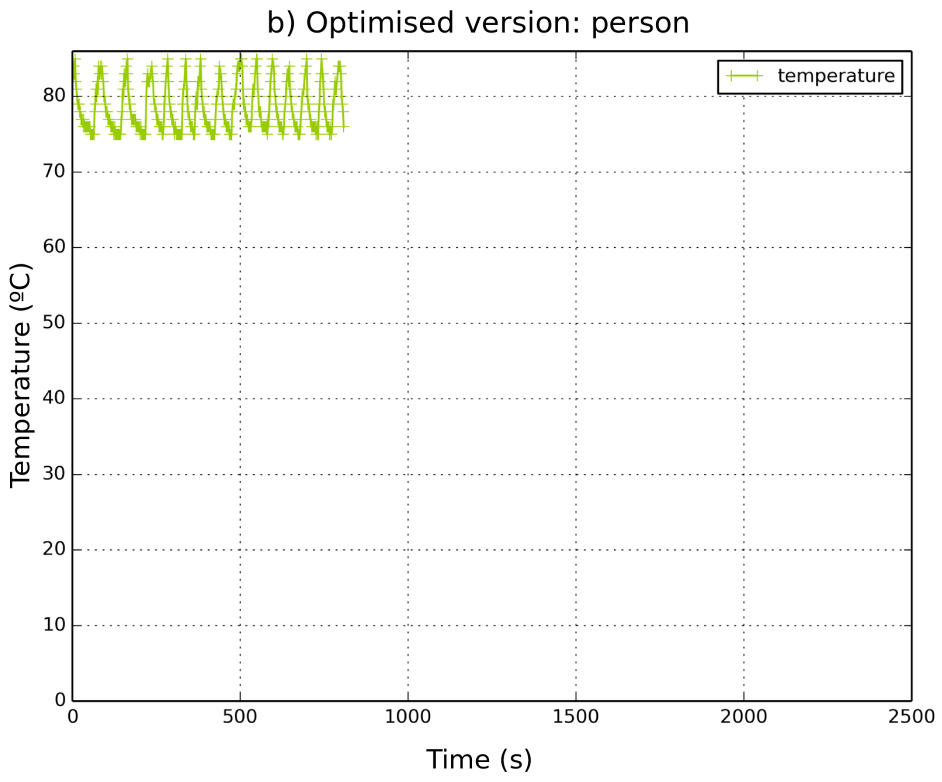


Figure 7.8: Temperature in Celsius degrees comparing the original version (person benchmark) with the optimised version. a) Execution of the original version without cooling elements in the board. b) Execution of the optimised version without cooling elements.

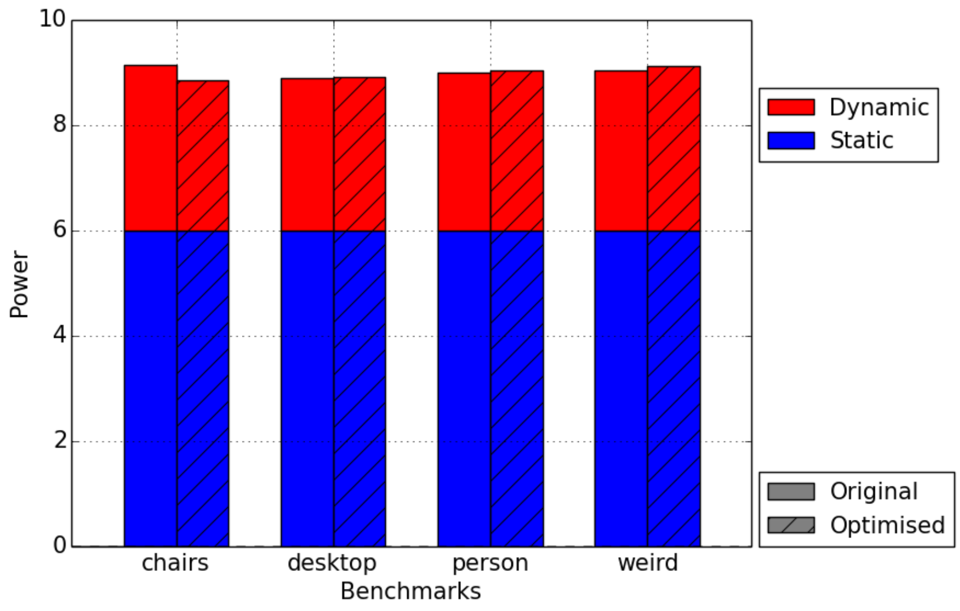


Figure 7.9: Average power consumption in Watts comparing the original and optimised versions without cooling.

7.4 Evaluation in Odroid-XU3

We evaluated our optimised code in a different development board, the Odroid-XU3, without cooling. This board, as explained in Section 2.3 integrates a more powerful SoC featuring the next generation of the GPU. The optimised version runs at a speed between 1.8 FPSs. This is 3.5× faster than the original version in the Arndale board but only 1.2× faster than the optimised version in the Arndale board.

Figure 7.10 shows the execution time breakdown of our optimised version on the Odroid-XU3 compared to the execution of the original and optimised on the Arndale. The main differences between the optimised versions are the Rendering and Tracking stages, all other stages take a similar time to execute. We executed the program in the Odroid with two CPUs that are exactly the same as the ones in the Arndale board but running at a higher speed. The GPU which is similar to ours but with six cores, is seen as two different devices, and we are only using one of them because the application is not prepared to run on more than one device. In the optimised version, the *track* kernel from the Tracking stage runs in the GPU and the Rendering stage is completely executed on the CPUs. So, the stages that run mostly on the GPU take similar time on both platforms. However, the stages that run mostly or completely on the CPU run faster on the Odroid because the CPUs run at a higher frequency.

Figure 7.11 shows the comparison of the performance achieved by the Arndale and the Odroid. As explained before, the Odroid takes less time to execute the optimised version, which leads to higher FPS for each benchmark.

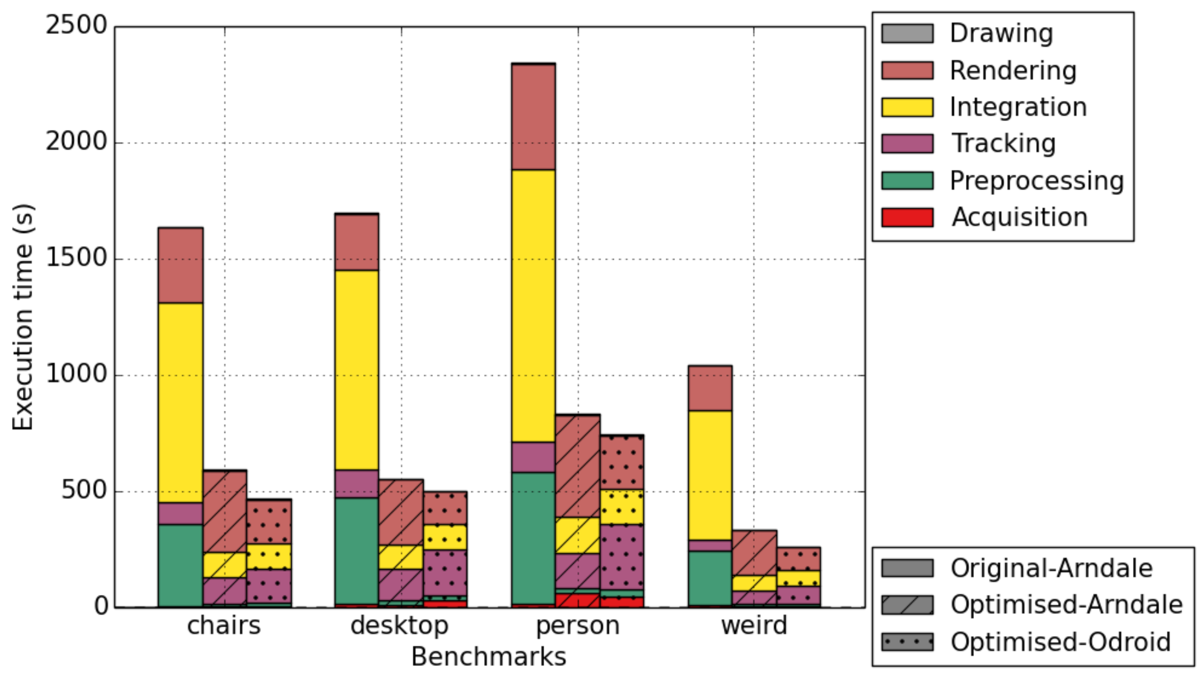


Figure 7.10: Execution time breakdown of each stage in microseconds (us) of the original and optimised versions on the Arndale board without cooling, and the optimised version on the Odroid-XU3.

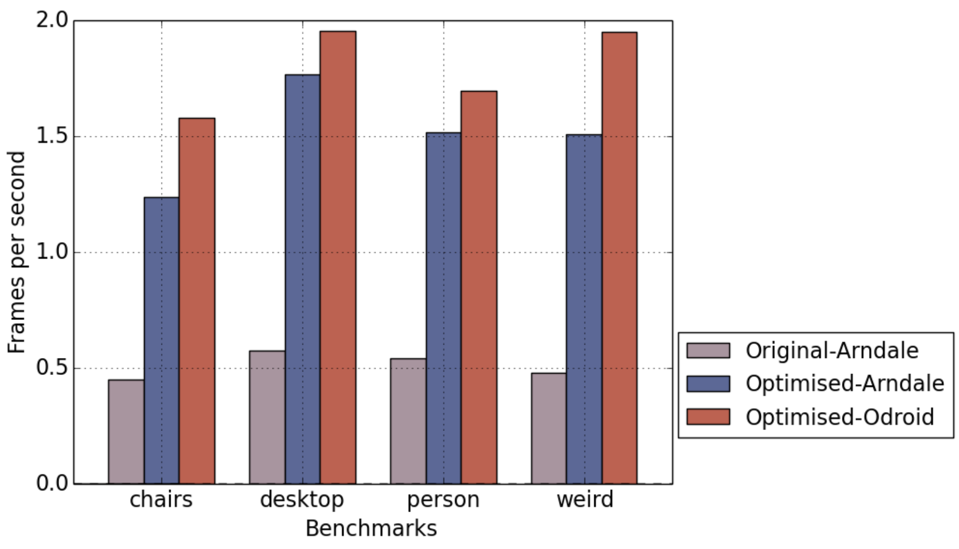


Figure 7.11: Performance comparison in FPS of the original and the optimised versions on Arndale and Odroid-XU3 for the different benchmark.

Chapter 8

Lessons learnt

During the development of this project I learnt many things. Some of them can be very useful for a similar project.

8.1 OpenCL porting

I learnt OpenCL from scratch during this project. I did not know about GPU programming at all. This implied to self-study GPU architecture and OpenCL, but there were some details that if you do not study them, it may lead you to suppose something works differently. The OpenCL standard has some specificities that are not obvious and may lead to errors. Among these there is the memory alignment issue explained in Section 8.1.3. Also, at first one does not know the complexity of certain programming steps until trying them, for example, debugging.

8.1.1 OpenCL language

OpenCL is based on C99. This means that all the code executed in the GPU must be written in C99 only. That is a problem when using other libraries in other languages like C++. This is an extra work added on the top of the OpenCL porting.

In this project, it was necessary to translate some data structures and functions from C++ to C to use them from the OpenCL kernels.

8.1.2 OpenCL libraries

OpenCL lacks of a high-level API. This makes long and error-prone codes. I had to create my own class to encapsulate many OpenCL calls in wrappers. So it is also more difficult to learn this language and create programs as it is difficult to understand and remember the steps to follow.

8.1.3 Memory alignment

Vector types of size 3 (e.g., float3, uint3) do not exist in OpenCL, they are defined as size 4 (e.g., float3 is defined as float4). Vector types must be aligned to powers of 2 to pass them as arguments to a kernel. I learnt this in the hard way after some days of debugging.

The problem was that in the OpenCL kernels file, *float3* and *float4* are defined by the OpenCL library with size four (same for both). But, in the host, OpenCL has the types *cl_float3* and *cl_float4*, which have the same sizes as in the kernel (four floats each) and therefore can be used as arguments of a kernel. But "vector_types.h" defines float3 and float4, and these are used by the host, so, the arguments passed from the host, and the ones received by the kernel had different sizes (a nightmare to debug). This gives a compilation error having the *float3* type passed as an argument. But it does not complain if you have those inside a struct that you pass as an argument.

I was passing a struct with two elements of float3 and a pointer, to an OpenCL kernel, and it was compiling but behaving erroneously. The values of the *float3* attributes inside the struct were read incorrectly (due to the different sizes assumed by the host and the device), so the calculations were terribly wrong at some point, and it was hard to find the bug. I spent a few days to find the bug and then to find how to solve it in a decent way.

I tried to redefine *float3* (the one defined as three floats) to cl_float3 (the one with size four from the OpenCL API) but the file "cutil_math.h" defines some operators for *float3* and *float4*, and since these are the same in OpenCL (literally, "typedef cl_float4 cl_float3;") there is a problem of duplicate operators.

I solved this issue by creating a *cl_float4* and using the values in the first three positions. This was the easiest way of fixing this and still reasonable as it is not a big overhead.

8.1.4 Debugging

Debugging this program was not straightforward. Not only for running in a GPU (impossible to print values as with *printf*) but also for the large size of vectors and matrices used. I read that it is possible to debug OpenCL kernels with *GDB* (passing the required flags when compiling the kernels' file), but there are also several problems depending on the platform where it will run. I have not tried to use it because I think the effort was not worthy as having large data structures would be too complex to read position by position.

One approach I used was to create an extra buffer to store some values in the kernel and read them in the CPU and print them to compare them with the original version ones. This means to, at least: create a CPU pointer, create an OpenCL buffer, set the argument for the kernel, change the kernel code to receive the extra argument, store the results in the buffer, map the buffer to the CPU and print the buffer. All these steps could potentially introduce another bug. Another problem was to select a small portion of the buffer to print that was a representative one, as they were very large.

Other approach I used, mainly to get an overall picture of what is happening and where, was to modify or remove parts of the target kernel and watch the differences in the output. This was easy and quick as I even did not have to compile, as the OpenCL kernels are compiled at runtime.

When porting the kernels I had to debug because the output was not correct at first. Some results were wrong after several frames and only at some specific positions, so debugging was a challenge itself. I compared the versions using *vimdiff.* I chose different debugging techniques depending on what was happening in the output.

8.1.5 Shared memory

Mobile devices do not have a powerful GPU but they usually have shared memory. It helps to reduce the time of sending the data from the CPU to the GPU. This does not help much as usually the kernels selected for running in a GPU are the ones that have enough computation to improve performance despite the overheads. Nevertheless, this allow us to remove the overheads of transferring data and create more kernels. See Section 2.4 for more details and Section 5.5 for reading about my optimisation.

8.2 Frequency throttling

We saw that the standard deviation of the performance when running the benchmarks several times was considerably high. We decided to check the temperature during one execution and I saw that was pretty close to the maximum temperature allowed. I created a script to save the temperature of the SoC in a file every 2 seconds and I made a plot with it. I realised that there was frequency throttling.

My supervisor agreed to buy a heat sink and a fan to fix this so the results will be more stable.

8.3 Using development boards

Using development boards has inconveniences as it is not a product widely sold, but a product for developers and researchers. The consequence is having poor support and little information about them.

8.3.1 Disk space

The Arndale board has limited disk space. The problem was that the input files were too large to store them locally. First, I tested the application with just one input file in disk, which was the maximum I could fit. I thought that the best way of solving this was to use a USB stick to store all the data sets, but my intuition told me that this was going to decrease performance. So, I added in the script that executes all the benchmarks, an instruction to load the input file needed from the USB stick to disk before the execution. Then I also tried to execute everything directly from the USB and I realised that both performed equally. So, we decided that using the files directly from the USB was correct.

8.3.2 Graphics

The drivers for the GPU of the Arndale board are proprietary of ARM. The board we used came directly from ARM with the OpenCL drivers because ARM licensed the drivers to the partners of the PAMELA project. The output graphic windows that open when executing KF were shown in the host computer through the network (via ssh). To avoid the network overheads, we tried to install *xorg*, the X's server and *openbox*, a light window manager, to display it directly in the board and watching them through HDMI. This was not possible as we are missing the drivers to use the GPU for graphics display.

8.4 Technical and soft skills

I learnt about:

- CV
- GPU programming
- OpenCL
- Development kits and how to use them
- Inkscape, a very useful tool for creating images
- Working with version control software: git and svn
- Python and the use of matplotlib for creating plots
- How research environments work

I improved some skills:

- Writing with LaTeX
- Automation of tasks through scripts
- Presentation skills (poster presentations and GEP)
- Team skills (having to ask for help or advice to PhD students, postdocs and professors working on the same topics or with knowledge on something I needed to learn)
- English skills (all of them: speaking, listening, reading and writing)
- Programming with vim

I achieved many things by doing this project:

- A job offer to work at BSC with the same and similar hardware
- I lead a team to compete in the Student Cluster Competition at ISC15 with the same hardware I used in this project
- The acceptance in the MIRI master
- The attendance to several research conferences and events
- Meeting important people of the field

Chapter 9

Conclusions

Mobile devices tend to be heterogeneous systems-on-a-chip with specialised processing elements or accelerators for faster and more energy-efficient computation of certain computations, including GPUs. Those GPUs were being used only to accelerate graphics. Nowadays, the performance of embedded GPUs, and mobile SoCs in general, is improving, and embedded GPUs are becoming general purpose GPUs. This context and the many sensors mobile devices have, creates the perfect environment for Computer vision (CV) in mobile devices.

CV applications consist of applying several computations on a large amount of data, commonly to a matrix of pixels. These algorithms are parallel by their input files and its own nature. GPUs are very useful for speeding up graphics but also for speeding up this kind of computer vision applications.

We optimised the KinectFusion (KF), as a representative of CV applications, for an specific SoC commonly found in mobile devices using the Arndale development board. The main optimisations consisted on porting parts of the application to the GPU on which they execute faster and requiring less energy thanks to the higher energy efficiency of the GPU compared to the CPU for compute-intensive operations. We also executed our optimised version on a newer and more powerful platform, the Odroid-XU3 development board. Just by recompiling and rerunning the optimised code, we achieved better performance, which demonstrates the performance portability of our optimisations. The optimised version resulting from our work serves as a base code that can be easily adapted to other platforms with embedded GPUs.

From what we learnt during this project, we can anticipate the difficulties that may be encountered for anyone trying to implement CV on mobile devices. These challenges are not only due to performance but also due to software and hardware constraints. Regarding software there are two main challenges. Firstly, not all vendors have compute-capable embedded GPUs or they only have support for their own specific language. We had a variety of development boards with mobile SoC and we could only find one of them with both OpenCL hardware and software support, and at the middle of the project we acquired another just launched to the market. Secondly, the OpenCL library needs a high-level API. I created my own small one to avoid errors and simplifying the code. OpenCL requires you to execute more than one instruction to do a single operation and also with a large amount of parameters. Programming with OpenCL without a high-level API is not only tedious and difficult to learn but also error-prone. Talking with other people working with OpenCL, they told me that they have implemented their own high-level library. It should exist an official and common one.

Regarding hardware constraints, apart from the compute-capable support, by the nature of the packaging of a mobile device, it is difficult to dissipate heating. Current mobile chips are not ready for sustained full utilisation because the chip overheats and starts throttling, slowing down performance. In this project, we saw that the standard deviation of the performance of each execution was quite large. Consequently, we decided to measure the temperature of the chip and we realised that there was thermal throttling. Each benchmark execution was between 15 and 45 minutes, enough time to become an important problem. We thought that a heat sink may solve this issue but it just improved it a bit. Then we added a fan which solved the problem. Nevertheless, we cannot include these cooling elements on a real mobile device as they do not fit. Also, the performance gained was not enough comparing the extra power it consumes with the fan.

Despite all our optimisations, we have learnt that mobile devices are not yet ready for computer vision. Combining all our optimisations using the GPU, we obtained between 2.2 and 3.1 FPS which represented a 4× speed up compared to the original CPU version. Although this is still far from an interactive rate (30 FPS), the work in this project is one step forward towards this target. We contributed to this objective, and we expect that follow-up efforts and works in other projects such as PAMELA and Google Tango will achieve it in a near future.

Chapter 10

Future work

From the work presented in this thesis, we propose the following topics of future work:

- Optimise the application: further optimisations in the GPU or the CPU.
  There are some optimisations yet to be done in both devices, for example, vectorisation.
- Optimise the application: Port more kernels.
  Porting many kernels may allow us to fuse some of them and reduce OpenCL overheads by not using much the CPU. Also, if all the kernels that modify the main buffers are in OpenCL we can remove the clFinish() call as the command queue is the same and is executed in order (no race conditions due to outputs/inputs) and we can avoid mapping them to the CPU. This will significantly reduce the OpenCL overhead.
- Optimise the application: Adding other programming models.
  For example, to add OmpSs to the OpenCL version. This may help for better distribute tasks both in the CPU cores as in the GPU.
- Further research in which are the most common computations in CV applications.
  Analyse more CV applications to find the different computation patterns they have. This enable us to know which computations are more necessary to improve either for software or for hardware.
- Performance analysis in other platforms.
  This would consists of analysing the same code in different platforms. This will help us to understand which hardware is better for this application and how they can be improved for this purpose.

Chapter 11

Bibliography

[1] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in heterogeneous computing," *Scientific Programming*, vol. 18, no. 1, pp. 1–33, 2010.

[2] J. Rivera and R. van der Meulen, "Gartner Says Worldwide Traditional PC, Tablet, Ultra-mobile and Mobile Phone Shipments to Grow 4.2 Percent in 2014." `http://www.gartner.com/newsroom/id/2791017`, jul 2014.

[3] N. Trevett, "Accelerating mobile augmented reality." Keynote at InsideAR Conference, 2012.

[4] B. Smith et al., "More iPhone Cool Projects," June 2010.

[5] "PAMELA project." `http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/K008730/1`. Last accessed: September 2014.

[6] J. Stone et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[7] R. Klette, "Concise computer vision," 2014.

[8] "IKEA catalogue app." `http://www.ikea.com/gb/en/catalogue-2015/index.html`. Last accessed: October 2014.

[9] "Word Lens." `http://questvisual.com/`. Last accessed: October 2014.

[10] S. Izadi et al., "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, 2011.

[11] "Odroid-XU." `http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137510300620`. Last accessed: October 2014.

[12] "Jetson TK1." `https://developer.nvidia.com/jetson-tk1`. Last accessed: October 2014.

[13] "Arndale board." `http://www.arndaleboard.org`. Last accessed: October 2014.

[14] "Odroid-XU3." http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127&tab_idx=1. Last accessed: January 2015.

[15] N. Rajovic et al., "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, (New York, NY, USA), 2013.

[16] NVIDIA, "Compute Unified Device Architecture (CUDA) C Programming Guide." http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. Last accessed: October 2014.

[17] P. Du et al., "From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming," *Parallel Computing*, vol. 38, no. 8, pp. 391–407, 2012.

[18] "OpenACC Home." http://openacc.org. Last accessed: October 2014.

[19] K. Gregory and A. Miller, *C++ AMP: Accelerated Massive Parallelism with Microsoft® Visual C++®*. " O'Reilly Media, Inc.", 2012.

[20] A. Duran et al., "OmpSs: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[21] M. Lippmann, "Integral photograph," *J. Phy*, vol. 7, p. 821, 1908.

[22] N. Rajovic et al., "The Low-power Architecture Approach Towards Exascale Computing," in *Proceedings of the Second Workshop on Scalable Algorithms for Large-scale Systems*, ScalA '11, (New York, NY, USA), pp. 1–2, 2011.

[23] "Mont-Blanc project selects Samsung Exynos 5 Processor." http://www.montblanc-project.eu/press-corner/news/mont-blanc-project-selects-samsung-exynos-5-processor-0. Last accessed: October 2014.

[24] I. Grasso et al., "Energy Efficient HPC on Embedded SoCs: Optimization Techniques for Mali GPU," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 123–132, 2014.

[25] A. Maghazeh et al., "General purpose computing on low-power embedded gpus: Has it come of age?," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, pp. 1–10, IEEE, 2013.

[26] "Irida Labs Website." http://www.iridalabs.gr. Last accessed: October 2014.

[27] "Movidius Website." http://www.movidius.com. Last accessed: October 2014.

[28] "Project Tango." https://www.google.com/atap/projecttango/. Last accessed: May 2014.

[29] K. Cheng and Y. Wang, “Using mobile GPU for general-purpose computing–a case study of face recognition on smartphones,” in *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*, pp. 1–4, IEEE, 2011.

[30] G. Wang et al., “Computer Vision Accelerators for Mobile Systems Based on OpenCL GPGPU Co-Processing,” *J. Signal Process. Syst.*, vol. 76, pp. 283–299, Sept. 2014.

[31] “Driverless cars.” `http://www.forbes.com/sites/chunkamui/2013/01/22/fasten-your-seatbelts-googles-driverless-car-is-worth-trillions/`. Last accessed: October 2014.

[32] “Car insurances.” `http://motherboard.vice.com/read/driverless-cars-are-going-to-kill-insurance-companies`. Last accessed: October 2014.

[33] “e-Waste.” `http://www.greenpeace.org/international/en/campaigns/toxics/electronics/the-e-waste-problem/what-s-in-electronic-devices/bfr-pvc-toxic/`. Last accessed: September 2014.

[34] M. Hill and M. Marty, “Amdahl’s law in the multicore era.,” *IEEE Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[35] S. Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd Edition)*. Addison-Wesley Professional, 2005.

[36] D. Chisnall, “Understanding ARM Architectures.” `http://www.informit.com/articles/article.aspx?p=1620207&seqNum=4`. Last accessed: January 2015.

[37] E. L. Padoin, D. A. de Oliveira, P. Velho, and P. O. Navaux, “Time-to-solution and energy-to-solution: a comparison between arm and xeon,” in *Applications for Multi-Core Architectures (WAMCA), 2012 Third Workshop on*, pp. 48–53, IEEE, 2012.

Acronyms

ABI application binary interface. 30

AR augmented reality. 4

CPU central processing unit. 1, 6–12, 23, 29–31, 44–46, 48, 51, 52, 54

CV computer vision. 2–5, 7, 11, 12, 15, 23, 24, 53, 54, 56

FLOPS floating point operations per second. 7

FPS frame per second. 35, 42, 44, 46, 48, 49, 55

GPGPU general purpose graphic processing unit. 6, 7, 12

GPU graphics processing unit. 1–3, 5–12, 23, 24, 30–32, 39, 44–46, 48, 50–56

KF KinectFusion. v, 2, 3, 5, 12, 15–17, 25, 28, 33–35, 52, 54

LED light-emitting diode. 36

MMU memory management unit. 6–8

SIMT single instruction multiple threads. 6

SoC system-on-a-chip. 1, 2, 6–8, 11, 36, 37, 39, 48, 52, 54