UNIVERSITY OF LINCOLN

BACHELOR (HONS) THESIS

Parallelisation of Motion Estimation Techniques Applied to Cardiac Imaging

Author:
Raymond KIRK

Supervisor:
Dr. Grzegorz CIELNIAK

A thesis submitted in fulfilment of the requirements for the degree of BSc (Hons) Computer Science

in the

School of Computer Science

September 21, 2017

Abstract

Parallelisation of Motion Estimation Techniques Applied to Cardiac Imaging

by Raymond KIRK

This thesis focuses on the implementation of a Full Search Block Matching Motion Estimation algorithm using sum of absolute difference error criterion to find optimal block displacement, and to estimate the heart rate of apical four chamber view ultrasound images. With the rapid growth of GPU processing power in recent years an effective deployment medium for parallelisation of traditionally serial algorithms exists. This project will attempt to evaluate the feasibility of using heterogeneous systems for motion estimation by comparing algorithm execution times and accuracy on the CPU and GPU. This thesis concludes by showing the GPU implementation is sufficiently fast enough for real-time processing and can shorten computation time of accurate heart rate estimates by a factor of 450 times.

Acknowledgements

I would like to thank and express my greatest appreciation to both my friends and family in the support provided throughout writing this thesis. Also to thank the University of Lincoln faculty for their dedicated time and effort. With a special thank you to both Dr. Grzegorz CIELNIAK and Dr. Massoud ZOLGHARNI , who have both been extremely helpful providing their time and patience throughout writing this thesis. They're a credit to the University and have taught me so much in such a short period of time.

In particular, I would like to extend my gratitude to Louella Minter for being a source of inspiration throughout the entire duration of my academic career to-date and for fulfilling every day with joy and a challenge to succeed - my motivation is superseded with your encouragement. A special thank you also to Joseph Donaldson for making the three years as an undergraduate fun, the time wouldn't have been nowhere near as enjoyable without you.

Contents

List of Figures

List of Abbreviations

BPM	**B**eats **P**er **M**inute
SAD	**S**um of **A**bsolute **D**ifferences
DICOM	**D**igital **I**maging and **C**ommunications in **M**edicine
CT	**C**omputed **T**omography
MRI	**M**agnetic **R**esonance **I**maging
JPEG	**J**oint **P**hotographic **E**xperts **G**roup
MPEG	**M**oving **P**icture **E**xperts **G**roup
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
API	**A**pplication **P**rogramming **I**nterface
OpenCL	Open **C**omputing **L**anguage
OpenCV	Open Source **C**omputer **V**ision Library
FSBM	**F**ull **S**earch **B**lock **M**atching
MAD	**M**ean **A**bsolute **D**ifference
FFT	**F**ast **F**ourier **T**ransform
DFT	**D**iscrete **F**ourier **T**ransform
SDLC	**S**oftware **D**evelopment **L**ife **C**ycle
MATLAB	**Mat**rix **L**aboratory
RAD	**R**apid **A**pplication **D**evelopment
GUI	**G**raphical **U**ser **I**nterface
IDE	**I**ntegrated **D**evelopment **E**nvironment
DSV	**D**elimiter-**S**eparated **V**alues
FPS	**F**rames **P**er **S**econd
TDD	**T**est **D**riven **D**evelopment
ROI	**R**egion(s) **O**f **I**nterest

Chapter 1

Introduction

This chapter aims to outline the motivation and context for the project. Throughout it the aims of the project, objectives, development time scale and the problem approach will be discussed. Hopefully empowering the reader of this paper with a greater comprehension of the overall project scope.

1.1 Context

Parallel computing is the core fundamental focus of this project, evaluating the performance increase obtainable through the use of parallel processes over traditional serial processes. Programming is generally taught using sequential paradigms (Robins *et al.*, 2003); due to this many concepts in Computer Science revolve and depend around the idea that instructions always execute incrementally. Frequently however a set of instructions or single instructions within the set have no dependency on one another. This is an situational example of a highly parallelisable problem, wherein each instruction can be simultaneously executed across multiple devices/cores resulting in a performance increase (Denning and Tichy, 1990). This project aims to take advantage of the performance increase that parallel computing can offer.

Motion estimation is determinant process returning motion vectors from one image to another; typically, both images are captured within a short period of time of each other or are structurally descriptive of one another i.e. two adjacent frames of a multi-frame image or video (Po and Ma, 1996). The motion vectors calculated in this process describe the transformation that has occurred between the two images. Motion estimation within filmed videos are generally less accurate as the video itself is a projection of three dimensional space onto a two dimensional plane, however within the scope of the project the data is captured using sound waves along a two dimensional plane. Motion vectors of an image can be used for multiple purposes such that of compression but within this project it is used to estimate BPM by calculating the average motion of the heart over time.

Due to the aforementioned caveat of low correlation algorithms for greater performance increase there are certain factors that went into the algorithm selection process for the project. The overall goal of either the serial or parallel solution is BPM estimation through motion estimation averages of multi-frame images. So multiple motion estimation algorithms were considered before finally using Full Exhaustive Block Matching with SAD error calculations for matching sections of the images between frames. The reason behind this is that motion vectors must be calculated for the entire frame to get an estimate of the overall motion while matching each block to its best suited neighbour. While Full Exhaustive search does this, it is reasonably slower than some other techniques (Liu *et al.*, 2003). This does however provide a platform to fully compare both serial and parallel implementations as the number of computations is large and constant. SAD is used to match each block of the image

and it is used because SAD calculations for blocks of the image can be calculated independently, making this process parallelisable. Analysing data as fast as possible opens up new possibilities motion estimation algorithms, possibly allowing real time analysis of data as it is captured. In the case of the heart and this project this would allow an estimation of BPM at the time of capture.

Using the average motion data calculated for each frame and the computation time required overall and between frames, evaluations will be drawn about the effectiveness and beneficial applications of parallel solutions to problems. The data that will be used in this project is a multi-frame apical four chamber echocardiogram. Echocardiography provides many different types of helpful information from the size and shape of the heart to the location of tissue damage and estimations of cardiac output. In this context it will be analysed for estimations of cardiac output or more specifically BPM. There are 10 different datasets that are used throughout the project to fully test the accuracy of the algorithms implemented and the throughput of data in the serial and parallel implementations of the algorithms which will be further used to evidence the evaluation.

1.2 Motivation

Image processing for medical imaging can be comprised of complex and computationally expensive algorithms, increasing the performance of these applications using specialised hardware allows for greater accuracy when analysing the data. If more data can be processed in a shorter time frame, more accurate results can be obtained by sampling over more observations or over a larger time series. Cardiac analysis using image processing techniques benefits directly from performance. Enabling faster diagnosis of heart conditions could possibly be vital in some situations, and could facilitate development of more advance cardiac algorithms for assessment of the heart. Parallelism can increase the performance of programs as aforementioned, and is currently used in a wide range of applications. One example currently where parallelism is exploited is in MRI reconstruction (Chang *et al.*, 2012).

Some traditional image processing techniques will not adopt to being executed in parallel easily when the algorithm has highly cohesive processes. CPUs are not as competent in areas such as computing floating point numbers or processing three dimensional graphics as GPUs which motivates the conversion of traditional algorithms from serial execution to parallel (Padua, 2011). Mathematically complex equations and algorithms can directly exploit the benefits heterogeneous systems can provide and this project hopes to encourage more work to be done outside of the sequential programming paradigm especially when frameworks exist that are more applicable to the problem scenario. Image processing is a great example of a project area where applications can exploit parallel frameworks due to the sheer computational complexity of some of the algorithms.

With the rise of GPU power in recent years an exploitable deployment medium for parallelisation of image processing exists. Axel and Sodickson (2014) summarise the importance of greater performance in cardiac assessment in their paper "The need for speed" where they state "Assessment of cardiac function is a cornerstone of the diagnosis and care of patients with heart disease. The use of various imaging methods for measuring cardiac function has become increasingly important in both clinical practice and research." and conclude with "The advent of parallel magnetic resonance imaging, in which arrays of radio frequency coils are used to gather

essential data simultaneously rather than in a traditional sequential order, has circumvented previous speed limits".

1.3 Project Aim

The aim of this project was to develop an application that utilised both sequential and parallel algorithms to effectively track motion within the heart. The motion data would then undergo a process to estimate the BPM of the heart. BPM is a unit that describes the number of contractions the heart takes over the period of a minute, BPM will vary between person to person and will change in different environments. The estimated BPM from the application will be used as a heuristic to determine the accuracy of the motion detection; with BPM values closer to the ground truth information being more desirable.

Originally the goal was to use the motion data for the detection and diagnosis of heart conditions, but due to the short time frame of the project, focus was shifted to more accurately determining the heart rate. A key component in the project is assessment of the performance gains achieved from the parallelisation of a naturally serial algorithm. This would determine and conclude the usefulness of real-time motion analysis. The effectiveness will be derived from comparisons against the conventional serial implementation and consequently the baseline results obtained from this.

1.3.1 Objectives

The original objectives of the project were altered to accommodate for realisations made throughout the development process; mainly regarding the time-frame required for each stage. The research was a fundamental part of redefining the objectives to be both more achievable and effective in a single application; otherwise incomplete features would be present subtracting from the purpose of the application. Below the updated objectives are listed.

1. Investigate environments and frameworks to facilitate the implementation of motion estimation techniques.
2. Analyse parallelisable motion estimation techniques so that a conclusive evaluation of performance can be obtained.
3. Implement, benchmark and visualise a serial implementation of the chosen motion estimation algorithm, to work as a baseline for performance comparisons.
4. Implement, benchmark and visualise a parallel implementation of the chosen motion estimation algorithm.
5. Evaluate the performance gain and effectiveness of the parallel implementation over the equivalent sequential implementation.

Chapter 2

Literature Review

The following chapter will outline current areas of research involving motion estimation and medical imaging within Computer Science. It will discuss several topics in relation to this thesis including image processing in parallel programming languages. All of the background research below is in direct correlation to the final artefact and contributed to its design and completion.

2.1 DICOM Standard

The data used in this project is stored within DICOM datasets with 10 different DICOM datasets being used overall. DICOM itself is a standard that attempts to provide interoperability between multiple modalities and medical imaging equipment such as CT, MRI and ultrasounds. It defines semantics of how the data should be communicated and exchanged between varying equipment (*Digital imaging and communications in medicine (DICOM)* 1998). The file format also stores all of the data required for this project. The raw pixel data has to be extracted or parsed from these files into more workable data representations such as a buffer of character values so that it can be universally used in any part of the finished application. The DICOM files store much more data about the image acquisition than just the raw pixel data such as overlay data which enables easy data cleaning prior to image processing (W.Dean *et al.*, 1997).

DICOM is almost universally used in contemporary medicine because most digital image acquisition devices in medicine currently will output a file in DICOM format and communicate through a DICOM network. They provide higher quality image representations such as 16 bit monochrome images allowing 65,280 more shades of grey than standard 8 bit JPEG images. This ensures that all of the smallest details captured from the acquisition devices are represented correctly, due to this using JPEG for processing the DICOM image sequences would waste most of the data (Pianykh, 2012). Consequently this will increase the accuracy when calculating the BPM estimate of the image as the image contains more information about its state and transformations over the sequenced frames.

2.1.1 Data Handling

The DICOM files store an excess of information that isn't required by this application so it was decided for portability that the application would be able to parse both MP4 video files and DICOM image sequences. This removes a requirement on having additional dependencies on DICOM reading libraries when building the implementation of this project. MP4 or MPEG-4 Part 14 is a digital multimedia container

format used to store video and audio (Kwon *et al.*, 2006). Motion estimation algorithms are typically applied to MP4 video files because they contain only the pixel information and are natively supported on most modern devices (Banas, 2016).

2.2 Parallel Computing

The following section will explain exactly what parallel computing is and the applications it has in relevance to this project and other fields of research.

2.2.1 What is Parallel Computing?

Parallel Computing is becoming an increasingly important paradigm in computation. Instructions and processes in parallel computing aren't carried out sequentially, they happen simultaneously which usually relates to a large performance boost (Almasi and Gottlieb, 1989). Solutions to problems are generally broken down into multiple smaller solutions which can be executed independent of one another. A combinative function of all the smaller solutions results will be the solution of the initial problem.

Frequency scaling is a technique used in computing that increases the performance of a system by increasing the frequency or clock rate of the CPU thus increasing the number of instructions that can be calculated per unit time. However the frequency of a CPU is directly related to how much power it will consume and the amount of heat it will dissipate (Rabaey *et al.*, 2002). Due to these constraints achieving more performance is now done by not increasing the speed of a single core but spreading the load over multiple cores instead. Applications now have to exploit the device parallelism to increase performance rather than relying on the sequential paradigm and nature of single core processors (Kumar *et al.*, 2008).

2.2.2 Parallel Algorithms

The speed of computation is cornerstone to almost every algorithm developed in a system, generally the faster it is the better. An algorithm is a self-contained sequence of instructions that calculates a function in a non-infinite amount of space and time. Parallelism allows us to reduce the time required to execute algorithms, Amdahl's law states that for applications that are 95% parallelisable a theoretical speed up of twenty times the original speed can be achieved. This speed up will always be limited by parts of the application that do not benefit from extra resources such as memory or the number of cores the system has (Rodgers, 1985).

Vast amounts of research have found success in the parallelisation of traditionally sequential implementations of algorithms.Pacheco (2011) lists climate modelling, protein folding, drug discovery, energy research and data analysis as areas of research that are directly in need of faster algorithms. He also goes on to state that the reasoning for parallel development of algorithms is generally descriptive of the limitations of current available processing power. Since only limited success has been achieved in automatic translation of serial programs into parallel programs has been realised the development of parallel applications to fully utilise available resources is even more important (Shen and Lipasti, 2013).

2.2.3 Parallel Programming

The most challenging part of this project was programming in parallel, the fundamental concepts in contrast with sequential programming are largely different. When programming in parallel there's more of a focus on generality, problems can be expressed as multiple separate problems and can be broken down in many different ways. The programs built in a parallel framework are generally executed in a serial language which is responsible for the execution and data streams to and from the parallel programs. The parallel framework or API used to facilitate the development in this project was OpenCL.

OpenCL is an open source standard maintained and documented by the Khronos Group. It is a standard used in programming with heterogeneous systems (Khronos-Group, 2011). Heterogeneous computing refers to systems that are comprised of multiple different processors and cores. Generally heterogeneous systems are comprised of specialised hardware designed to increase energy efficiency or performance. The specialised hardware enables the system to be able to handle a wider variety of tasks (Khokhar *et al.*, 1993). These systems are also much more common currently with the advent of desktop gaming wherein a dedicated graphics card is frequently required for higher end graphically demanding games. The OpenCL API can fully take advantage of these desktop systems in delivering fast and reliable performance, it also has very comprehensive documentation available on-line for each version of the OpenCL standard. It's used by a lot of larger companies in production products and it's enabled this project to develop portable code that can be run on many different machines (Xu, 2008).

Many factors contributed to the difficulty of programming in parallel, one of the factors was having to account for the modification of the data and access to the data randomly. Due to the nature of parallel computing accessing data will always be random access unless specified or forced otherwise. Assumptions on the time frame that modification to the data will occur can't be made which disrupts the general way of thinking when programming in serial. Race conditions are events or unexpected modifications to the data that can occur when two operations are performed on the data at the same time (Netzer and Miller, 1992). An example of this is if two threads are calculating the summation of adjacent values in an array left to right the order of access to the data will directly affect the end result. In this case particularly, sequencing is important or the implementation of preventative features to ensure the data is calculated correctly.

When developing a system with OpenCL the API has abstracted away from specific hardware vendors making it a neutrally supported framework. All code developed is close to the implementation of the hardware across many devices allowing portability (Choudhary *et al.*, 2010). The execution model when programming in this framework can be broken down into two main concepts. Firstly kernels, a kernel is similar to functions in languages such as C and C++. They're small pieces of code that take inputs and can write to output buffers, generally there purpose is to solve small simple tasks. The second concept is the host program, this would be in the case of this project the C++ code base. The host program is important for many things and it's responsible for the invocation and management of the OpenCL kernels. By invoking one or more kernels, the host program can solve tasks. The kernels are initialised with data and given access to read and or write to the data. When kernels are executed they are placed into a vector or matrix space known as the index space, where each element is one instance of a kernel (Trigkas, 2014).

2.2.4 Parallel applications in Motion Estimation

Video encoding and decoding are two reasons why motion estimation and block matching algorithms are in development for the GPU currently. Graphical processing units were originally developed to accelerate the performance of three dimensional video processing in games. Due to this they are now more common place and can be used to accelerate the calculation of arithmetic heavy algorithms through a large number of decoupled processing cores (Hwu and Kirk, 2009). Cheung *et al.* (2010) points out that to fully take advantage of the GPU the algorithms need to utilise as much data parallelism as possible, by distributing data across all available cores of the GPU to keep the processing cores working for as long as possible. Some algorithms however may struggle to do this, which is generally due to inter dependencies within the data. This is demonstrated many times in previous work done in this field especially in work by Kung *et al.* (2008), Cheung *et al.* (2009) and Jin and Lee (2006).

In a paper by Massanes *et al.* (2011) they found that using certain optimisation strategies across multiple GPU cards they could ascertain real-time motion estimation of 720p images which outperformed their CPU benchmarks by many orders of magnitude. They also compared their GPU implementation to two optimised CPU implementations and plotted the results. The two CPU techniques they benchmarked against the GPU were Pyramidal Lucas Kanade optical flow algorithm (Bouguet, 2001) and SmpUHex (Yi *et al.*, 2005). They concluded that with the GPU implementation still being 1.75 times faster than the two optimised CPU methods that a reimplementation of the optimised methods on the GPU would decrease the computation time even more. Baglietto *et al.* (1995) also considers a Full Search block matching algorithm similar to this project and work by Massanes *et al.* (2011).

A full search is usually more accurate than other block matching algorithms as it considers a comparative result against more sections of the image for every section of the image. Due to this however it is very computationally expensive which is why other block matching algorithms are generally used in parallel and serial applications to estimate motion with a much lower computation cost. Two examples of these algorithms are proposed in papers by Chen *et al.* (1991) and Ghanbari (1990) respectively. Since accuracy of the motion estimation is imperative in this project to gauge BPM the full search block matching algorithm was chosen over these algorithms however. Also the full search block matching algorithm can be represented with a very simple control flow which can be exploited to allow for high levels of data parallelism. This exploitation comes from the fact that the calculation of each block in each image is independent of other blocks in the same image and the similarity error of each block is independent of the insularity error for other blocks (Saha *et al.*, 2011).

2.3 Image Processing with OpenCV

OpenCV (2017a) is a library used in this project that facilitates the serial full search block matching algorithm as well as facilitating the visualisation of the results and real-time plotting of the motion estimation data overlaid on the original. OpenCV is an open source library that is used for software relating to computer vision and machine learning. It's original goal was to provide an standard interface for computer vision applications. Since it's open source all of the code for functions and classes is available on-line which aids the development of personal code. It's also licensed

under the BSD licence which means anyone can modify the code for commercial or personal projects.

One of the many benefits of this open source library is that the documentation and community on-line is huge and many of the problems encountered over this project were documented very well so were easier to debug and resolve. It's also written in C and C++ so it can run on Linux, Windows or OSX which was a focus of this project; to be runnable on most operating systems. It was designed for computational efficiency and contains over 500 popular algorithms already optimised so it can provide good benchmarks for CPU execution compared to parallel GPU execution (Bradski and Kaehler, 2008).

2.4 Motion Estimation

Given two images such as two adjacent frames in a video sequence, motion estimation is a way of determining and calculating the transformation it has undergone from the first image to the second one. As discussed in the introduction of this thesis it is used in this project to estimate the BPM of the cardiac image sequences. Calculating motion estimation can be done in a variety of different techniques. One of the techniques used is phase correlation which is a type of image registration that estimates the relative translative offset between two images (Elmoataz *et al.*, 2008). The resultant image of this method is a correlation image where peak intensities represent the locations that both images match the best.

Similar methods estimating motion using a correlation heuristic between two images have been successfully evaluated and implemented in current research. One example of this is in the paper "Motion Estimation in Ultrasound Images Using Time Domain Cross Correlation With Prior Estimates" by Zahiri-Azar and Salcudean (2006). In this paper the real-time applications of the method they use are discussed and concluded. They conclude that their method of using time domain cross correlation with prior estimates can be used to estimate motion in cardiac images in almost real-time. A different method is used in this project to estimate the motion of the heart, the method used is full search block matching or FSBM.

2.4.1 Full Search Block Matching

In contrast to the motion estimation method mentioned above Block Matching is also a viable alternative, and is used in this project. Block matching is a method of matching blocks of an image to another image; blocks are determined by a grid on the image i.e. a 10 by 10 grid on a square 200 pixel image would create 400 blocks to match between two images. In the context of this image the blocks are matched to adjacent frames to determine the motion that has occurred between them (Barjatya, 2004). Algorithms are generally comprised of splitting an image into blocks either by a generic full grid division or by calculating regions of the image where motion should be tracked then matching these blocks in a search window around the blocks in the next or previous image in the video sequence (Seferidis and Ghanbari, 1993). Motion vectors are then calculated as the displacement of the original blocks to the best matched block in the second image. The average of all the calculated vectors then gives an estimate of the overall motion in an image (Zhu and Ma, 1997).

Zhang *et al.* (2012) show that a calculating motion estimation using block matching algorithms can have a 100 to 150 times speed-up compared to a single core CPU implementation. In this paper they use SAD calculations to compare each block in

the first image to the second, however dissimilarly to this project they first calculate the sum of absolute differences in parallel and then use a separate kernel in parallel to compare all of the SAD calculations and select the best matching one. They also discuss how FSBM algorithms aren't a viable choice in real-time applications due to how computationally expensive they are. Metkar and Talbar (2013) also discuss this point and mention that the FSBM algorithm for motion estimation are both the simplest and most accurate way of calculating motion as long as the search window in the second image encompasses a reasonable area. This is further justification of a FSBM algorithm for motion estimation parallel implementation in this project, to calculate the most accurate BPM estimation.

2.4.2 Cost Functions

The cost function is something that is calculated when comparing two blocks of an image or images. Dependant on the method it will return a value representative of some similarity criterion such as intensity. The aim when comparing the cost functions in a search window is to minimize the cost; the lower the cost associated with the images the higher the similarity between them (Barjatya, 2004). A common example of a cost function used in motion estimation is mean absolute difference (Tankariya *et al.*, 2011). MAD is calculated by taking the element wise difference of two matrices or vectors, calculating the sum and dividing the value by the number of elements squared. The equation for MAD is given below where N is the number of elements in a block and A_{ij} and B_{ij} are the two blocks being compared.

$$\text{MAD} = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |A_{ij} - B_{ij}| \tag{2.1}$$

When calculating the cost function in parallel it would be beneficial to-do as few operations as possible to increase performance and possibly avoid multiplications and divisions. SAD is a measure of similarity that doesn't require any multiplication or division. It is calculated by taking summation of the absolute difference of each element in the blocks from the first image and second image respectively. This cost function is ideal for parallelisation due to these factors as well as having a very low computational cost (Mehta *et al.*, 2010). The equation for the SAD of two image blocks is given below where N is the number of elements in a block and A_{ij} and B_{ij} are the two blocks being compared.

$$\text{SAD} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |A_{ij} - B_{ij}| \tag{2.2}$$

2.5 Cardiac Anatomy and Physiology

The data set used in this report is comprised of 10 different ultrasound multi-frame images captured at 50Hz. The ultrasounds show the apical four chamber view which clearly shows the left ventricle, left atrium, right ventricle and right atrium as well as the mitral value and tricuspid valve (Allen, 1999) . To estimate BPM the frequency of a single cardiac cycle needs to be calculated from the average motion vector signal of the frames. The frequency of this signal will represent the number of frames it takes for one cardiac cycle to complete. Using the frequency and the capture rate it can be converted to heart rate using the formula below, calculating the number of

frames that were captured over a minute and dividing it by the number of frames in one cardiac cycle.

$$\text{BPM} = \frac{\text{Capture Rate} \times 60}{\text{Cardiac Cycle Frequency}} \tag{2.3}$$

The cardiac cycle is a chain of events that happen when the heart beats. The cardiac cycle is broken down into two periods and four phases. The two periods are systole and diastole where the ventricles contract and relax in each period respectively. When the ventricles relax the heart fills with blood and then when they contract blood is pumped into the arteries. One cycle is complete when ventricles have been filled with blood and then pumped back out to the lungs and the rest of the body (Martini, 2007). The average motion of the data should reflect and repeat this pattern over time, which can be extracted and it's frequency can be calculated to estimate the heart rate using the above formula.

2.5.1 Fast Fourier Transform

Calculating the period of one cardiac cycle requires splitting the average motion vector data from its time domain into a representation in the frequency domain. In a paper by Lam and Kuno (2015) they demonstrate a technique for extracting the frequency domain of a signal. By calculating the power spectrum of their samples over time, they estimate the power of the different frequencies of their time series. Using the time series power spectrum they select the frequency for estimating heart rate as the peak frequency and justify it by stating that the dominant frequency is the most probable selection due to their data set having a single factor primarily contributing to variation in their data. Similar to using only motion data of the heart in this project they're estimating heart rate by using a single measurement component.

Fast Fourier Transform algorithms are a way of calculating the discrete Fourier transform of a time series. The DFT is obtained by decomposing a time series into components of varied frequencies, FFT is used to compute the same result as DFT due to its complexity being $O(n \log n)$ rather than $O(n^2)$ (Johnson, 2010). Yang *et al.* (2015) use a FFT algorithm to compute a power spectrum of their data and calculate BPM as the peak frequency of the power spectrum. This paper is where the BPM estimation equation in 2.3 was derived from. The contrasting difference being their data is over time whereas the data in the project is over a number of frames so a conversion to seconds using the capture rate of the ultrasound scanner is made. Although both papers above demonstrate successful BPM estimation from videos using a similar method in this project proved to be effective.

Chapter 3

Methodology

The generalised goal of any project is to efficiently meet its aim and objectives. The SDLC describes the processes between the start of a project to its completion as seen in 3.1. Intermittent stages of the SDLC are comprised of areas such as development, implementation, delivery and maintenance (Rao *et al.*, 2011). Maintaining an efficient work flow throughout the duration of the project can be difficult. Methodologies provide a framework of different processes that can help achieve this goal in all areas throughout the entire process of the software development life cycle. They help maintain consistency and quality within the project by advocating standards or procedures that should be followed (Khan *et al.*, 2011).

3.1 Project Management

Selecting an appropriate Project Management methodology is of paramount importance when starting out in a project, as it can lead to a multitude of difficulties further on in the process. Charvat (2003) discusses potential problems of not choosing an appropriate project management methodology or choosing an appropriate methodology but not applying it correctly in the context of the project. Some examples of the potential problems he mentions are schedule and cost, when using an unsuited methodology they can easily overrun the initial allotted time or budget allocation. This could be due to many different factors, one example being that some methodologies can require a large amount of dedicated hours contributing to processes or project templates. Consequently, this can make the daily execution of the methodology more difficult (Kerzner, 2004). To minimise this at the start of a project an effective plan should be outlined, which is achieved by minimising scope changes with realistic and achievable goals. Progress should also be determined by a periodical review of the current project state and its relevance to any current milestones (Kerzner, 2013). This is evidenced in the time frame section below in the addition of periodic milestones throughout the project plan.

Project Management methodologies are usually structured to facilitate effective team communication and involvement, normally with two or more people working towards achieving the project aim through incremental steps and collaboration. Project Management methodologies have been deployed in countless organisations and have been found to be effective in most cases because they can effectively utilise all available resources (Munns and Bjeirmi, 1996). All of the work in this project that has been completed to date is by an individual, rendering most fundamental collaborative concepts of Project Management techniques incompatible with the structure of this project. This was the most important factor to consider in methodology selection as well as considering a methodology that was flexible to requirements change. Having the ability to accommodate for fluctuations of the requirements was

necessary due to the short project duration and initial lack of research in the areas involved.

Agile Project Management focuses on delivering the product in incremental steps, which allows the product to change over time. This enables changes in the requirements to fluctuate as much as they want as the product will adapt to fit current requirements (Westland, 2007). Due to the nature of this project it's fully expected to grow and change as features and requirements are implemented. As the feature list grows and consequently the product comes closer to completion the projects limitations, purpose, user interaction and capabilities will become more apparent which will inevitably drive project change of some sort (Karlesky and Voord, 2008). The "Agile Manifesto" by Beck *et al.* (2001) describes how the Agile framework shifts the focus away from process based approaches to weighted tasks from a pre-defined plan. Process based approaches commonly will weight all requirements equally consequently making the process rigid and slow, however Agile processes allow for prioritisation and progression of important project components first Elliott (2008).

3.1.1 Risk Management

In any project it's possible that unanticipated circumstances or situations can occur that can impact a projects ability to meet its cost, testing or time frame objectives. These can be described as risks that are hazardous to the intended project execution. A Risk Matrix is a structured approach to identify these risks and assess their criticality and possible impact to the program and ways to mitigate or avoid them entirely (Garvey and Lansdowne, 1998). Once risks are identified their impact, probability, severity, and a mitigation plan are outlined subsequently defining a Risk Matrix. Figure A.1 shows a Risk Matrix initially created at the start of this project, the sections in the figure describe multiple areas of the Risk Matrix. The sections it outlines are what the risk is, a contextual description of the risk, a quotient metric determining the impact of the risk based on severity and probability and a plan of mitigating the risk in the event it happens. Engert and Lansdowne (1999) clearly describe the motivation of a Risk Matrix, "This structured process was created to help programs identify, prioritize, and manage key risks associated with meeting program objectives.". With the advent of the Risk Matrix in figure A.1 it facilitates the management of this project by reducing factors that could impact its health.

3.1.2 Time Management

Due to the frequency in which requirements can change in this project, planning ahead will be an inaccurate way to monitor progress throughout the project. Maylor (2001) states that a plan when developing software doesn't guarantee that a project methodology or processes are being followed but it does mean there is an initial plan of how the project should execute over time. Since this project follows the RAD methodology described in 3.3 a Gantt chart will portray elements and milestones that will most likely change before they are reached thus increasing the work load of having to alter the Gantt chart every time a change occurs (Karlesky and Voord, 2008). The Gantt chart shown in figure A.2 shows a portrayal of how the project was meant to evolve over time to complete initial objectives. As mentioned above it is only used as a brief overview of the management that will happen over time and will be quickly dismissed as the project evolves and changes. Due to a lack of knowledge in the area that has been gained progressively over the course of the project, which

is subsequently reinforces the decision of using an Agile methodology, some of the elements in the Gantt chart are ambitious in their time frame.

3.2 Software Development

The software for this project has certain determinant characteristics that will help choose an optimal methodology to use. The software will be heavy on code due to the requirements that have been initially gathered. The project consists of only one team member which will code and design the software so a methodology that is comprised of multiple different processes will hinder the ability to write the amount of code necessary to complete all of the deliverables. Since the software will be developed in a short time frame the methodology should allow for the requirements to change to adapt to any circumstances not initially planned for. The software itself will be programmed in C, C++ and MATLAB for evaluation which should work across the three major operating systems, Windows, Linux and OSX. The software will also have a dependency on the heterogeneous system having both a CPU and a GPU since the code will utilise the Image Processing framework of OpenCL. Due to these two constraints, considerable testing will have to be undertook at the completion of requirements in multiple different environments. Agile methodologies as mentioned above accommodate for all of these constraints allowing more freedom to iteratively develop software through the SDLC shown in 3.1.

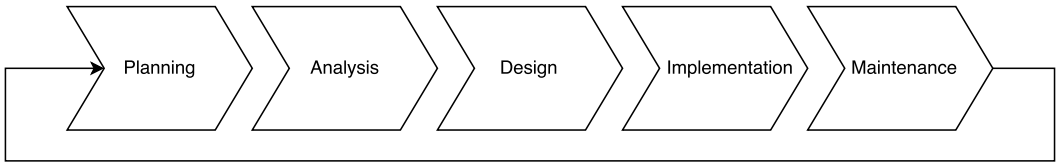


FIGURE 3.1: Software Development Life Cycle

3.2.1 Tools

This section aims to layout and justify the main components that facilitated the software development within this project as well as discuss their context to the final system. Some elements were imperative to the success of the project and as described in section 3.3 they increase productivity and utilise effective resource management by taking advantage of the power modern utilities and services provide (Association, 2013).

3.2.1.1 C++

C++ originally started as an extension to C that introduced classes and object-orientated principles. It's a multi-paradigm general purpose language that's used across many different domains from desktop applications to enterprise server back ends (Stroustrup, 1986). The language is comprised of many common programming features such as exception handling, type checking, in-line functions, default arguments, operator overloading and templates. C++ is a compiled language that has solid support on most platforms, it's a reliable cross platform solution that has freely available compilers for most architectures. The philosophy and evolution of C++ since first being developed by Stroustrup (1986) has been to allow programmers to develop their own style and be able to use it within the language, allow programmers to manually

tell the compiler what their intentions are, it will not have a hierarchy of languages that are interpreted in C++ apart from being compiled into assembly, enable high performance when interacting and hardware and low level management of memory (Stroustrup, 2007).

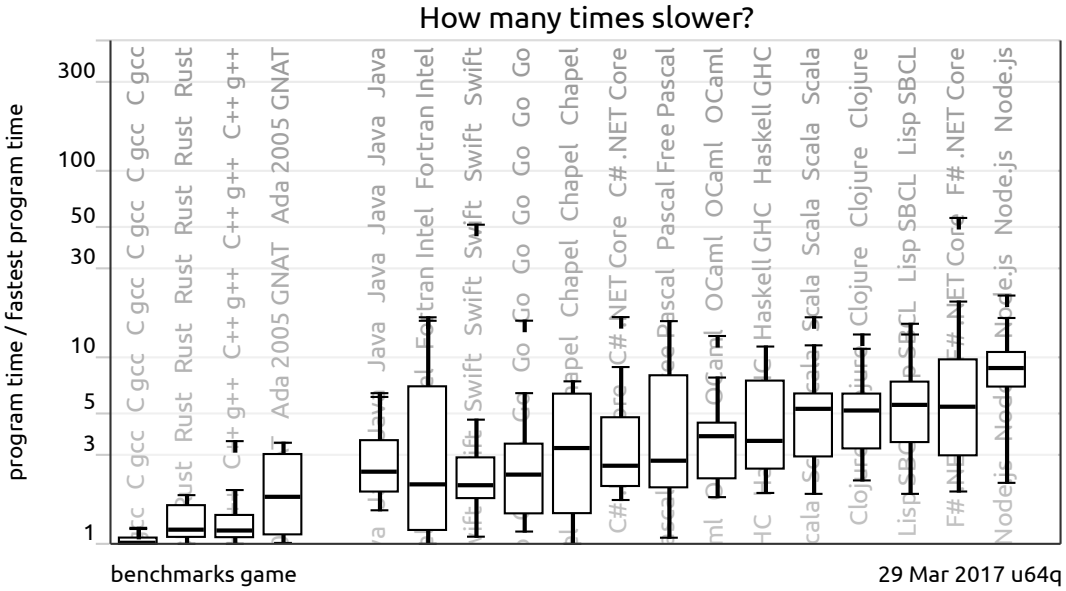


FIGURE 3.2: Programming Language Comparisons against the Fastest Benchmarked Implementation (Fulgham and Gouy, 2009)

Since C++ is well supported, many popular libraries are either implemented in the language or have wrappers to interface with their functionality. As a language it offers many different types of data structures that allow a great amount of flexibility in application design and implementation. Cross compilation was important in the duration of this project because multiple different machines were used in the development and it was a focus to conform with OpenCL standards of parallel code working on a variety of different architecture and vendor GPUs. Being able to compile on large number of generic household heterogeneous systems was important in the scope of the project which is facilitated by the C++ language. Both OpenCL and OpenCV are commonly used in C++ also, so documentation and the on-line community made debugging easier. C++ is also considered one of the best performing languages, which makes it suitable for applications in performance dependant domains such as image processing. Figure 3.2 shows a performance comparison of different languages and it shows that C++ is the second fastest in terms of median computation time (Fulgham and Gouy, 2009). In this project it's especially important as performance will play a large part in the evaluation and comparison of two programming paradigms.

3.2.1.2 C99

The language OpenCL kernels are programmed in is essentially an extension to the C99 standard with some restrictions. Syntactically it can be programmed in the same as C, figure 3.3 shows some valid OpenCL C code to add two integers. Some main points of the restrictions over the C99 standard include pointers to pointers cannot be passed as arguments to the OpenCL kernel, C99 headers cannot be included in the source, variadic functions and arguments aren't supported, the double precision

floating point data type has limited support so may not be supported on a device being used and recursion isn't supported. OpenCL C has some built in functions that can be accessed since C99 headers cannot be included. These functions encompass common ones from the "math.h" package in C such as "exp()", "log()" and "cos()" also some geometry and work item functions are also included such as "dot()" and "get_group_id()" respectively (Munshi *et al.*, 2011).

```
1 int AddTwoIntegers(int a, int b) {
2         return a + b;
3 }
```

FIGURE 3.3: Valid OpenCL C Syntax for the Sum of two Integers

C++ and C have always shared similar syntax to one another, mostly due to C++ originally being an extended version of C (Stroustrup, 1986). This overlap directly results in an increase in the productivity when switching between programming OpenCL kernels and programming on the sequential host in C++ because of the syntactic similarities between the C and C++ standards used. This also makes learning the OpenCL C language easier because of how much its fundamentals are shared with C++ fundamentals. An example of the shared similarity can be seen in figure 3.3 where the C code presented would also be valid in a C++ application. The Kernel code once written is compiled in the host program using a call to the OpenCL C99 (ISO/IEC 9899:1999) runtime compiler (Banger and Bhattacharyya, 2013). Any compile time errors can be reported and printed to the host making debugging issues in the code more efficient.

3.2.1.3 MATLAB

MATLAB is a multi-paradigm numeric programming language developed by MathWorks (2017). Contrasting to many other languages it deals directly with matrix arithmetic and manipulations, has in-built plotting functions that handle many types of data and allows the creation of GUIs for adding interactivity to applications. MATLAB is commonly used in a range of applications from modelling and simulations to data processing and plotting data (Gilat, 2014). Programs in MATLAB are generally written in scripts which are groups of commands stored in a file that can be interpreted and ran by the MATLAB software. One of the best features MATLAB is to offer is the number of comprehensive function packages it ships with or that can be downloaded and installed in the form of "Toolboxes" (Attaway, 2016). Higham and Higham (2016) lists three distinctive features of MATLAB that distinguish it from other computing environments; automatic storage allocation, variable argument lists and complex arrays and arithmetic.

Automatic storage allocation is a feature in MATLAB that essentially removes the need to preallocate variables, it will dynamically replace any existing variables in memory with the newly allocated values. Preallocating data is sometimes desirable for performance and can be done also. This feature makes resizing and manipulating data easier as less considerations need to be made for it's storage. Variable argument lists are essentially self explanatory, the enable user defined and software defined functions to accept any number of variables. This allows modification of multiple different data types in different ways without redefinition of multiple different functions (Chapman, 2016). This creates an abstraction from data specific functions and enables the programmer to define operations in a logical way for a range of data types. Higham and Higham (2016) concludes by describing the final

defining feature of complex arrays and arithmetic by stating that the fundamental data type of MATLAB is a multidimensional array of complex numbers comprised of real and imaginary parts that can represent any number. Compared to C++ where different data types are necessary to store real and complex numeric data.

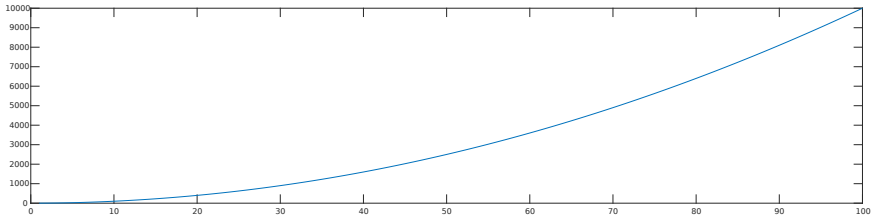


FIGURE 3.4: Result of MATLAB function "plot(1:100, (1:100).^2)"

MATLAB isn't used in the core of the application development process. However it's used for debugging purposes to cross reference values obtained in the C++ application. MATLAB is also used in the testing phases to fully analyse the results obtained and plot them on a graph as well. It is also later used in the Cut-over phase described in section 3.3 to validate results and obtain BPM estimates. MATLAB is used for this as it makes the process of plotting data effortless and reinforces points made when evaluating the data. Due to time constraints of the project development, MATLAB helped by providing an easy way to analyse the data without having to write unnecessarily long code files or manually stepping through the results. A visualisation of function and plotting simplicity in MATLAB is given in figure 3.4, where arguments to the plot function x and y are a sequence from 1 to 100 and x^2 respectively.

3.2.1.4 Version Control System

This project comprises mostly of C++, C and MATLAB code and storing it all in a reliable and secure location is imperative. Managing all of the code files can be confusing, overwhelming and can result in a loss of work fairly easily. The C++ code is logically broken into objects, name-spaces and functions and when working on a particular class it would be easy to overwrite functionality and compatibility between classes especially when making many changes in a short time frame. In this example an old version of the file would be helpful in fixing the compatibility issues. This is one contextualised benefit of version control systems. Version control systems help manage modifications made to documents, code bases, data and most file types over time. The modifications are usually sorted and organised by the creation of the revision of the files and can be easily reverted to or viewed.

This project has utilised a version control system throughout to track the changes for a few reasons. One reason is that the changes are a form of documentation that clearly shows the evolution of the project code base over the course of this project. Secondly because all of the changes are tracked, reverting to earlier iterations of the project is trivial. The version control system used in this project is Git (2017) offered via a web based repository service GitHub (2017). GitHub allows anyone to host files on-line either in a public repository or a private one. A repository is a storage location where different files can be stored and saved. GitHub also has some added features over using standard Git such as an on-line source viewer that can step through iterations and clearly visualise the changes made between iterations. It

was reported in 2014 that it had become the largest host of source code in the world (Gousios *et al.*, 2014).

3.2.1.5 IDE

The IDE that was used in this project was dependant on the operating system. Since multiple operating systems were used throughout two editors facilitated the development by providing an array of advanced features designed to increase productivity. The two editors used were CLion (JetBrains, 2017) and Visual Studio 2017 (Microsoft, 2017). CLion is a cross-platform IDE for C and C++ development, however compatibility with some libraries and the supported compilers of CLion made it difficult to use this editor on Windows. Visual Studio 2017 was used on Windows instead of CLion because of this issue. Modern IDEs have features built in to increase productivity and automate common tasks. Code completion is one of the features, the editors suggest code snippets while typing which makes typing full functions or variables only require a small proportion of the input it would require in a plain text editor.

CLion and Visual Studio 2017 both have syntax highlighting for most programming languages, separating common code concepts such as function names, variables, types and comments into colour groups that can more easily be distinguished by eye. They also facilitate testing and debugging by providing error messages, code analysis and runtime breakpoints that can halt the application at specific point to analyse the state on the program. Both editors directly link in with CMake for directly compiling the project inside the IDE, and integrating with the IDEs debugging environment. They both offer many more features than mentioned but overall the point is that they promote faster software development by focussing on the software development over repetitive arduous aspects of software engineering. RAD described in section 3.3 benefits from the use of modern development tools to accelerate development and both IDEs contributed to increasing work flow efficiency greatly.

3.2.1.6 CMake

CMake is a cross platform open source software that automates build processes of code bases. It's compiler independent which enables most operating systems to use their native compilers and tools to build software with CMake. This simplifies compiling cross platform applications significantly because it reduces the knowledge required of how multiple different systems and build tools work. It utilises native build environments of operating systems such as Make (FreeSoftwareFoundation, 2017), XCode (Apple, 2017) and Visual Studio (Microsoft, 2017). The only dependencies CMake has is a valid C++ compiler on any operating system. It's used in this project because it provides a reliable method of building, testing and packaging the C++ code independent of the development environment. Hoffman *et al.* (2009) summarise the effectiveness of using CMake in a similar project, "Given the heterogeneous nature of software development across Linux, Windows, Mac, and HPC platforms, the build system should not be a proprietary solution for one platform. Maintaining a separate build system for each of the supported platforms, although common in many projects, is brittle and does not encourage collaboration and code reuse.".

In the development stages of this project only one set of files needed to be written in the CMake language to enable building on multiple platforms instead of writing

a separate build file for each system that would need to be updated regularly as the project structure and intent changed. It translates a high-level build language description of the build process named "CMakeLists.txt" into lower level files supported by tools such as Make (Smith, 2011; FreeSoftwareFoundation, 2017). This increased productivity significantly and was one less process to complete when implementing and testing the system at multiple stages throughout. Initially the learning curve of how to configure and use CMake took a little longer than expected but it saved a lot of time later on in the project as it never needed much maintenance. It allows the project to have minimal duplication, it can modify and change many aspects of the code base to get it to compile successfully including linking with libraries (Smith, 2011). An example of this is that files can be shared between multiple different targets, reducing the time taken to build the software by placing the files in all target directories. Instead you can just include them in the CMake build files "CMakeLists.txt" (Joshi *et al.*, 2016).

3.3 Rapid Application Development

Rapid Application Development is an Agile methodology originally developed by Martin (1991), it focusses on fast iterative development of a system. This is an applicable methodology in this project due to a few factors. It is designed to take advantage of the power that current development tools and software can provide, it enables organisations to develop "strategically important systems faster while maintaining quality" (Association, 2013). RAD is flexible about requirements change which are introduced through incremental collaboration throughout the project. RAD has been found to work better with small scale projects and have teams of no more than ten people which is well suited to the constraints of this project (Berger *et al.*, 2004) (Mackay *et al.*, 2000). Due to the small size of the teams within RAD, development requirements are generally introduced periodically with focus groups or weekly meetings (Beynon-Davies *et al.*, 1999). Since this project consists of a single researcher the requirements are only introduced through either new transparency in the progression of the project or from collaboration with the project supervisor.

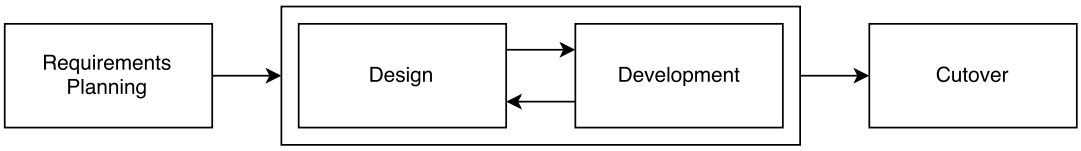


FIGURE 3.5: Rapid Application Development Phases

RAD allocates more resources to implementation over planning, and doesn't require a rigorous specification prior to the development of the project. This is beneficial to project management aspects since the initial requirements were defined with minimal current knowledge in their respective areas. Unanticipated problems can arise from poor specification and requirements gathering, RAD processes allow for greater control and adaptability when problems arise decreasing wasted resources on fully redefining the development and/or design of a product. Consequently benefiting the development and management of the project with a positive response to gained knowledge over the course of the project. Martin (1991) broke RAD down into four phases, some of them containing or resembling typical approaches from the SDLC processes as seen in figure 3.5.

The first phase of RAD is requirements planning which encompasses similar processes as the analysis and planning stages in the SDLC shown in figure 3.1. In this stage initial requirements are defined by all members of the team which in the scope of this project is the author and project supervisor. The scope of the project and any constraints/risks are defined also before this stage is concluded. The following stage is user design, in this stage the client is involved in the creation of models that represent the system processes, inputs and outputs. The client in this scope is the author of this paper and the user design phase allows prototyping via models to facilitate faster development and early testing of class relationships.

This process of user design is continuous until the models are representative of the requirements and then approved. After the design phase the development of the design starts, however if design changes occur at any point during this phase the changes are integrated in the current development cycle. The development phase is respective of the development phase in the SDLC. Testing of the system occurs at this stage, if the tests pass the process flow either moves back to the user design or goes to the final Cut-over phase. In the last phase final tests are made and evaluations are drawn on the project. In industry this phase would usually be the deployment and integration of the product. A benefit of this model is that documentation is created in the development phases, which creates a better groundwork for requirement modifications in the future or for debugging purposes.

Chapter 4

Requirements Planning

This chapter aims to describe and justify the requirements the software was developed from and how they were derived from the objectives. The objectives will be broken-down into requirements that will guide the software development. An attempt will be made to maximise the interdependency of the objectives. This is to facilitate the software development of the project. The lower the cohesion is between objectives, the more they can be split into multiple different RAD cycles. Which will allow components of the project to be designed and developed in parallel over multiple iterations encompassed of similar phases to the phases shown in figure 3.5.

4.1 Requirement Elicitation

Initially Dr. Massoud ZOLGHARNI proposed the project as "Parallel Computing for Cardiac Image Processing" which entailed using a variety of image processing and parallel computing techniques to develop fast computer programs for object tracking in human heart images. This was further refined in collaboration with Dr. Grzegorz CIELNIAK to develop a fast program using parallel computing techniques to estimate heart rate and draw evaluations on the performance gained over an equivalent serial application. Over the course of the project the author has acted as the client because of the self-defined nature of the project and the initial strong specification that the objectives formed. Modifications to the project objectives sourced from realisations made from greater knowledge of the subject areas and from collaboration with both Dr. Grzegorz CIELNIAK and Dr. Massoud ZOLGHARNI to maximise content in the final deliverable. This projects objectives are outlined in section 1.3.1, minimal modifications have been made to them over the project duration. The changes mostly reflect more logical objective ordering and additional analytical steps that reduced the risk of time slippage.

Christel and Kang (1992) outline three major areas that can impact effective requirement elicitation; Problems of scope, understanding and volatility. Respectively they describe the challenges of defining requirements from ill-defined system boundaries or vague client objectives, a communication barrier between what the client needs and what they can describe and unexpected requirement modifications over time. These three areas were accommodated for in the prior processes to the requirements elicitation to maximise resources available in other stages of the project life cycle. At this stage the objectives are clearly defined and bounded, the client is the author of this project removing limitations in the communication of objectives and requirement volatility has been accommodated for with utilisation of appropriate project and software methodologies and processes. This criteria conformity allowed for easy transitive requirement gathering from the objectives. The objectives were broken down into software requirements and subsequently independent

deliverables to modularise the development processes and are listed in sections 4.2 and 4.2.1.

4.2 Project Requirements

The objectives of this project outlined in section 1.3.1 describe what the scope of the project is and the overall aim. Whereas the requirements listed below are a broken down subset of the objectives that outline the minimal capabilities required of the system to achieve the project aim and objectives in the project time frame. Furthermore section 4.2.1 breaks the requirements down even further into features that will be developed incrementally from the requirements and subsequently help achieve the project objectives. The software development stages of this project will aim to implement the deliverables and meet the requirements set. Managing the project this way has allowed for a greater abstraction between the goal and the software development phases. This in turn has facilitated the design and development of the system due to each requirement and deliverable being almost independent of one another.

1. Software must be implemented in an appropriate environment using appropriate libraries.
2. MPEG-4 Part 14 video files and DICOM medical image files must be both parse able and interchangeable in the application.
3. The chosen motion estimation algorithm for cardiac assessment in the serial implementation should be selected based on the level of data parallelism that can be exploited to ensure the minimal project slippage due to algorithm development.
4. Raw pixel data from the data sets should analysed, benchmarked and the resultant motion data outputted frame-by-frame so that the application can work from constant data streams.
5. Serial and Parallel implementations should be independent applications that log their data separately to indexed files.
6. Both applications should have the ability to show processed data in real time and plot motion data on a real time graph for applications with a continuous stream.
7. All evaluations and tests on the motion data should be done externally to the applications, only raw data should be written from the file writer and only motion vectors should be calculated in the applications.
8. Motion vector data from the applications must be evaluated and analysed.

4.2.1 Project Deliverables

As aforementioned regarding the development phases of the project life cycle the requirements have been broken down into iterative development stages below that are descriptive of the software construction phases that occurred.

1. Configure the development environment and libraries such as OpenCV, OpenCL and the CMake build system.

2. Creation of interchangeable raw pixel capture classes for MPEG-4 Part 14 video files and DICOM medical image files.
3. Implement a motion estimation algorithm on the CPU.
4. Creation of a plotting class to visualise the motion vectors and plot the calculated motion vectors.
5. Creation of a file writer to export the motion vectors to files for later evaluation.
6. Creation of a MATLAB script that can analyse the processed data and estimate heart rate.
7. Creation of a performance class that measures processing time between frames for evaluation of the target algorithm.
8. Implement a motion estimation algorithm on the GPU and link in with the plotting, logging and performance classes to gain the same measurements as the CPU implementation.

Chapter 5

Development

This section details the development phases of the project, that encompasses design, implementation and testing phases. Initially at the start of each section the conceptual design and requirements of the system will be discussed and further expanded on how they were implemented. Relevance to other software components will be discussed as well as any constraints that had a detrimental effect on the project. The software components were designed using processes took from the test driven development cycle wherein initial tests were written during the implementation, the code was written and had it's output compared to the expected test output. When all tests passed the code was re-factored for readability and the next development cycle was started. This type of testing completed in the project through TDD cycles was testing to pass and helped make the system more reliable.

5.1 CMake

CMake facilitated the cross platform build process of the application and initially it needed to be configured. Configuration is comprised of creating CMake language files that describe how the project should be built in which CMake will translate to the native operating system build files.

5.1.1 Build System Design

Since the project is comprised of two separate applications, a serial and parallel implementation of motion estimation techniques it was logical to have CMake build two separate targets. A target is a term that is used to refer to a separate application to be built. However the parallel implementation is still governed by a serial application and most of the steps excluding the motion estimation algorithm will be symmetrical to the features in the serial application. Since the parallel implementation will be using many of the same features as the serial implementation it's logical to set up the project so that files can be shared between the two implementations.

This formed the first requirement that the CMake build system should facilitate. The build system should allow for two different applications to be built and share code files without any file duplication. Another requirement of the build system is that the external libraries and frameworks used in this project should be configured once and linked to the applications independently to reduce build time. It was important also to have an out of source build in this project due to the many advantages they provide. Out of source builds are when applications are compiled in a different directory to the source directory. All binaries, object, CMake and project files and are then stored separately to the code files.

One of the advantages is that build configurations are easy to clean by just removing the sub-directory they reside in. Since all of the build files are in one directory it creates a cleaner development environment, files are clearly organised into respective folders allowing for quick navigation. Building the project in another directory also means that the current source tree isn't polluted with build files and multiple different application configurations such as debug and release can be built without interfering with one another. The CMake build system was configured to accommodate for all of the requirements listed above. Figure 5.1 shows the folder structure that was used in the project and that met all of the requirements discussed.

```
1  build/
2  data/
3  results/
4      Parallel/
5      Sequential/
6  scripts/
7  src/
8      Parallel/
9          CMakeLists.txt
10     Sequential/
11         CMakeLists.txt
12     Shared/
13 CMakeLists.txt
```

FIGURE 5.1: CMake Project File Structure [1]

5.1.2 Build System Output

In figure 5.1 the "CMakeLists.txt" files are the configuration files for the build process, they describe directory wide instructions on how the project should be built. To meet all of the requirements in the previous section the file structure as seen in the figure was set up as well as the configurations for the configurations files were written. The configuration files are split into three sections, one project wide file that manages environment variables, external packages, compiler options and targets and two target wide files that create the executables, manage included files and link libraries. The project configuration file also handles the shared libraries by adding the directories as environment variables to the target files. The source code for

5.2 Pixel Capture

The datasets in this project are in the form of ten different DICOM files, which were also converted to MPEG-4 Part 14 video files for greater portability and version control. The DICOM data files have an average file size of 200MB whereas the maximum file size GitHub has full support for is 50MB. Conversion to MPEG-4 Part 14 video files allowed the pixel data to still be stored on the project repository allowing greater flexibility when developing the system. The following section will outline the design and implementation of the frame capture C++ classes.

[1] Figure 5.1 shows the project file structure, the build directory is only created after the configuration files "CMakeLists.txt" are executed with CMake

5.2.1 DICOM and MPEG-4 Part 14

Since two different file types are going to be used in the project it was decided that the classes should be interchangeable with a similar class structure the main code base could interface with. Figure 5.3 illustrates the minimal capabilities of the capture classes, and the functionality they will inherit. Important features of the classes are capturing frame information such as width and height and tracking the current position of the data stream. The method "»" in the class diagram is a C++ operator overload which returns the next frame of the data stream and updates the class attributes accordingly. This is for conformity with the OpenCV video capture (OpenCV, 2017b) class which utilises operator overloading to increase code readability. The overload takes a reference to an image container, in this case it's the OpenCV Mat data type which is used to store and access elements in a Matrix, and it will fill the matrix with all of the pixels in the next available frame. To maximise project portability and minimise dependencies, the DICOM image files were converted to MPEG-4 files by utilising inbuilt functionality of the MATLAB toolboxes. Figure 5.2 shows the MATLAB script that was created to achieve this. It first prompts the user to select the DICOM file and its output location, then the file is saved there in the MPEG-4 format.

```
% Get File Path
[in_file_name,in_file_root] = uigetfile('*.dcm');
in_data_path = strcat(in_file_root,
↪ in_file_name);

% Read in DICOM file
D = dicomread(in_data_path);

% Create VideoWriter Object
[out_file_name, out_file_root] =
↪ uiputfile('*.mp4');
out_data_path = strcat(out_file_root,
↪ out_file_name);
V = VideoWriter(out_data_path);

% Write data to video file
open(V)
writeVideo(V, D)
close(V)
```

FIGURE 5.2: MATLAB DICOM file conversion to MPEG-4 video files

The video capture class of OpenCV is used to read pixel data from the ".mp4" files on the hard drive so the implementation class in this project didn't need much more functionality than what was provided apart from the frame capture interface wrapper seen in figure 5.3. However the DICOM image file is a more complicated data structure and the ".dcm" file type isn't supported by the OpenCV video capture class so much more functionality had to be added to the DICOM capture class in the project to provide the same functionality as the base video capture class.

The DICOM image files used in this project were initially analysed to find out what type of data they contained, figure C.1 shows a log of some of the relevant information stored in the DICOM files. This data was obtained by writing a script in

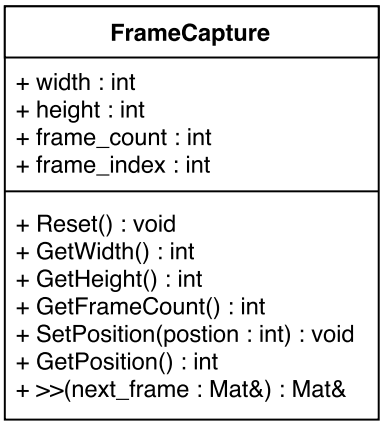


FIGURE 5.3: Frame Capture Interface Class Diagram

MATLAB and using the inbuilt DICOM functionality to read in the data and output it shown in figure 5.4. This data outlines that the transfer syntax used in the DICOM datasets is "1.2.840.10008.1.2.4.50" which is the default transfer syntax for lossy JPEG 8-bit image compression (*DICOM Library* 2017). Transfer syntax describes the encoding of the DICOM file and is used so that software systems can negotiate the management of the internal data more accurately. It also states that the raw pixel data is in a compressed format which will be unusable in this project that will use standard character arrays represent monochrome images. The photometric interpretation specifies how the pixel values are to be interpreted, figure C.1 shows that the photometric interpretation is "YBR_FULL_422" for the datasets used in this project.

```
% Select DICOM file
[file_name, root] = uigetfile('*.dcm');
file_path = strcat(root, file_name);

% Read in DICOM file information and print
D = dicominfo(file_path)
```

FIGURE 5.4: MATLAB DICOM information extraction script

From this data gathered from the MATLAB script it's clear that the DICOM frame capture class needs to detect the type of the file being read and interpret it accordingly. Due to time constraints of this project only the default transfer syntaxes of DICOM files and photometric interpretations were accounted for, to decrease development time while also still being able to parse all of the data files in the project. The DICOM class must first open a stream to the file and be able to return decompressed frames in an RGB colour space rather than YBR colour space. The conversion between the two colour spaces is facilitated by the DCMTK library used (Eichelberg *et al.*, 2014).

5.2.2 DCMTK

DCMTK, developed by Eichelberg *et al.* (2014), is a cross platform library that implements most of the DICOM standard, they provide a mostly C++ code base that can link directly in with the project via CMake. Most of the functionality of the DICOM frame capture class was facilitated by this library, it allowed the class to be

able to read and interpret the data sets. The pixel data was decompressed utilising this library also, which directly met one of the image capture requirements.

5.2.3 Frame Capture Classes

This stage of the implementation was crucial to the development of the rest of the project requirements, the final classes created provided a clean and simple interface to step through frames of the respective containers whether it was a MPEG-4 or DICOM file. Both classes implemented the basic functionality seen in 5.3, however because of the greater complexity of the DICOM data set the DICOM capture class is considerably larger and more intricate. The reason for this is that it has to modify the data in certain ways dependant on the input, for example if an uncompressed DICOM file is read in the data can be directly interpreted and passed to the calling program, however if the data is compressed then the data has first be uncompressed and changed to an appropriate colour model. The final class diagrams of both implementations can be seen in figure 5.5. Development of the MPEG-4 capture class was relatively straight forward, especially due to the clear on-line documentation. The DICOM frame capture class however took a lot longer to develop than initially anticipated, due to the underestimation of the file complexity which is represented in figure 5.5.

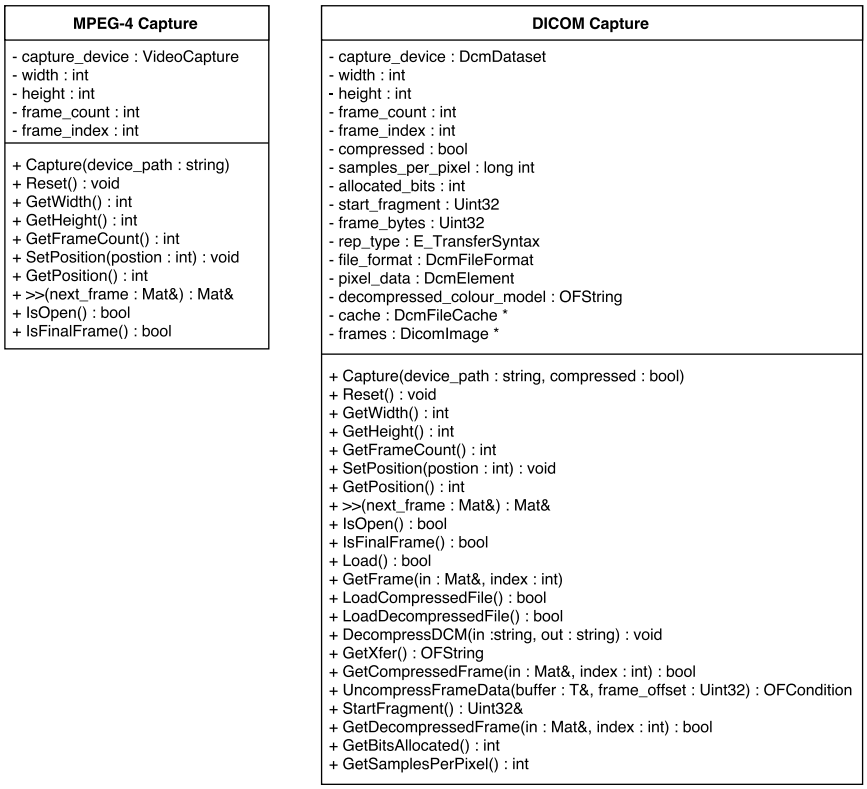


FIGURE 5.5: MPEG-4 and DICOM frame capture class diagrams

5.3 Motion Estimation on the CPU

This section will specify the algorithms used in the serial application for motion estimation and justify their inclusion. To gain an accurate benchmark of a GPU and

CPU implementation the code will be structured as similarly as possible to how the implementation on the GPU will be formed.

5.3.1 Motion Estimation Algorithm

As discussed in the background research section 2.4.1, of this report full search block matching is a viable way of calculating motion of a time series data set. The conceptual method of calculating the motion of an image sequence is to split an image frame into blocks every x pixels in the horizontal and vertical directions. These become the reference blocks for comparison and consequently the blocks that the motion will be estimated for. The next step is to iterate over each block and compare it to every possible block bounded by a predefined search area around the original reference block location in the next or previous frame in the time series. This will return some error function in which the process will aim to minimise subsequently finding the best matching block around the original location. The motion is then calculated as the displacement between the two block locations. The search window size in this project is the block size multiplied by three.

```
1  width = 100;
2  height = 100;
3  block_size = 10;
4  step_size = 1;
5
6  for i = 0 to 90
7      for j = 0 to 90
8          x_range1 = i to i + block_size
9          y_range1 = j to j + block_size
10
11         block = frame_1(x_range1, y_range1)
12
13         lowest_error = +Infinity
14         best_match = block
15
16         for ii = -block_size to +block_size
17             for jj = -block_size to +block_size
18                 x_range2 = x_range1 + ii
19                 y_range2 = y_range1 + jj
20                 compare = frame_2(x_range2, y_range2)
21                 error = CostFunction(block, compare)
22
23                 if error less than lowest_error
24                     lowest_error = error
25                     best_match = compare
26
27         estimated_frame(x_range, y_range) = best_match
28         j = j + step_size
29     i = i + step_size
30 accuracy = CostFunction(estimated_frame, frame_2)
```

FIGURE 5.6: Pseudo-code for Full Search Block Matching

A motion compensated image of the reference image is created by taking all of the best matched blocks and placing them in a new matrix in the locations of the best matched reference blocks. An error rate of the accuracy can then be obtained via several methods by comparing the motion compensated image to the actual data in the time series if present. A simple pseudo-code example using similar syntax to MATLAB can be seen in figure 5.6.

The motion vectors are obtained in the execution of the FSBM algorithm in the sequential and parallel implementations. However this data alone isn't very useful for estimating the BPM of the time series. The angle and magnitude of the motion vectors relative to the starting position are how the BPM is estimated, and they are calculated in the FSBM algorithm. The angle describes the direction of sections of the image in a single frame, once calculated the average of all the angular motion values will be a good representation of the motion direction in the image.

5.3.2 Algorithm Implementation

The pseudo-code in figure 5.6 was an extremely helpful base algorithm to follow, in generating this prototype aspects that previously seemed trivial such as the inner loop over neighbouring blocks were found to be implementable in many different ways. In the process of prototyping the algorithm code like this before it saved development time by specifying some fundamental concepts of the FSBM algorithm. The actual serial implementation however varies slightly to the pseudo-code version. One of the ways it's different is that it has boundary checks so that it will never calculate or compare two blocks whose locations are outside of the image width and height bounds, this increases performance and reduces code out of range index exceptions.

Another major difference is that the pseudo-code doesn't take into account two identical surrounding neighbourhood blocks. In the example case the last block to be processed with the lowest error will always be the best matched block, however in reality this isn't true and there isn't a reliable way to tell which block stems from the original if at all. In the serial and parallel implementations this was overcome by taking the block displacement into account also. The Euclidean distance to each block is computed and the block with the lowest error is only set as the best block if it has the lowest Euclidean distance of all best blocks so far. This way minimised outliers by giving the blocks with zero variance a displacement and motion vector of exactly zero which subsequently increased the application accuracy significantly. Figure 5.7 shows the logic of the inner loop of the serial application, this would be a direct replacement for the logic inside the fourth for loop in the pseudo-code seen in figure 5.6. The final serial FSBM code can be seen in figure B.3 with the supporting functions in B.2.

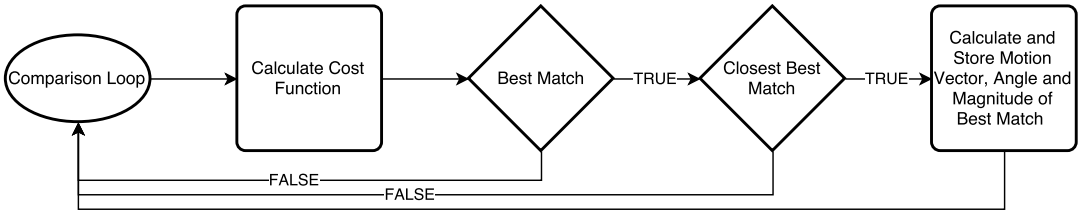


FIGURE 5.7: Inner FSBM comparison loop logic for best block selection

5.3.3 Program Structure

In the implementation the program structure was laid out in a modular style. This was so that the later GPU implementation would conflict with as little as possible to save on development time. The short time frame of this project presented challenges when trying to implement two separate systems, this amount of code duplicity will help to ensure the project deadline will be reached with a complete artefact. Figure 5.8 shows the planned program structure from the previously implemented components and the initial requirements gathered in chapter 4. This will allow for the parallel processes to be inserted between the phases outlined and for the algorithm implementation step to be interchanged with the parallel kernel algorithm.

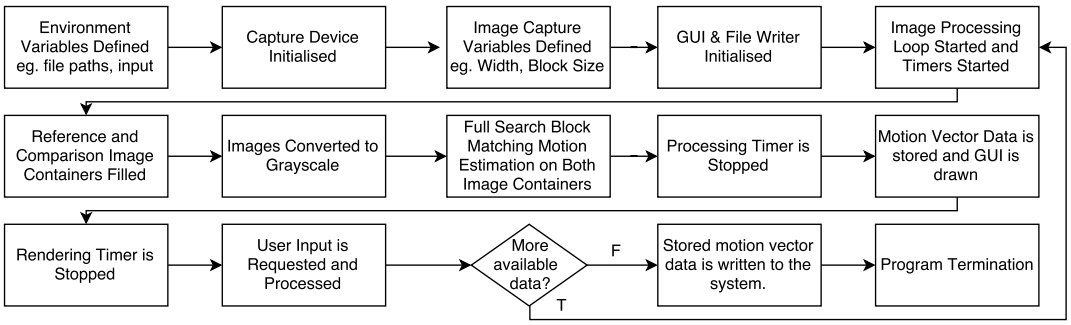


FIGURE 5.8: Conceptualised Serial Application Structure

5.4 Exporting Motion Vectors

Once the motion vectors are calculated for a frame of the dataset, they will proceed to the next frame and continue this cycle until they reach the end of the dataset. Once this point is reached the cycle will either restart, the next dataset will start to be analysed or the program will exit. If the program exits then all of the information that has been calculated for the images has been wasted. This would mean that the program would need to be executed again which would waste a considerable amount of time if this had to be done every time some results wanted to be analysed. This workflow also doesn't allow for any external validation or analysis of the data either, so post processing of the data in software packages such as MATLAB wouldn't be possible. This is the main justification for what the following section will document; a file writer class that outputs all of the calculated information.

5.4.1 File Writer Class

The file writer class has some requirements that it needs to achieve to allow the serial and parallel implementations to fully utilise their retained data efficiency. It must be modular in a sense that new sources of data can be added and logged without obtrusive additions to the class file. It must also work in any scenario with any data type, to prevent cases in the future where calculated data is incompatible with the file writer class data structure. Externally to the application it must be easily interpreted by other software packages to decrease time writing file parsers for written data as this would be counter-intuitive. If all these requirements are met the file writer class should be able to accommodate the analysis of data in this project greatly.

To make the file writer class modular in writing data, consideration had to be given into three main areas; how the data should be represented as a data type,

how the class will distinguish between different data sets, and how the output file will know what the data is representative of. It made logical sense that the data representation should either be character arrays or the C++ string data type, because this will support almost all types of data the application will need to write to file. It was quickly realised that in order to meet the second consideration the functions had to accept a variable number of arguments. Otherwise there wouldn't be a reliable way to pass multiple different permutation types and lengths to the writer function.

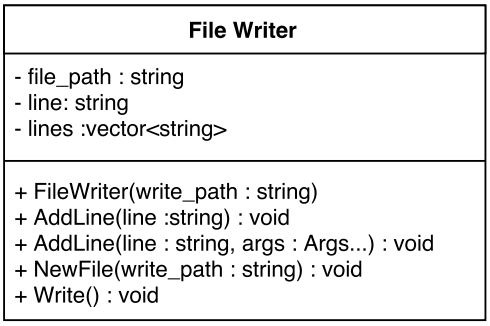


FIGURE 5.9: File Writer Class Diagram

To solve this issue a C++ feature called vardiac functions was exploited. Vardiac functions are functions that can take any number of arguments, which would allow the file writer class to accept a number of parameters and write data frames rows at a time as the data becomes available. This conceptual way of storing the data would mean that the column index of the output file will represent time, which is appropriate for the later uses of the data. Utilising this feature of C++ the class would be able to meet the second requirement and also the third requirement of the file writer. This is because parameters could be passed to the class at the time of program initialisation in the form of column headings that describe the order of data they will pass to the file writer. This class design makes the file writer class simple to write and maintain while also having a very low cohesion to the project which will make it applicable in many other applications also. As shown in figure 5.9 the vardiac functions are the two methods "AddLine()", the latter method is called until only one parameter remains and the first method is called ending the loop of function calls.

5.4.2 Export Format

As mentioned above the output format of the file needed to represent the data in a way that would be easy to analyse and interpret externally. To achieve this the file writer class shown in 5.9 writes data out in tab delimited rows better known as a DSV file. DSV files are files that store data by separating them in rows with a delimiter and in columns with a new line. MATLAB will be used to analyse the data at later stages, and it's inbuilt functionality is very effective at handling DSV files. The simplicity of reading the files in can be seen in figure 5.14, not having to write a parser for the file saved development time and added cross compatibility with software systems by using a format that's widely accepted.

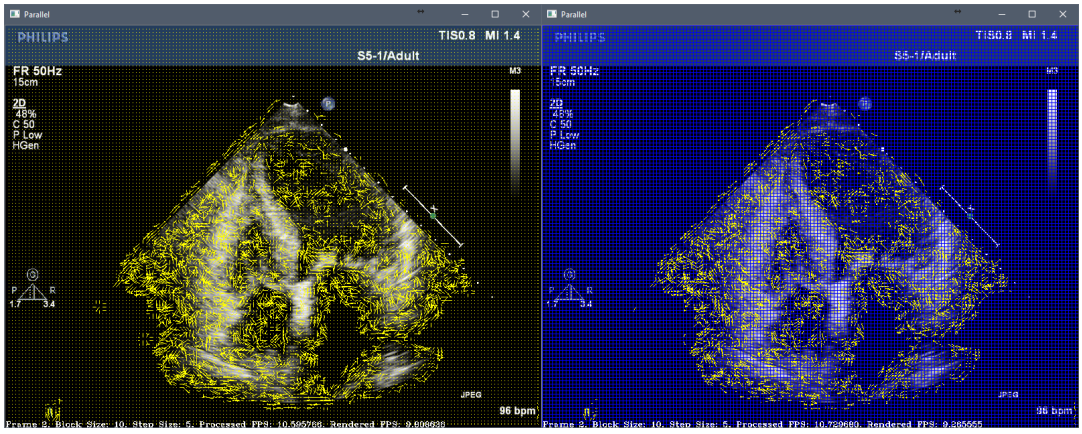
5.5 Motion Vector Visualisation and Plotting

One of the requirements of the software system shown in section 4.2 is that the motion vector data must be visualised in real-time as it is calculated to visually assess

the accuracy of the algorithm and performance metrics such as frames per second. The OpenCV (2017c) drawing library was utilised to achieve this. It provides functions that can draw shapes on matrices with anti-aliasing. The resultant images were then displayed using the "highgui" GUI functionality of OpenCV.

5.5.1 Visualisation

The motion vector data can be represented simply by plotting the vectors as arrows over the original position and frame they were captured from. All of the drawing functionality that has been implemented in the C++ source code is contained within a drawing name space, promoting code readability. The name space contains functions for calculating the Euclidean distance of two vectors, calculating the angle of a motion vector around a circle with the radius of the vector magnitude, drawing arrows onto an image, drawing text, drawing motion vectors for a whole image and visualising the angles of the motion vectors by plotting them around the HSV colour space. The requirements of the visualisation functions are that they must be independent to the vector analysis and not disrupt the program execution. The functions must also not affect the results of both the motion vectors and the performance data and finally they should represent the motion vector data accurately and clearly.



(A) Motion Vector Plot (B) With Block Grid

FIGURE 5.10: Motion Vector Visualisations

The drawing functions do not affect the execution of the program or affect the calculated motion vector values; in the C++ code the drawing functions do not modify any of the data and the performance heuristics are calculated independently to the processing heuristics. They do however slow down the execution of the program because of the amount of time taken to render the images on screen as well as the overhead of drawing calculation for each separate motion vector. Due to this they can be toggled on and off at execution time or previously in the code. To plot the motion vectors the data buffer is passed to the drawing function which then loops over every value in the order they were calculated. The position of each of the vectors is offset to position them in the centre of the block position they were calculated for. After this they are drawn with the arrow pointing to the centre of the best matched block location and optionally a grid is drawn showing all of the block locations. Figure 5.10 shows the result of the drawing functions with figure 5.10a and 5.10b showing only the motion vectors and both the motion vectors and the searched block grid respectively.

5.5.2 Plotting

The above figures are a good representation of the individual motion vectors, however the overall motion of the image isn't easily calculable by eye due to the sheer number of motion vectors on screen at one time. The average direction of the frames is what will be used to calculate to heart rate, so a logical representation would be to have a live graph that plots the average motion of each frame over time. Similar to the electrocardiography electrocardiograms of voltage over time. Initially plotting libraries for C++ were researched but most of them were found to be too heavy or they weren't being actively maintained or had poor platform support. Time was wasted in this development phase trying to link some of the libraries into the build process via CMake. It was eventually decided that a plotting class would just be created that will utilise existing resources in the project.

OpenCV already has the functionality to display image matrices on screen so using this the plotting class would just have to generate matrices representative of a graph showing average angular motion of the data set over time. The requirements of the plotting class is that it must be able to plot axes that are clearly labelled and remain static over all rendering. The angular motion axis must be responsive to the data range in view so that the plotted line is clear and is effectively using the available space of the window. The class must also be configurable for aspects such as x and y axis ranges and window size to maintain support for multiple systems. It should also provide a simple way to add points to the plot. With all of these requirements it was decided that the class should be modularised into functions that can draw the key features in a layered style. Three main elements have to be drawn onto the plot; the axes, labels and lines between in-view plotted points. They should be drawn in this order so that when the pixels of the matrix are overwritten the lesser important elements such as the labels and axes will be lost rather than the plotted information.

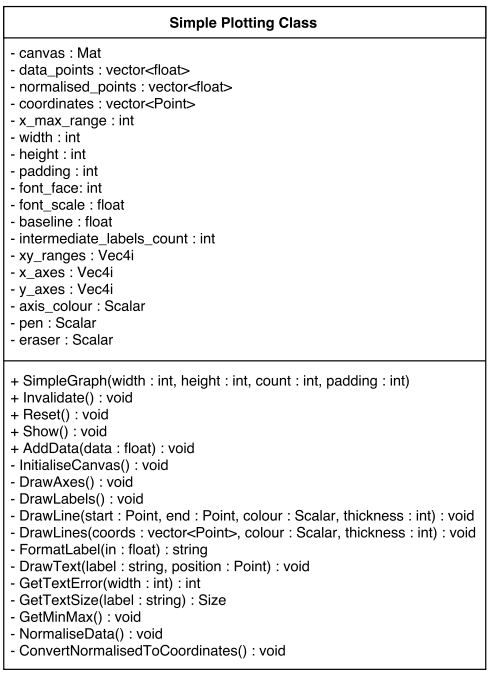


FIGURE 5.11: Plotting Class Diagram

Figure 5.11 shows the class diagram design for the plotting class. It's based on having a single canvas that is updated with elements and plotted points every time

the "Invalidate()" function is called. This function can be called by the host program or it will automatically be called whenever the "AddPoints()" or "Reset()" functions are called. The canvas is cleared every time this function is called and the axes, labels and points are drawn in that order. The plotted points are normalised and translated into coordinates between the bounds of the y axis, giving the appearance of a responsive axis system thus meeting the requirements laid out above. When the class is initially constructed the parameters it accepts define the structure of the plot. The width and height determine how much of the window it will take up, the padding decides how much gap will be between the plotted graph and the side of the window and finally the count variable decides how many points can be plotted along the x axis at any one time. Figure 5.12 shows the resultant plot of the first 128 frames of one of the datasets.

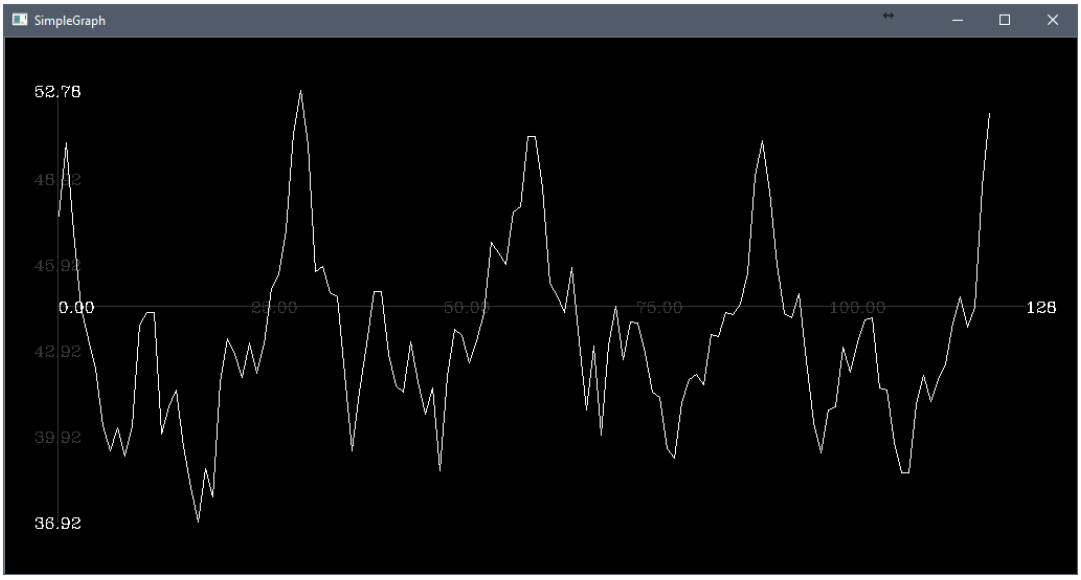


FIGURE 5.12: Average Angular Motion Plot of the Ultrasound Images

5.6 Motion Vector Analysis

At this stage in the development process the motion vectors have been recorded, visualised and written to a file for external analysis. In this section the external analysis that occurs on the data will be discussed. BPM estimates are calculated for the datasets and the data is evaluated at in this development phase. The main tool that facilitated this is MATLAB, and this section will describe the script that estimates the BPM data from a time series of motion vector data.

5.6.1 MATLAB

MATLAB is used to evaluate the motion vector data calculated by the serial and parallel implementations because of the inbuilt analytical and plotting libraries. They saved a lot of development time especially with testing, since the functions are built into MATLAB they have already been tested exhaustively by the development team and the community. There are three main requirements of this stage in the development process. The first one is that the MATLAB script will have to retrieve the original capture rate of the DICOM file that was used to calculate the data and the

heart rate recorded in the file dataset to compare with the estimated heart rate. Secondly it must be able to parse the motion vector data files and finally it must be able to decompose the time series of motion vectors into components of different frequencies to find the period of one cardiac cycle.

```
%% Read in DICOM file and Get heart rate

[file_name, root] = uigetfile('*.dcm');
file_path = strcat(root, file_name);
info = dicominfo(file_path);
images = dicomread(info);
ground_bpm = info.HeartRate;

%% Get Input for Capture Rate
% Display DICOM overlay underneath input dialogue
imshow(images(:,:,:,1));

% Ask the user to input the capture rate (50Hz)
input = inputdlg('What is the capture rate?', ...
    'Input Capture Rate', 1, {'50'});
capture_rate = str2double(input{:});

% Close all windows
close all;
```

FIGURE 5.13: MATLAB Code to Extract the Heart and Capture Rate from DICOM Files

The first requirements are met using the MATLAB code in figure 5.13. This code first creates an open file dialogue window where it prompts to select a ".dcm" file. It uses this file to extract the heart rate from the dataset. Extracting the capture rate using this method however wasn't possible. In the process of creating this script the dataset, shown in C.1, information was searched, but no record of the capture rate was recorded. One piece of data that was recorded was "FrameTime" which is the nominal time in milliseconds between each frame of the DICOM dataset (Eichelberg *et al.*, 2014). Using this piece of information the frame capture rate can be calculated as 1000 divided by the "FrameTime" value, modifying the BPM equation in 2.3 to the equation in 5.1.

$$\text{BPM} = \frac{\frac{1000}{\text{FrameTime}} \times 60}{\text{Cardiac Cycle Frequency}} = \frac{60000}{\text{FrameTime} \times \text{Cardiac Cycle Frequency}} \tag{5.1}$$

To extract the information from the saved files MATLAB provides an inbuilt function that can read tab delimited files into column vectors. The code in figure 5.14 does this effectively, meeting the second requirement of this implementation phase. It first prompts the user with a dialogue box that allows them to select a text file to read in. Since the serial application outputs the files with a unique file name there will be many data files to choose from, from every time the program has executed in the past. At the bottom of the code snippet the angles are multiplied by the magnitude values to weight them to prefer values in the result that had a greater velocity component in the original image.

```
%% Get File Path
[file_name,file_root] = uigetfile('*.txt');
data_path = strcat(file_root,file_name);

%% Read and parse file
data = tdfread(data_path, '\t');

angles = data.Angle_00x2D360;
magnitude = data.Magnitude;

weighted_angles = angles .* magnitude;
```

FIGURE 5.14: MATLAB Code to Parse a Tab Delimited File

5.6.2 Fast Fourier Transform for BPM Estimation

This section will describe how the development for the third requirement of the external data assessment was met. The third step of calculating the BPM of the motion vector data is more complicated than the previous two done in this MATLAB script. It performs the Fast Fourier Transform described in section 2.5.1 on the parsed dataset subsequently retrieving the peak frequency which utilises the equation in 5.1 to estimate heart rate. The FFT will decompose the signal in figure 5.15 into frequency components, and the frequency with the highest amplitude should represent the frequency of one cardiac cycle.

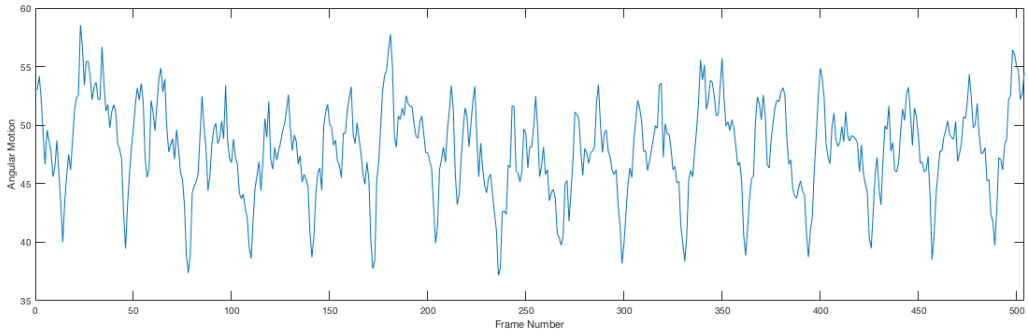


FIGURE 5.15: Average Angular Motion of Apical Four Chamber View

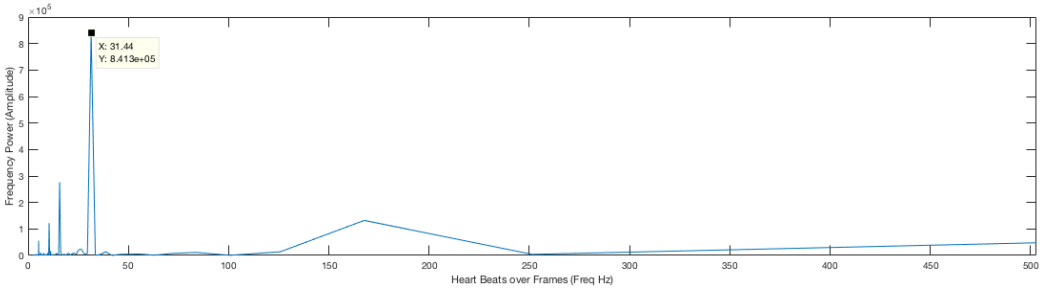


FIGURE 5.16: Fast Fourier Transform Frequency Components

Figure 5.17 shows how the FFT of the angular motion data was calculated in MATLAB. The first step uses the "fft()" function of MATLAB on the Y axis data. This Fourier transform return of this function places a new element into the first

position of the new vector, this element is the sum of all the frequency powers and is removed since it's not relevant in this scenario. Half of the data is duplicated after the midpoint in the new vector so this is also removed before any further processing. The power of the frequencies is then calculated for the Y axis and a sequence of values normalised between 0 and 0.5 the is generated for the X axis in which the period is calculated as the inverse of the sequence. The cardiac cycle frequency is then taken as the peak frequency on the x-axis as shown in 5.16

```
ffY = fft(Y);
ffY(1) = [];
n = length(ffY);

max_frequency = 0.5;
Y = abs(ffY(1:floor(n * max_frequency))) .^ 2;

frequency = (1:n/2)/(n/2) * max_frequency;
period = 1 ./ frequency;

cardiac_cycle = period(find(Y==max(Y), 1, 'first'));
```

FIGURE 5.17: MATLAB code to calculate FFT of Angular Motion Data

5.7 Motion Estimation Algorithm Performance

All of the components of the serial implementation have been implemented, including external validation scripts to predict the heart rate from the motion vector data. However the performance of the implementation is still unknown, performance data will be crucial in the assessment of parallel applicability to this problem scenario. This section details the implementation of a timer class that will estimate the frames per second the application can achieve when processing data and rendering it. Frames per second are being used as the measurement heuristic because it will provide a direct comparison to the capture rate of the device being processed. A shortened log of information from device that captures the ultrasound image data can be seen in C.1. It shows that the nominal time between each frame capture is 19.8390 milliseconds which can be converted to a frame rate of 50.4058. So a real time implementation should be able to process at least 50-51 frames a second to keep up with the rate of capture. This provides a baseline and justification for using FPS to evaluate algorithm performance.

5.7.1 Timer Class

The timer class will be used in both the sequential and parallel implementations so it should be compatible in both these environments. It's required to have an easy to use interface with negligible impact on application performance. It should also be re-usable in the same application because the implementations will want to capture the rate of processing and rate of visualisations to show the comparison. Ideally it would also be beneficial to show the FPS as an overlay on the visualised data and have it update over time. The derived class diagram is shown in figure 5.18.

It provides simple functionality that will be highly pluggable into almost any application. It works by initially setting a max number of samples to record in the

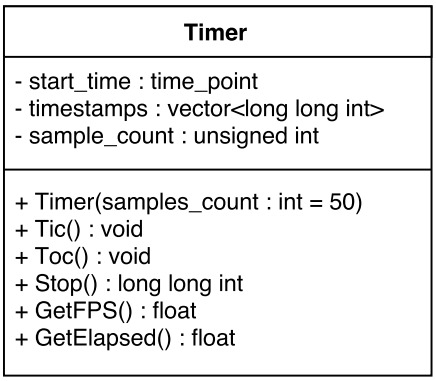


FIGURE 5.18: Timer Class Diagram

constructor. This value determines how many values the FPS is derived from, the greater the number the more generalised the result will become until eventually it's the FPS for the entire file. The default was decided as 50 due to this being the capture rate of the devices that produces the ultrasound images. This would mean that the timer class would be accurate to the last second if the rate of processing was 50Hz in the application which is idea for this situation. The other functions of the class "Tic()" and "Toc()" provide a similar interface for resetting the start time and adding the elapsed time to the dataset of timestamps respectively.

5.8 Motion Estimation on the GPU

This stage in the development process focusses on the conversion from the sequential implementation to the parallel implementation. Software components so far have been designed with this stage in mind, trying to maximise portability and compatibility of classes such as the timer in section 5.7 and file writer in section 5.4.1. The main challenges of the implementation and design will be the kernel execution code and the host management of the kernel queue and memory buffers since these are areas that the author had no initial experience in or knowledge of. The full parallel application will share a large percentage of the same program structure as the serial application. The changes will be mostly made up of configuring the GPU environment, managing the kernel code and reading and writing to buffers.

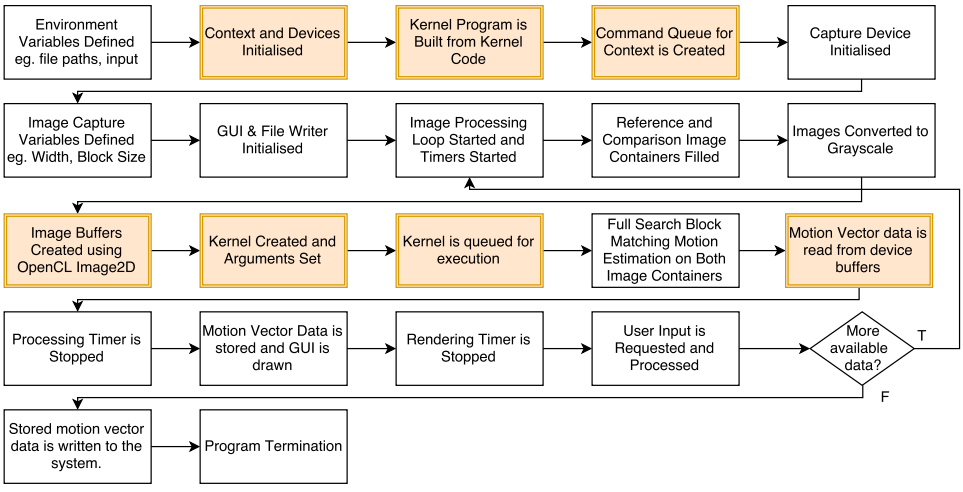


FIGURE 5.19: Conceptualised Parallel Application Structure

5.8.1 Algorithm Design

Figure 5.8 shows the conceptualised structure of fundamental logic components of the serial application, in figure 5.19 the differences in structure compared to the serial implementation are highlighted to show the contrast of the two programs. The algorithm design in section 5.3.1 will be the basis for the algorithm implementation in this stage of the implementation. However in its current state its not compatible with coding in the parallel kernel. If it was executed in parallel at the moment it would just be a sequential algorithm being executed many times with the possibility of causing unintentional data corruption. This could happen due to the way parallel kernels are executed out of order allowing random access to a dataset. The nature of parallel algorithm design is to increase the performance of a naturally sequential algorithm, in this implementation focus is initially shifted to implementing a working solution in parallel before any optimisations are made. For this reason the Parallel kernel structure will have many aspects that are identical to the structure found in figure 5.6. The parallel application will implement a full search block matching algorithm also so that the performance can directly compared to the serial program.

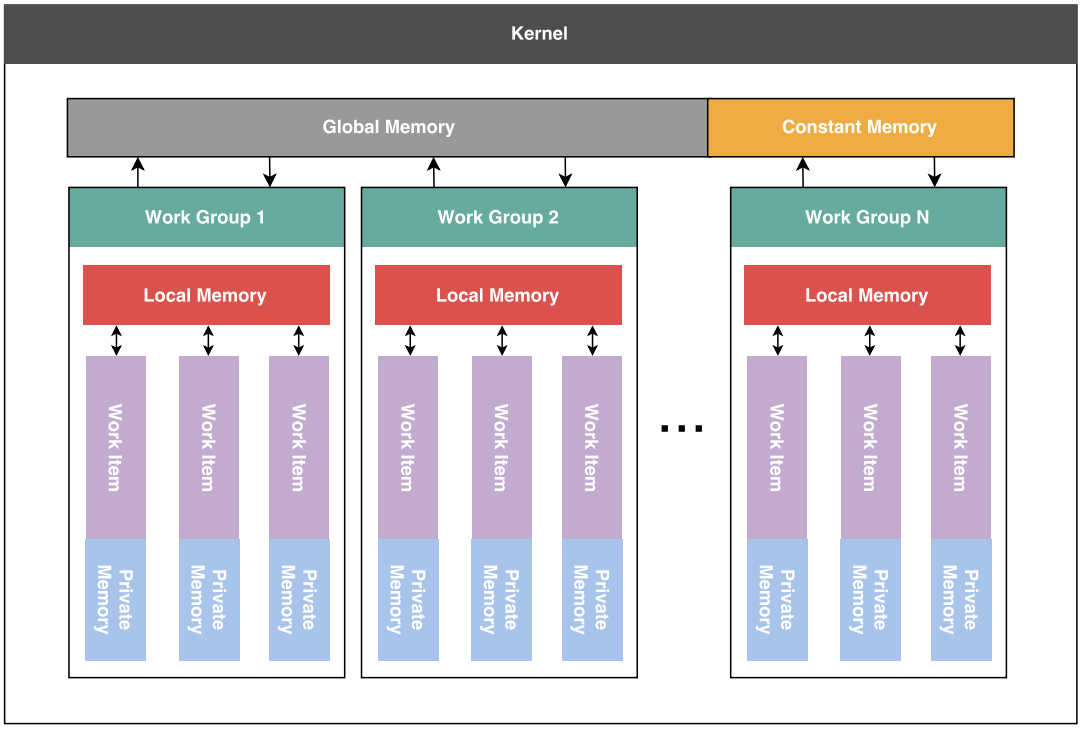


FIGURE 5.20: Representation of GPU memory access in OpenCL

When kernels are executed they are placed into vector or matrix space where each element is one instance of a kernel as described in section 2.2.3. All of the kernels in this index space execute the same kernel code and have access to all of the same information written globally to the device or passed in as a parameter to the kernels. The variables that are created in the kernels are stored in private memory and are independent to each individual kernel. To share information between kernels, local memory can be utilised to efficiently write and read data within a work group. Similar to how kernels in the global space are placed into an index space, kernels that are queued with local memory parameters will be split into work-groups where each work group can share an amount of allocated memory between their processes. In this implementation only global memory was used initially because

it simplifies the logic, however it will be later implemented after the base algorithm has been implemented. The other type of memory available to the kernels in OpenCL is constant memory, which is similar to global memory in the essence that it's accessible kernel wide however it's read only. A conceptual representation of the memory for each work item can be seen in figure 5.20.

5.8.2 Algorithm Implementation

When implementing the FSBM algorithm in parallel, the first concern was how to split the data up and process it separately to maximise data parallelism. Since the implementation will be in the global memory space it makes the decision easier since the global index space can be multidimensional. This reduces the complexity greatly because similar parallel patterns such as map are well documented on-line and are extremely simple to implement. One example of the simplicity is a global threshold application, where the global kernel range is expressed as the width and height of the input image, the kernel would just take the pixel value that corresponds to its ID and threshold it. This is called the map pattern where a function is applied to every element in a dataset in parallel (Samadi *et al.*, 2014). Instead of defining the global space as a vector it can be defined number of blocks that fit into the width and height of the frames respectively. Processing the data this way allows each blocks best match to be found independently of all the others. There are many different ways the data could be split and processed however this way fit the design and requirements specification and seemed logical. Defining the global space of the command queue subsequently places all of the kernels in a two dimensional matrix space. This effectively replaces the first two loops of the pseudo-code example, in place of allowing multiple work items to process every block instead.

To differentiate the information each kernel computes, the kernels need to know what block to calculate the best match for otherwise all of the kernels would just process the first element in the image. By splitting the data up as aforementioned, the ID of each kernel will give a position along the x and y axis. The function "get_global_id(dim)" returns the ID of the kernel in the dimension specified by the "dim" parameter. Since the global space will be split into two dimensions the functions "get_global_id(0)" and "get_global_id(1)" would return the position of the currently executing kernel in global space. Translating this two dimensional coordinate into a one dimensional position however will need to be done for assigning the motion vector output into the results buffer. Since the "Image2D" data type of OpenCL is being used, this way of addressing the array made the coding easier. Conceptually the implementation is now the same as the serial implementation bar the first two loops since that's facilitated by the global kernel execution index. The final OpenCL kernel code can be seen in figure B.5 with the supporting functions in B.4. The supporting functions allow the kernel code to be more readable since no utility functions such as the OpenCV "sum()" functions exist in the OpenCL C99 language with the exception of basic math operators such as "atan2()" and "sqrt()".

5.8.3 Experimental Sub-Optimal Optimisations

The OpenCL FSBM Motion Estimation code is a basic implementation and it doesn't take advantage of many of the features that parallelism and OpenCL can offer. Throughout the development phases of the parallel implementation consideration was given on areas that could be greatly improved, in terms of the data throughput and algorithm inefficiencies. There are many accurate algorithms for estimation of motion

in image sequences as described in section 2.2.4. However they trade off accuracy for better performance which may be a worthwhile trade-off to achieve much faster parallel processing, even for the serial implementations the algorithms mentioned can have catastrophic performance boosts. In terms of increasing accuracy within the serial and parallel certain aspects could be changed. One of which is that due to the nature of apical four chamber ultrasounds the vertical component of velocity is much higher usually than the horizontal. With this the accuracy can be improved by tracking a larger rectangular window in the vertical direction rather than in the horizontal or equal proportioned blocks.

Due to time constraints in this project this wasn't experimented with and for the most part this is the reasoning for why further requirements to further improve the parallel performance weren't added. Attempts were made to improve the performance externally to the time plan of this project but not enough resources were available to fully test and evaluate their accuracy in comparison to the baseline FSBM using SAD criterion. In the attempt to increase the performance of the parallel system assumptions were made about the data and how they could be calculated. The current bottleneck of computing the SAD criterion for neighbouring blocks is that a large portion of the data has already been processed but it has to be recalculated every time. This attempt tried to derive a new criterion based on SAD that could fully exploit data parallelism of computing FSBM. The new experimental sub-optimal criterion assumes that the values calculated are always greater than zero, which in this case they always will be. It also assumes that the sum of both blocks the SAD is being calculated for share the same sign, which will always hold due to assumption one of being 8 bit images. It attempts to estimate or calculate SAD by taking the summation of the two matrices independently, and then taking the absolute difference of the two matrices.

$$\sum_{i,j=0}^{n-1} y_{ij} = \sum_{i,j=0}^{n-1} x_{ij} - \sum_{j=0}^{n-1} x_{*,j} + \sum_{j=0}^{n-1} y_{*,j} \tag{5.2}$$

$$\sum_{i,j=0}^{n-1} x_{ij} \geq \sum_{i,j=0}^{n-1} z_{ij} \rightarrow \left(\sum_{i,j=0}^{n-1} x_{ij} - \sum_{i,j=0}^{n-1} z_{ij} = \sum_{i,j=0}^{n-1} |x_{ij} - z_{ij}| \right) \tag{5.3}$$

$$\sum_{i,j=0}^{n-1} y_{ij} \geq \sum_{i,j=0}^{n-1} z_{ij} \rightarrow \left(\sum_{i,j=0}^{n-1} |y_{ij} - z_{ij}| = \left(\sum_{i,j=0}^{n-1} x_{ij} - \sum_{j=0}^{n-1} x_{*,j} + \sum_{j=0}^{n-1} y_{*,j} \right) - \sum_{i,j=0}^{n-1} z_{ij} \right) \tag{5.4}$$

Equation 5.3 shows that the SAD of two matrices will be equal to the difference of the sums of the independent matrices if the sum of the comparison matrix x is greater than or equal to the sum of the reference matrix z. This is based off the premise that $|x - y| \leq |x| - |y|$ and that $xy = |xy| = |x||y| \rightarrow |x - y| = |x| - |y|$. The constraint of calculating the SAD using this method is that it isn't guaranteed that elements of the comparison matrix x will be greater than the reference matrix z. So to remove this constraint either the max value of the reference matrix z is added to every element of the comparison matrix, or the max intensity value is added. In the implementation of this experimental feature the max intensity level was added to save computation of finding the max element of an array.

The major benefit of computing the cost functions of neighbouring blocks using this method is that the sum of adjacent blocks in the vertical or horizontal direction can be estimated as long as the initial sum of the most top-left matrix block has been calculated. This is demonstrated in 5.2 where only a third of the y matrix from figure

5.21 is needed to compute it's sum and subsequently it's cost function, sum of absolute difference. Equation 5.4 reinforces this by showing how the SAD is computed from using previously calculated sums. The number of saved computations scales with the size of the block. In the example figure 5.21, where it's shown that this method can successfully estimate SAD using independent sums of greater element wise values than the reference blocks, the number of saved computations is six in the new matrix. Using the conventional SAD cost function 18 elements would have to be accessed to evaluate the SAD of matrix y however with the new way of calculating SAD only 6 elements have to be accessed to get the full SAD of the matrix y. This reduces the number of times the kernel has to read from global memory by a factor of three. The number of memory accesses will always be a factor $O(n)$ faster than the traditional way of calculating the cost functions.

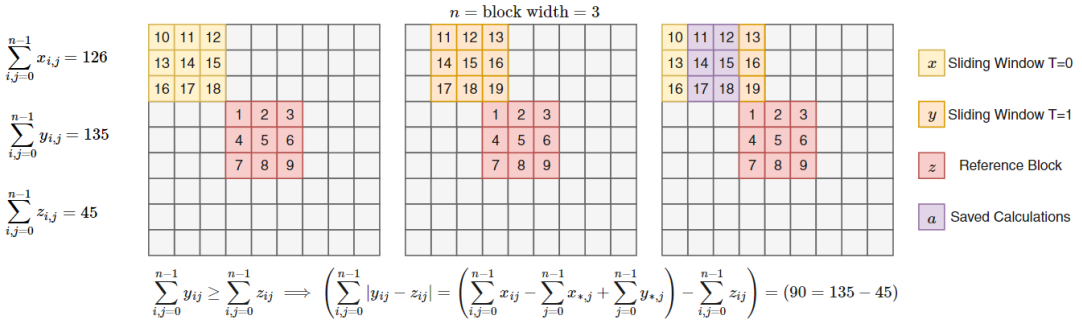


FIGURE 5.21: Calculating SAD as Two Separate Absolute Matrix Summations

Other methods of increasing the performance in the parallel kernel code could have also been implemented but for similar reasons as above not enough time was given to correctly test and evaluate the accuracy of the implementations so that they could be justified and valid in the final application. The final way of improving the performance that was considered is vectorisation of the code and unrolling the parallel loops. Simple code vectorisation exploits the data types in the OpenCL C99 language such as "int4" by processing multiple data points simultaneously instead of one "int". This improves the performance because the overhead of calculating an element wise vector operation is less than defining the individual elements in memory and performing the operation on them independently. Similar to this local memory could have been used to allow fast access to elements in the same work group, accessing elements in local memory is a lot faster than accessing the same data in global memory.

Chapter 6

Testing and Evaluation

The final stage of this project before the evaluation stages is the final testing phase, which is descriptive of the "Cutover" phase of RAD. This is where the initial requirements and deliverables of the system listed in section 4.2 are tested to see if they were met. Throughout the project the implementation stages have been guided by a test to pass system where the development was continuous until a certain output or test was passed. This process is from the Test Driven Development Cycle and has proved to be effective in RAD. Since Rapid Application Development has a heavy focus on implementation the merge of these processes maximised the allocated resources to the implementation stage of each iteration. This was especially suitable in this project due to the heavy code requirement for the multiple applications and algorithms. The tests executed at each individual stage acted as a disposable model and prototype in place of a full design document, they were facilitated by the partial designs completed for core software components such as the Motion Estimation algorithm sections and the modular classes such as the File Writer and Timer. In this section the project will be built and tested from scratch to validate the integrity and conformity with the requirements specification

6.1 Environment Configuration

The OpenCV, OpenCL, DCMTK and CMake dependencies of the project helped facilitate a large chunk of what was achieved in the implementation stages. One aspect that benefited the development was that after the initial configuration the setup scripts barely needed changing at all. After downloading a fresh copy of the repository from GitHub it quickly became apparent that the installation instructions for multiple third-party libraries varied massively. Due to this a set of instructions was added to the project to ensure reliable installation on multiple different systems. DCMTK was also set to an optional library since it isn't needed since the advent of the converted video files stored on the GitHub repository. For demonstration purposes this facilitates a faster build process, however for full program compatibility the DICOM files are required. Especially when estimating the BPM in MATLAB. If the DICOM files aren't present then the values for capture and heart rate have to be entered manually. Figure 6.1 shows the output of the build process on a Windows 10 machine using Visual Studio 2017.

6.2 Interchangeable Frame Capture and Timer Classes

To allow portability and maximise development time one of the requirements was that the program had to be able to parse the data from the DICOM data set and a much smaller ".mp4" file stored on GitHub that only contains the raw pixel data.

```
1> Working directory:
.../UOL-FinalYearProject/build/ProjectY3 x86-Release
1> -- Trying to find DCMTK expecting DCMTKConfig.cmake - ok
1> -- OpenCV ARCH: x86
1> -- OpenCV RUNTIME: vc15
1> -- OpenCV STATIC: ON
1> -- Found OpenCV 3.2.0 in ...
1> -- Found OpenCL in ...
1> -- Configuring done
1> -- Generating done
1> -- Build files have been written to:
.../UOL-FinalYearProject/build/ProjectY3 x86-Release
```

FIGURE 6.1: CMake Build Process Output

Since this file type is supported by base OpenCV, OpenCV is a required package for this project. It's also required due to the extensive use of the matrix class it provides through the code base and the use of utility functions such as "sum()". The DICOM and MPEG-4 Frame Capture classes share the same interface facing the main code base and are constructed in the same way allowing interchangeability dependant on the type of file being used for the project. Visually the two datasets are identical when displayed on the GUI. However when capturing frame data from the compressed DICOM files it was found that the rendering speed was slower than just reading from a normal video file as shown in figure 6.4. Both figure 6.2a and 6.2b show that the performance data is also correctly being recorded for any form of file capture and implementation type.

(A) MPEG Capture Performance

(B) DICOM Capture Performance

FIGURE 6.2: Performance Difference between DICOM and MPEG-4 Capture

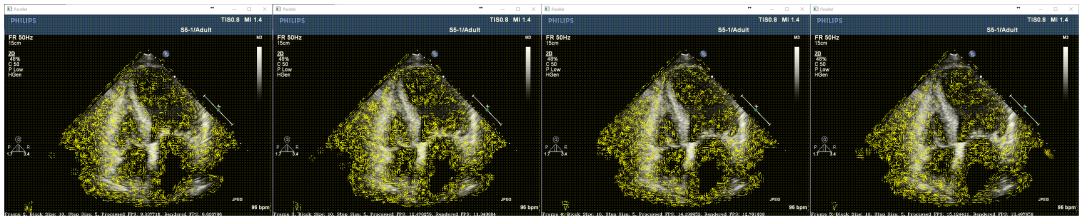
6.3 Motion Estimation Data and Visualisation

The FSBM motion estimation algorithm was initially implemented sequentially to later help speed up the development of the parallel implementation. It also presented some good areas for optimisation which are discussed in section 5.8.3. The main requirement of this section is that both the serial and parallel implementations could estimate motion effectively. They would pass their tests and requirements if they could calculate log the output of the motion vectors to a file in the form of angles and magnitude and to draw visualisations of the motion vector data over the

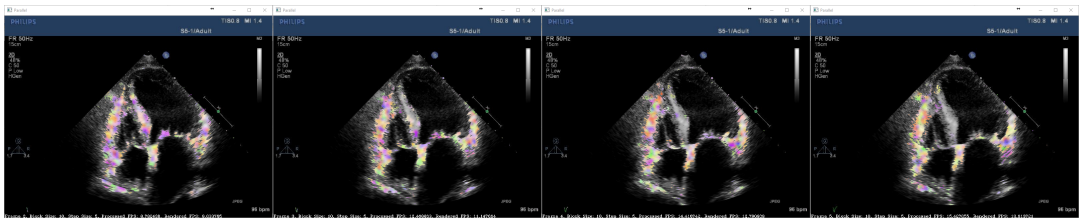
original images. The image visualisations happened in two stage, motion vector visualisation and angular motion visualisation. The first method plotted the motion vector displacement using arrows on an offset grid and the latter method visualised the angle and magnitude of the motion vectors as a colour component placed around the HSV colour wheel. Figure 6.3 shows actual data that was calculated for a DICOM image file and written to disk, while figures 6.4a and 6.4b show the drawn visualisation data of the motion vector, angles and magnitude respectively over the original data in parallel. The results of both the algorithms should be identical, since the sequential output was used as the baseline for the parallel. Figure 6.5 shows two identical motion vector results from sequential and parallel validating the motion estimation requirements.

```
1  Angle 0-360   Magnitude
2  53.016045     1.580787
3  54.175884     1.705664
4  52.252216     1.606699
5  49.206436     1.506651
6  ...           ...
```

FIGURE 6.3: Motion Vector Data written from File Writer Class



(A) Motion Vector Displacement



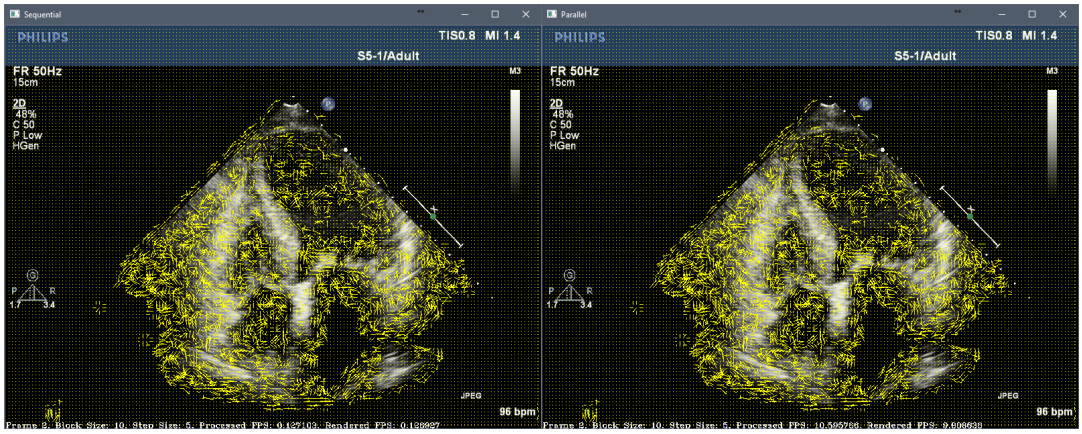
(B) Motion Vector HSV Angle

FIGURE 6.4: Sequence of Motion Vector Visualisation Techniques

6.4 BPM Estimation

The components of heart rate estimation from the motion vector data could have been implemented in the C++ code alongside the FSBM code. However to achieve this would require doing double the work in this project, since the implementation of an efficient Fast Fourier Transform algorithm in parallel could take weeks or months if developed alongside this application. Due to this constraint it was developed in MATLAB instead, while MATLAB isn't as fast as raw C code, the functions in the provided toolboxes are already rigorously tested and heavily optimised. Something that couldn't be achieved in the time frame of this project.

The requirements for the MATLAB script is that it could successfully evaluate the data. The evaluation metric used is accuracy to the original BPM data captured by



(A) Sequential (B) Parallel

FIGURE 6.5: Comparison of Parallel and Sequential Motion Vectors

the Ultrasound devices and stored in the DICOM data. To do this the frequency components were extracted from the angular motion data and the strongest frequency was extracted. The peak frequency of this data represents the number of frames one cardiac cycle occurs within, the definition of a single cardiac cycle is given in 2.5. The period of the frequency can be converted directly into BPM using the equation in 5.1. So for the data shown in 6.6 the peak frequency period of 31.44 translates to a BPM estimate of 96.19, calculated as $60000 \div (19.8390 \times 31.44)$ using information from figure C.1. This value was obtained for data with a ground truth BPM of 96, shown in C.1, so the results put it within 0.197721% accuracy to the original data and subsequently the requirements of BPM estimation were successfully met with an astounding end accuracy of almost the exact result. Since the value in the DICOM file is rounded, this level of accuracy is classified as a perfect result and possibly more accurate than the estimate given by the DICOM file.

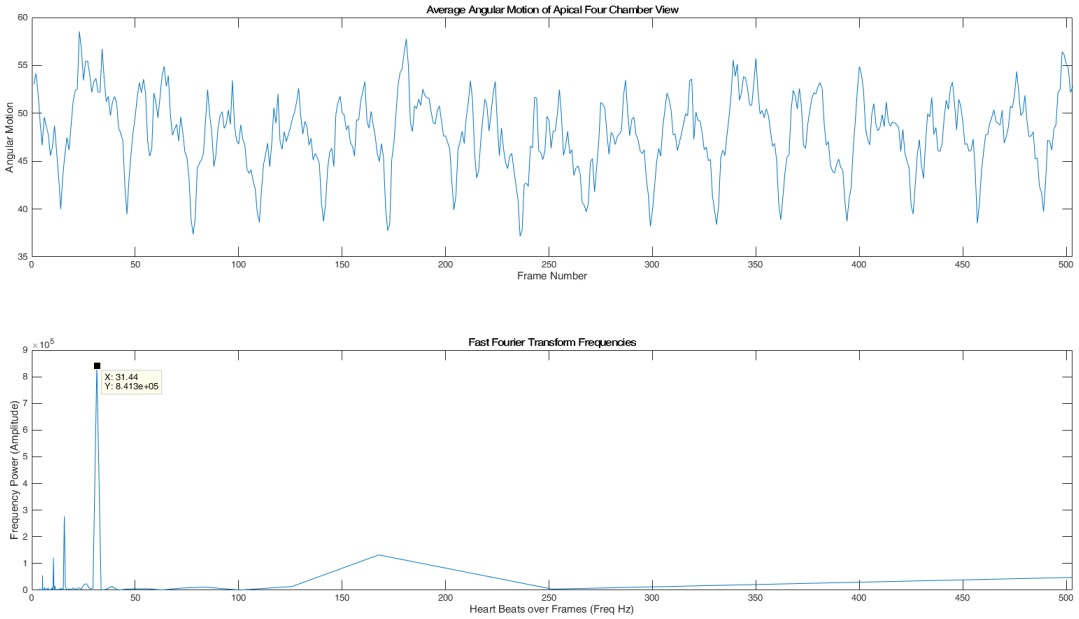
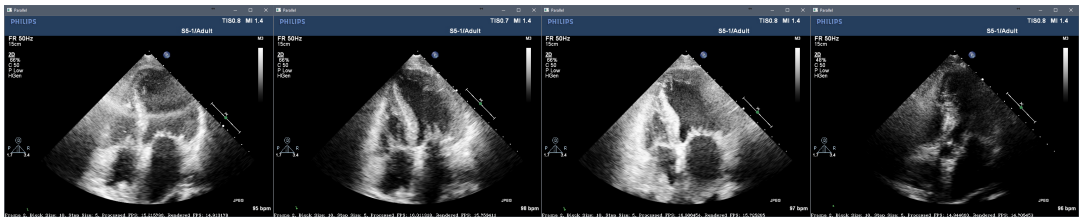


FIGURE 6.6: Component Frequencies of Motion Vector Data

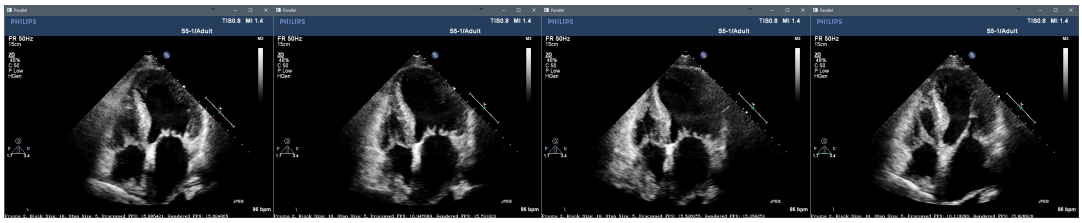
6.5 Evaluation

The final testing stages revealed that the serial and parallel implementations can successfully estimate the BPM of the ultrasound data from the DICOM files within an 0.197721% accuracy. While this result is within an astounding level of accuracy, this was obtained using the clearest recorded ultrasound data. Even though the ultrasound still had a lot of noise, it's inconclusive in the sense that it's a best case scenario for the data in this project. The following section will evaluate the estimation accuracy of all of the datasets used in this project. It will then use the accuracy results to evaluate the effectiveness of the parallel implementation, drawing conclusions from the performance data that contrasts the parallel and sequential implementations.

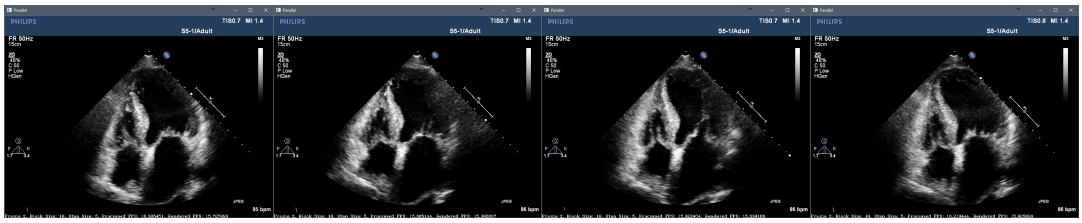
Overall there were twelve different datasets used in this project, for the sake of readability in this section they will be named 'DICOM-1' through to 'DICOM-12'. The relevant data needed for BPM estimation was extracted from each of the files and recorded in appendix C. The appendix shows the second frame of all of the datasets used in this project and figure 6.7 shows a short comparison of them all with links to the full images. The summary of the DICOM isn't in order, the first four images are DICOM file 1, 2, 3 and 7, the rest are in order from 4, 5, 6 to 8, 9, 10, 11 and 12. They are split into these two groups for one reason, at the start of the project when the DICOM files were initially analysed it became apparent that the DICOM files 1, 2, 3 and 7 were of a much lower capture quality. It was quickly realised that the motion estimation algorithms would produce lower quality representations of the overall motion due to the low contrast and variance between sections of the image. This is especially evident when looking at the mitral and tricuspid valves since they usually have a large component of vertical velocity in the image.



C.2 C.3 C.4 C.8



C.5 C.6 C.7 C.9



C.10 C.11 C.12 C.13

FIGURE 6.7: Quality Comparison of all DICOM dataset Images

The motion data around the heart wall and features of the heart seen in the ultrasounds of DICOM files 1, 2, 3, and 7 would be inaccurate due to the low variance regions polluting the neighbouring space especially in files 1 and 2. Due to this constraint it was initially expected that the BPM estimation later in the project would return poor results for these files. The other DICOM files have clear contrast between the ROIs in the images and the background making them ideal candidates for the FSBM motion estimation algorithm. The results of the BPM estimation are given in table 6.8 below, with a difference metric highlighting the accuracy of the estimated value.

FIGURE 6.8: BPM Estimation Accuracy of all Motion Vector Data Against Ground Truth Data

DICOM File	Frame Time	Ground Truth BPM	Estimated BPM	Difference
DICOM-1	19.8410	95	24.0480	119.19%
DICOM-2	19.8390	98	6.0126	176.87%
DICOM-3	19.8410	97	96.1922	0.8363%
DICOM-4	19.8410	96	96.1922	0.2000%
DICOM-5	19.8390	96	96.2019	0.2101%
DICOM-6	19.8390	96	96.2019	0.2101%
DICOM-7	19.8390	96	96.2019	0.2101%
DICOM-8	19.8390	96	96.1922	0.2000%
DICOM-9	19.8390	95	96.2019	1.2572%
DICOM-10	19.8390	96	96.2019	0.2101%
DICOM-11	19.8390	96	96.2019	0.2101%
DICOM-12	19.8410	96	96.1922	0.2000%

As predicted early in the project the DICOM files 1 and 2 performed the worst. The BPM estimates for these two files are over 100% over the ground truth data they should be. On further analysis of these two files it was found that the BPM was being successfully estimated but the motion data was corrupted with so much noise the components of frequency in the FFT graph had frequencies with a greater power amplitude than the cardiac cycle frequency as shown in figure 6.9. On manual inspection of the data shown in 6.9 the BPM can be estimated at 96.1845 and 96.1942 beats per minute respectively which are both less than 2% different to their respective ground truth BPM values. All of the other DICOM files have almost perfect estimations for BPM data, on average being at least 99.89% similar to their respective ground truth data, this clarifies that the FSBM motion estimation algorithm is successful and can be used to accurately measure heart rate in ultrasound sequences.

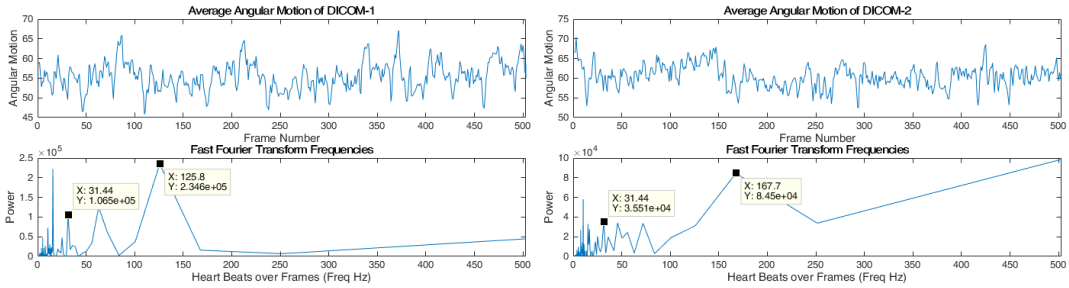
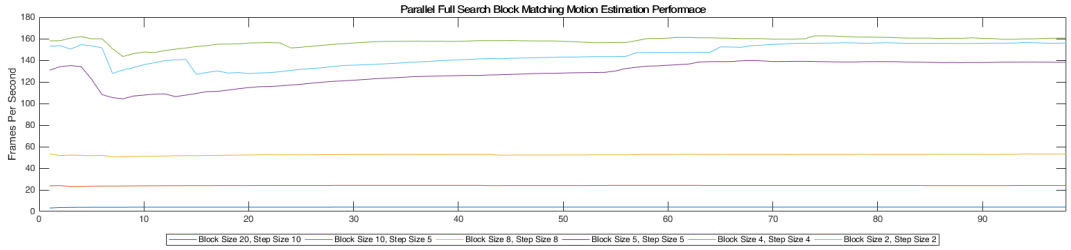


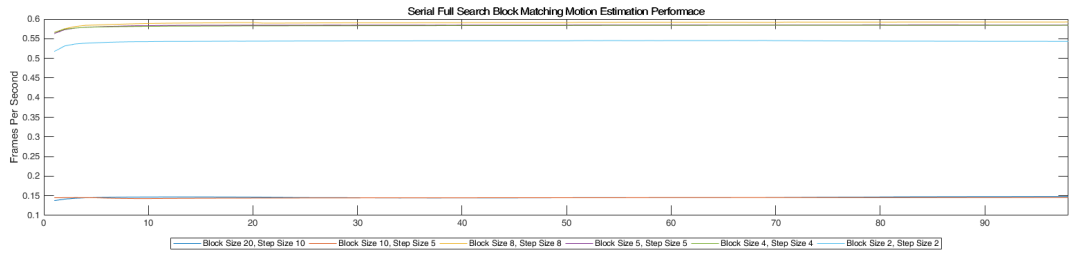
FIGURE 6.9: Incorrect Peak Frequencies of DICOM 1 and 2

The accuracy of the both the BPM and FSBM motion estimation algorithms could be improved in a variety of ways, the results achieved in this project conclude that the serial and parallel implementations are good enough in their estimates since the BPM estimates averagely only 0.2101% out of the original ground truth data, which would possibly be decreased if the DICOM files had recorded the nominal time between beats rather than just the BPM. Improving the estimate values for DICOM file 1 and 2 could have been achieved by smarter selection of the frequency components, since only the peak value is selected currently. The selection could instead be based on a range so that values that indicate in-human BPM values are ignored. As an example if values were weighted between a close to average human resting heart rate of 60-90 $\pm$ 30 (Aladin *et al.*, 2014) the estimates of DICOM file 1 and 2 would have discredited since their respective BPM estimates were 6.0126 and 24.0480 beats per minute. This however makes assumptions of the data which may not always be desirable. Another way they could have been improved is by implementing smarter motion estimation techniques such as lower weighting for low variance blocks to ignore parts of the image that are thought to not contain much information.

Now that the estimates of BPM have been found to be accurate, most of the deliverables have consequently been met, the next stage is to attempt to conclude the effectiveness of the parallelisation of traditionally serial algorithms. The algorithms used in this project were implemented without many optimisations so it was expected that it would be slow on the CPU. The nature of the GPU implementation was to retrieve estimates from BPM in a reasonable amount of time. The algorithms would be considered real-time if they could process the motion vectors faster than the rate of capture, in this case the rate of capture for all of the datasets is around 50Hz. This value was used to evaluate the effectiveness of the parallel implementations. Figure 6.10 below shows the performance of both implementations when they were ran with varying block and step sizes. In the implementations the block size is the size of the search block, the search window is the block size squared centred around the reference block and the step size is how the gap between blocks across the image.



(A) Parallel



(B) Serial

FIGURE 6.10: Performance in FPS of Parallel and Serial Implementations with Varying Block and Step Sizes

Figure 6.10 clearly shows that the parallel implementations performance is far greater than the sequential one. This was expected due to the processing power the GPU can provide. All of the data in 6.10 and 6.11 was obtained using a NVIDIA GeForce GTX 760 Graphics card (NVIDIA, 2017). Figure 6.11 shows a few permutations of block size and step size that were used in the benchmarking stages. The values were chosen on two criterion, the first is that the block size had to be a factor of the width and height of the input data so that the entire image could be processed in a grid. The second criteria is that the step size should be a factor of the block size so that it also stays within the bounds of the grid.

FIGURE 6.11: Serial and Parallel FSBM Performance

Block Size	Step Size	Serial FPS	Parallel FPS	Parallel Speed Increase
20	20	0.4916	4.7338	9.6293×
	10	0.1459	3.9902	27.348×
	5	0.1459	1.9970	13.687×
	4	0.1107	2.0306	18.323×
10	10	0.4789	29.744	62.109×
	5	0.1450	24.102	166.22×
8	8	0.5905	52.484	88.881×
	4	0.1358	39.846	448.58×
5	5	0.5840	128.22	219.55×
4	4	0.5829	157.71	270.56×
2	2	0.5435	145.06	266.90×

In figure 6.11 the largest block size that achieves real-time processing of the DICOM data is 8 with a step size of 8. These parameters were used to estimate the BPM of the DICOM-12 file to evaluate if the conversion to a parallel application from sequential could allow real-time BPM estimation of a data stream. Figure 6.12 shows that motion vector data from the parallel application with a block and step size of 8 results in the peak frequency having a period of 31.43. Since the "FrameTime" of the DICOM-12 file is 19.8410 this calculates a BPM estimate of 96.2151 using the BPM equation in 5.1. This is validation that the parallelisation of sequential cardiac techniques in image processing can help yield real-time results with speed ups of up to 448× being observed in this project.

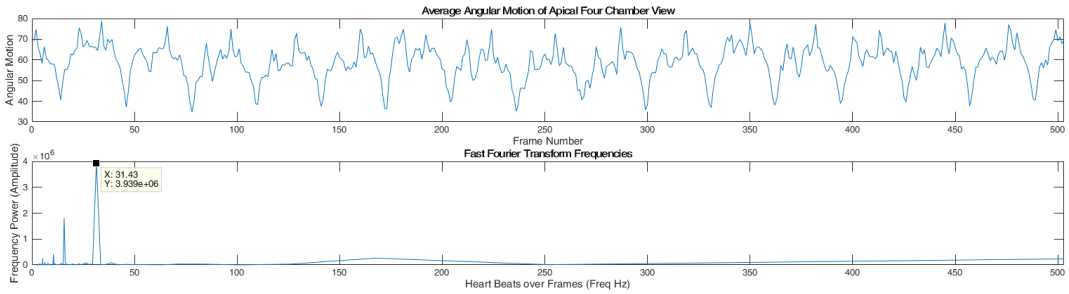


FIGURE 6.12: Real-Time BPM Estimation FFT Plot

6.6 Experimental Evaluation

This section will focus on implementing the novel optimisations discussed section 5.8.3. The optimisation is based around a novel cost function that can be exploited in the parallel kernel to find matching blocks around the candidate block. The constraints of the project have restricted the testing and evaluation of the method to draw conclusions to its accuracy to the SAD criterion. As aforementioned the bottleneck of SAD is that the cost criterion comes from element wise comparisons. When in parallel this exponentially increases the number of elements that have to be accessed slowing down the program execution considerably, especially when accessing the elements in global memory.

FIGURE 6.13: Experimental Parallel FSBM Performance

Block Size	Step Size	Parallel FPS	Experimental FPS	Experimental BPM
20	20	4.7338	112.29	96.2019
	10	3.9902	91.361	96.2019
	5	1.9970	52.582	96.2019
10	10	29.744	166.12	96.2019
	5	24.102	123.02	96.2019
8	8	52.484	184.36	96.2019
	4	39.846	139.40	96.2019

The new cost function computes the similarity of a comparison block in $2n + 1$ computations rather than $2n^2$ computations where n is the width of a square sliding window. This is because usually in SAD to compute the similarity of the reference and comparison block, the difference matrix has to be calculated which has to access every element of both matrices in memory and subtract them. In the new cost function this only has to be done once and every subsequent calculation only requires the computation of the non-overlapping values as shown in figure 5.21. For two 5 by 5 matrices this would save 39 computations compared to SAD. The requirement of this cost function is that the max value of the comparison matrix or max intensity level must be added to the reference matrix to force the sum to be greater than the comparison sum since the absolute difference is no longer calculated on a per pixel basis. Figure 6.13 shows the performance increase this method can achieve, fully exploiting features of parallel programming. It also illustrates that this cost function can successfully estimate BPM using the DICOM-12 file. The OpenCL kernel code for the experimental implementation can be seen in figure C.14.

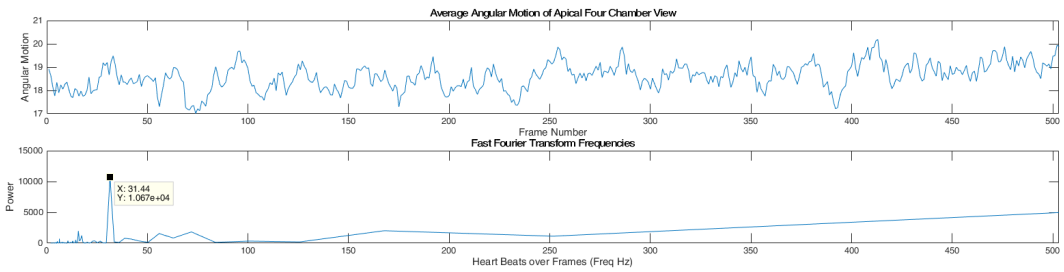


FIGURE 6.14: 184.36FPS BPM Estimation FFT Plot

Chapter 7

Reflective Analysis

Reflective processes are important for progression throughout the life cycle of projects, they help to refine and organise further development. Dybå *et al.* (2014) state that reflection on past practice is "important for continuous learning in software development. Reflection often takes place in cycles of experience followed by conscious application of learning from that experience, during which a software developer might explore comparisons, ponder alternatives, take diverse perspectives, and draw inferences, especially in new and/or complex situations". Within the project life cycle there are many situations where lessons learned can be applied to other areas to increase productivity or more efficiently manage resources. Many different models of reflection exist, such as Gibbs model of reflection which is based on Kolb's Learning Cycle (Clarke, 2013). Kolb's learning cycle consists of four stages for learning from experiences; the experience, the reflection on the experience, reflection analysis defining the experience and finally a stage where experimentation occurs using the newly derived ideas or concepts (Kolb, 2014). Gibbs model of reflection is based on this and promotes systematic thinking about project processes and stages that occur that can help improve future projects or processes that are undertaken. This section will outline effective parts of this project and the realisations that were made throughout it that would alter the future execution of project processes.

7.1 Reflection

The overall project execution happened without many drawbacks, although more consideration should have been given at the project outset to select more achievable goals. This project could have been easily converted into two separate projects in both the parallel and sequential paradigms. Although developing for both paradigms was a learning experience that showed the benefits and constraints of developing parallel or serial applications. Parallel applications can provide more performance for algorithm execution however they take longer to develop and test than serial implementations due the architecture of the parallel paradigm. Consideration was given to this point in the project and it was concluded that parallel programming is the most effective in computationally expensive programs. In some scenarios it makes more logical sense to implement serially as parallel programming requires much more set-up and in many cases specialised hardware and software such that of GPUs and OpenCL. Having to develop two separate algorithms and two separate applications placed a large proportion of the project resources on the development processes which didn't allow for any further development or experimentation with the algorithms or time to be allocated to more research in the area which could have increased the accuracy and performance of the applications.

Parallel programming as aforementioned is more time consuming than sequential programming especially when there is limited previous knowledge of the algorithms, for this reason the serial development of the algorithms were crucial to the parallel implementation. The development of the serial code helped understand and envision ways they code could be converted for parallel execution. Any future work will consider this when developing parallel code, since this project found the most effective method of design to be serial prototyping that can outline challenging areas of the algorithms for parallel execution. Consideration would also be given to algorithm selection for parallel programming in the future. The most important aspect of implementing ideas in parallel is trying to maintain a high level of data parallelism to fully exploit the performance programming on heterogeneous systems can provide. One of the unanticipated events that occurred in this project is the low performance of the serial code, this made it difficult to test and process large files. The serial code was benchmarked at around 0.15 frames per second in some situations which made processing minute long files take around five and a half hours.

7.2 Conclusion

Overall the project is considered to be a success, the end artefact not only fulfilled the project requirements, aim and objectives, but further experimented with the exploration of some optimisation strategies outside the bound of the project. This can be considered a starting point for future work, since limited testing was done to evaluate the accuracy of the extended functionality. The final base artefact builds two separate applications that are both capable of estimating the heart rate of apical four chamber view multi-frame ultrasound images in real-time. Real time processing of the data was achieved in the parallel implementation and could accurately predict the heart rate within 0.2% of the ground truth data. The underlying question in the aim of this research was whether GPU accelerated cardiac imaging techniques were effective in comparison to their serial implementation. From the results presented it's clear that the GPU provides much more performance for computationally expensive algorithms such that of the full search block matching algorithm and thus is largely more effective than the equivalent serial implementation.

7.3 Future Work

The short time frame of this project was the biggest constraint in the expansion of the project scope, many possible areas for future work and optimisations became apparent in the development phases. Future work in this project would focus mostly on the parallel implementation since the serial counterpart was found to be less effective in the analysis of ultrasound data in real-time. One aspect of the project as a whole that would be extended in future work would be to fully test and build the software on multiple different platforms and devices. The code base has been structured from the outset to allow for cross compiling, the libraries and code that has been developed are compatible across most platforms. Unfortunately, however time constraints restricted the deployment and testing of the software on multiple platforms. Similar to this the performance data and BPM estimation data recorded in this project is limited to only twelve datasets and one machine so for more conclusive results the software would be better benchmarked on an array of different machines and operating systems.

Other future work would consist of optimising the parallel kernel code to squeeze as much performance as possible out of the application. Attempts would be made to try and allow applications to generalise the approach so that multiple different types of analysis could be performed on the ultrasound data. Many methods of optimising the parallel code and the algorithms exist and have been deployed in current literature and future work could implement and benchmark these methods. One of the simpler ways of optimising the kernel code is to vectorise it to take advantage of the increase in performance of arithmetic of vectors in OpenCL. In example this would increase the number of blocks that could be compared to the reference block in FSBM at one time from one to four. Fundamentally this concept is old but has been proven to be effective especially in production code. Another fundamental concept of parallel programming and OpenCL that could be exploited in future work is taking advantage of the performance boost when accessing both local and constant memory over global. Reading and writing to the global buffers in the kernel is slow and utilising local memory can decrease the latency of memory access considerably.

In regards to the FSBM algorithm used the main area of optimisation would be to use different block matching algorithms that can ascertain a similar accuracy to FSBM. The main benefit of this is that the FSBM algorithm is exhaustive and especially in the case of the project a large portion of the input data contains information that's irrelevant to the final results making the extra computation of these areas unnecessary. Additionally using multiple GPUs to calculate the motion vectors of the images could be researched in future work to evaluate the scalability of the implementation. Removing as many dependencies from the application as possible is desirable to increase project portability. To achieve this future efforts could focus on implementing a Fast Fourier Transform algorithm in the C++ code to allow BPM estimation to occur while the data is being processed rather than pipe lining the data to external scripts after program execution.

Chapter 8

References

Aladin, A.I., Whelton, S.P., Al-Mallah, M., Blaha, M.J., Keteyian, S.J., Juraschek, S.P., Rubin, J., Brawner, C.A. and Michos, E.D. (2014) Relation of resting heart rate to risk for all-cause mortality by gender after considering exercise capacity (the henry ford exercise testing project). *American Journal of Cardiology*, 114(11) 1701-1706.

Allen, M.N. (1999) *Echocardiography.* : Lippincott.

Almasi, G.S. and Gottlieb, A. (1989) *Highly Parallel Computing.* Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc.

Apple (2017) *Xcode 8.* Online: Apple. Available from https://developer.apple.com/xcode/ [accessed 18th April 2017].

Attaway, S. (2016) *Matlab: A Practical Introduction to Programming and Problem Solving.* : Elsevier Science.

Axel, L. and Sodickson, D.K. (2014)*The Need for Speed*, .

B. Hoffman, D. Cole and J. Vines (2009) Software Process for Rapid Development of HPC Software Using CMake. In: *2009 DoD High Performance Computing Modernization Program Users Group Conference*, 378-382.

Baglietto, P., Maresca, M., Migliaro, A. and Migliardi, M. (1995) Parallel implementation of the full search block matching algorithm for motion estimation. In: *Application Specific Array Processors, 1995. Proceedings. International Conference on*: IEEE, 182-192.

Banas, C. (2016) *What media players support .mp4 files?* Quora. Available from https://www.quora.com/What-media-players-support-mp4-files; [accessed 8th April 2017].

Banger, R. and Bhattacharyya, K. (2013) *OpenCL programming by example.* : Packt Publishing Ltd.

Barjatya, A. (2004) Block matching algorithms for motion estimation. *IEEE Transactions Evolution Computation*, 8(3) 225-239.

Beck, Kent and Beedle, Mike and Van Bennekum, A and Cockburn, Alistair and Cunningham, Ward and Fowler, Martin and Grenning, James and Highsmith, Jim and Hunt, Andrew and Jeffries, Ron and others (2001) Agile manifesto. Http://www.Agilemanifesto.Org, .

Berger, H., Beynon-Davies, P. and Cleary, P. (2004) The utility of a rapid application development (RAD) approach for a large complex information systems development. *ECIS 2004 Proceedings*, 7.

Beynon-Davies, P., Carne, C., Mackay, H. and Tudhope, D. (1999) Rapid application development (RAD): An empirical review. *European Journal of Information Systems*, 8(3) 211-223.

Bidgood, J.,W.Dean, Horii, S.C., Prior, F.W. and Van Syckle, D.E. (1997) Understanding and using DICOM, the data interchange standard for biomedical imaging. *Journal of the American Medical Informatics Association*, 4(3) 199.

Bouguet, J. (2001) Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10) 4.

Bradski, G. and Kaehler, A. (2008) *Learning OpenCV: Computer vision with the OpenCV library.* : O'Reilly Media, Inc.

Chang, Y., Liang, D. and Ying, L. (2012) Nonlinear GRAPPA: A kernel approach to parallel MRI reconstruction. *Magnetic Resonance in Medicine*, 68(3) 730-740.

Chapman, S.J. (2016) *Essentials of MATLAB Programming.* : Cengage Learning.

Charvat, J. (2003) Project management methodologies. *New Jersey: John Willey \& Sons*, .

Chen, L., Chen, W., Jehng, Y. and Chiuch, T. (1991) An efficient parallel motion estimation algorithm for digital image processing. *IEEE Transactions on Circuits and Systems for Video Technology*, 1(4) 378-385.

Cheung, N.M., Au, O.C., Kung, M.C., Wong, P.H.W. and Liu, C.H. (2009) Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(11) 1692-1703.

Cheung, N.M., Fan, X., Au, O.C. and Kung, M.C. (2010) Video coding on multicore graphics processors. *IEEE Signal Processing Magazine*, 27(2) 79-89.

Choudhary, N.K., Ginjupalli, R., Navada, S. and Khanna, G. (2010) An exploration of OpenCL for a numerical relativity application. *ArXiv Preprint arXiv:1010.3816*, .

Christel, M. and Kang, K. (1992) *Issues in Requirements Elicitation.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Clarke, R.J. (2013)*Novel Aspects of a Training Program for Research Supervision*, .

Denning, P.J. and Tichy, W.F. (1990) Highly parallel computation.

DICOM Library. (2017) Online: DICOM Library. Available from http://www.dicomlibrary.com/dicom/transfer-syntax/ [accessed 21st April 2017].

Digital imaging and communications in medicine (DICOM). (1998) Washington, D.C.: National Electrical Manufacturers Association.

Dudin, D., Kunitskiy, S., Lamachenka, A., Khlud, M., Artsimenja, E., Kislova, M., Melnikov, Y., Smirnova, Y., Kukhnavets, P., Dubovik, K. and Lamachenka, A. (2017) *Online Gantt chart software for project management* [app/game/software]. Version 1. Online : Online.

Eichelberg, M., Onken, M. and Thiel, A. (2014)*OFFIS DCMTK-DICOM Toolkit*, .

Elliott, S. (2008) Agile project management. In: *Seminar on Current Trends in Software Industry. University of Helsinki, Finland*.

Elmoataz, A., Lezoray, O., Mammass, D. and Nouboud, F. (2008) *Image and Signal Processing: 3rd International Conference, ICISP 2008, Cherbourg-Octeville, France, July 1-3, 2008, Proceedings.* : Springer.

Engert, P.A. and Lansdowne, Z.F. (1999) Risk matrix User's guide. *Bedford, MA: The MITRE Corporation*, .

FreeSoftwareFoundation (2017) *GNU Make.* Online: Free Software Foundation. Available from https://www.gnu.org/software/make/ [accessed 18th April 2017].

Fulgham, B. and Gouy, I. (2009) *Which programs are fast?* Online. Available from https://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.html [accessed 17th April 2017].

Garvey, P.R. and Lansdowne, Z.F. (1998) Risk matrix: An approach for identifying, assessing, and ranking program risks. *Air Force Journal of Logistics*, 22(1) 18-21.

Ghanbari, M. (1990) The cross-search algorithm for motion estimation (image coding). *IEEE Transactions on Communications*, 38(7) 950-953.

Gilat, A. (2014) *MATLAB: An Introduction with Applications, 5th Edition.* : Wiley Global Education.

Git (2017) *Git.* Online: GitHub. Available from https://github.com/git/ [accessed 17th April 2017].

GitHub (2017) *GitHub.* Online: GitHub. Available from https://github.com [accessed 17th April 2017].

Gousios, G., Vasilescu, B., Serebrenik, A. and Zaidman, A. (2014) Lean GHTorrent: GitHub data on demand. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*: ACM, 384-387.

Higham, D.J. and Higham, N.J. (2016) *MATLAB Guide, Third Edition:* .

Hwu, W. and Kirk, D. (2009) Programming massively parallel processors. *Special Edition,* 92.

JetBrains (2017) *CLion: A cross-platform IDE for C and C++.* Online: JetBrains. Available from https://www.jetbrains.com/clion/ [accessed 17th April 2017].

Jin, G. and Lee, H. (2006) A parallel and pipelined execution of H. 264/AVC intra prediction. In: *Computer and Information Technology, 2006. CIT'06. the Sixth IEEE International Conference on*: IEEE, 246-246.

Johnson, D. (2010) Fast fourier transform (FFT). *Connexions, June,* 20 2010.

Joshi, P., Escrivá, D.M. and Godoy, V. (2016) *OpenCV By Example.* : Packt Publishing Ltd.

Karlesky, M. and Vander Voord, M. (2008) Agile project management. *Esc,* 247(267) p4.

Kerzner, H. (2004) *Advanced project management: Best practices on implementation.* : John Wiley \& Sons.

Kerzner, H. (2013) *Project management: a systems approach to planning, scheduling, and controlling.* : John Wiley \& Sons.

Khan, A.I., Qurashi, R.J. and Khan, U.A. (2011) A comprehensive study of commonly practiced heavy and light weight software methodologies. *ArXiv Preprint arXiv:1111.3001,* .

Khokhar, A.A., Prasanna, V.K., Shaaban, M.E. and Wang, C. (1993) Heterogeneous computing: Challenges and opportunities. *Computer,* 26(6) 18-27.

KhronosGroup (2011) *OpenCL- The open standard for parallel programming of heterogeneous systems.* Online: Khronos Group. Available from https://www.khronos.org/opencl/ [accessed 9th April 2017].

Kolb, D.A. (2014) *Experiential learning: Experience as the source of learning and development.* : FT press.

Kumar, R., Marinov, D., Padua, D., Parthasarathy, M., Patel, S., Roth, D., Snir, M. and Torrellas, J. (2008) Parallel computing research at illinois the UPCRC agenda.

Kung, M., Au, O.C., Wong, P. and Liu, C.H. (2008) Block based parallel motion estimation using programmable graphics hardware. In: *Audio, Language and Image Processing, 2008. ICALIP 2008. International Conference on*: IEEE, 599-603.

Kwon, S., Tamhankar, A. and Rao, K. (2006) Overview of H. 264/MPEG-4 part 10. *Journal of Visual Communication and Image Representation,* 17(2) 186-216.

Lam, A. and Kuno, Y. (2015) Robust heart rate measurement from video using select random patches. In: *Proceedings of the IEEE International Conference on Computer Vision*, 3640-3648.

Liu, L., Chien, J., Chuang, H. and Li, C. (2003) A frame-level FSBM motion estimation architecture with large search range. In: *Advanced Video and Signal Based Surveillance, 2003. Proceedings. IEEE Conference on*: IEEE, 327-333.

Mackay, H., Carne, C., Beynon-Davies, P. and Tudhope, D. (2000) Reconfiguring the user: Using rapid application development. *Social Studies of Science*, 30(5) 737-757.

Management Association, I.R. (2013) *Software Design and Development: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications.* : IGI Global.

Martin, J. (1991) *Rapid application development.* : Macmillan Publishing Co., Inc.

Martini, F. (2007) *Anatomy and Physiology' 2007 Ed.2007 Edition.* : Rex Bookstore, Inc.

Massanes, F., Cadennes, M. and Brankov, J.G. (2011) Compute-unified device architecture implementation of a block-matching algorithm for multiple graphical processing unit cards. *Journal of Electronic Imaging*, 20(3) 033004-033004.

MathWorks (2017) *The Language of Technical Computing.* Online: MathWorks. Available from https://uk.mathworks.com/products/matlab.html [accessed 17th April 2017].

Maylor, H. (2001) Beyond the gantt chart:: Project management moving on. *European Management Journal*, 19(1) 92-100.

Mehta, S., Misra, A., Singhal, A., Kumar, P. and Mittal, A. (2010) A high-performance parallel implementation of sum of absolute differences algorithm for motion estimation using CUDA. In: *HiPC Conf*, 6.

Metkar, S. and Talbar, S. (2013) Performance Evaluation of Block Matching Algorithms for Video Coding. In: *Motion Estimation Techniques for Digital Video Coding.* India: Springer India, 13-31.

Microsoft (2017) *Visual Studio 2017.* Online: Microsoft. Available from https://www.visualstudio.com/en-us/news/releasenotes/vs2017-relnotes [accessed 17th April 2017].

Munns, A. and Bjeirmi, B. (1996) The role of project management in achieving project success. *International Journal of Project Management*, 14(2) 81.

Munshi, A., Gaster, B., Mattson, T.G. and Ginsburg, D. (2011) *OpenCL Programming Guide.* : Pearson Education.

Netzer, R.H. and Miller, B.P. (1992) What are race conditions?: Some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 1(1) 74-88.

NVIDIA (2017) *GeForce GTX 760.* Online: NVIDIA. Available from http://www.geforce.co.uk/hardware/desktop-gpus/geforce-gtx-760 [accessed 26th April 2017].

OpenCV (2017) *About.* Online: OpenCV. Available from http://opencv.org/about.html [accessed 10th April 2017].

OpenCV (2017) *cv::VideoCapture Class Reference.* Online: OpenCV. Available from http://docs.opencv.org/3.1.0/d8/dfe/classcv_1_1VideoCapture.html [accessed 21st April 2017].

OpenCV (2017) *Drawing Functions.* Online: OpenCV. Available from http://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html [accessed 23nd April 2017].

Pacheco, P.S. (2011) *An Introduction to Parallel Programming.* : Morgan Kaufmann.

Padua, D. (2011) *Encyclopedia of parallel computing.* : Springer Science and Business Media.

Pianykh, O. (2012) *Digital Imaging and Communications in Medicine (DICOM).* 1st edition. Berlin, Heidelberg: Springer Berlin Heidelberg.

Po, L. and Ma, W. (1996) A novel four-step search algorithm for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3) 313-317.

Rabaey, J.M., Chandrakasan, A.P. and Nikolic, B. (2002) *Digital integrated circuits.* : Prentice hall Englewood Cliffs.

Rao, K.N., Naidu, G.K. and Chakka, P. (2011) A study of the agile software development methods, applicability and implications in industry. *International Journal of Software Engineering and its Applications,* 5(2) 35-45.

Robins, A., Rountree, J. and Rountree, N. (2003) Learning and teaching programming: A review and discussion. *Computer Science Education,* 13(2) 137-172.

Rodgers, D.P. (1985) Improvements in multiprocessor system design. *SIGARCH Comput.Archit.News,* 13(3) 225-231.

Saha, A., Mukherjee, J. and Sural, S. (2011) A neighborhood elimination approach for block matching in motion estimation. *Signal Processing: Image Communication,* 26(8) 438-454.

Samadi, M., Jamshidi, D.A., Lee, J. and Mahlke, S. (2014) Paraprox: Pattern-based approximation for data parallel applications. In: *ACM SIGARCH Computer Architecture News*: ACM, 35-50.

Seferidis, V.E. and Ghanbari, M. (1993) General approach to block-matching motion estimation. *Optical Engineering,* 32(7) 1464-1474.

Shen, J.P. and Lipasti, M.H. (2013) *Modern Processor Design: Fundamentals of Superscalar Processors.* : Waveland Press.

Smith, P. (2011) *Software build systems: principles and experience.* : Addison-Wesley Professional.

Stroustrup, B. (2007) Evolving a language in and for the real world: C 1991-2006. In: *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*: ACM, 4-1.

Stroustrup, B., (1986) *The C++ programming language.* Reading, Mass.: Addison-Wesley.

T. Dybå, N. Maiden and R. Glass (2014) The reflective software engineer: Reflective practice. *IEEE Software,* 31(4) 32-36.

Tankariya, A., Singh, J. and Tiwari, M. (2011) A review on comparative analysis of block matching algorithm.

Trigkas, A. (2014) Investigation of the OpenCL SYCL programming model.

Westland, J. (2007) *The Project Management Life Cycle: A Complete Step-by-step Methodology for Initiating, Planning, Executing \& Closing a Project Successfully.* : Kogan Page.

Xu, J. (2008) OpenCL--the open standard for parallel programming of heterogeneous systems.

Yang, C., Cheung, G. and Stankovic, V. (2015) Estimating heart rate via depth video motion tracking. In: *Multimedia and Expo (ICME), 2015 IEEE International Conference on*: IEEE, 1-6.

Yi, X., Zhang, J., Ling, N. and Shang, W. (2005) Improved and simplified fast motion estimation for JM. *Jvt-p021,* 18.

Zahiri-Azar, R. and Salcudean, S.E. (2006) Motion estimation in ultrasound images using time domain cross correlation with prior estimates. *IEEE Transactions on Biomedical Engineering,* 53(10) 1990-2000.

Zhang, J., Nezan, J. and Cousin, J. (2012) Implementation of motion estimation based on heterogeneous parallel computing system with opencl. In: *High Performance Computing and Communication \& 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*: IEEE, 41-45.

Zhu, S. and Ma, K. (1997) A new diamond search algorithm for fast block matching motion estimation. In: *Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on*: IEEE, 292-296.

Appendix A

Project Management

A.1 Risk Matrix

#	Risk	Risk Description	Probability 0-1	Severity 1-10	Quotient (P x S)	Consequence	Reducing Risk
1	Provided datasets are an example of a perfect ultrasound acquisition.	The datasets that will be used in testing and benchmarking could be perfect, in the way that they have no added noise or features that would affect the application of the project externally.	0.2	4	0.8	Externally this project would have very little relevance as the overall aim of achieve cardiac tracking in real-time using parallel computing techniques would only be successful with perfect image acquisition. Therefore negating the reasoning for being real-time.	Steps will be taken to try to get as large of a dataset as possible with varying features contained within the images.
2	Unrealistic timeframes within planning.	Within the Gantt chart tasks have been allocated a timeframe in which they should be completed, this risk considers the event where not enough time has been allocated	0.6	5	3	If a task has been allocated a shorter timeframe than what will be required to complete it, it will subtract time from the future tasks or push back the completion date.	Within the Gantt chart some time will be allocated called slack time which can be used in these situations.
3	Not having enough experience or knowledge to produce a working system.	This project focuses on two distinct areas, parallel computing and image processing. Both fields haven't been taught at university yet which may increase the difficulty in creating the desired system.	0.4	8	3.2	Objective three and four will not be met because the implementations of image processing algorithms will not exist or will be implemented poorly. The parallel algorithms will not be effective in providing a performance gain.	A larger portion of time will be dedicated to learning the dependant skills. Which will reduce the probability of this risk occurring.
4	Not having abundant resources since parallel computing regarding image processing is a relatively new field.	Since parallel computing hasn't been around for as long as some other disciplines within computer science it's possible the quantity of documentation or relevant literature will be sparse in comparison resulting in longer time frames for tasks and objectives.	0.5	7	3.5	Problems that would usually be solved by referencing the appropriate documentation will take longer to solve which by extension will make the time frames for the problem tasks/objectives longer.	Within the project techniques that are well documented will be preferred over other techniques to maximise the project efficiency.
5	Getting unrealistic benchmarks dependant on the hardware.	Due to the nature of parallel computing it is possible that the benchmarks could portray inaccurate results based on the machine they were ran on.	0.6	10	6	The evaluation objective of this project won't be justified in the results it presents. This may affect the conclusion that is drawn from the project.	Optimising the hardware to match the techniques used will reduce this risk significantly.

FIGURE A.1: Initial Project Risk Matrix

A.2 Gantt Chart

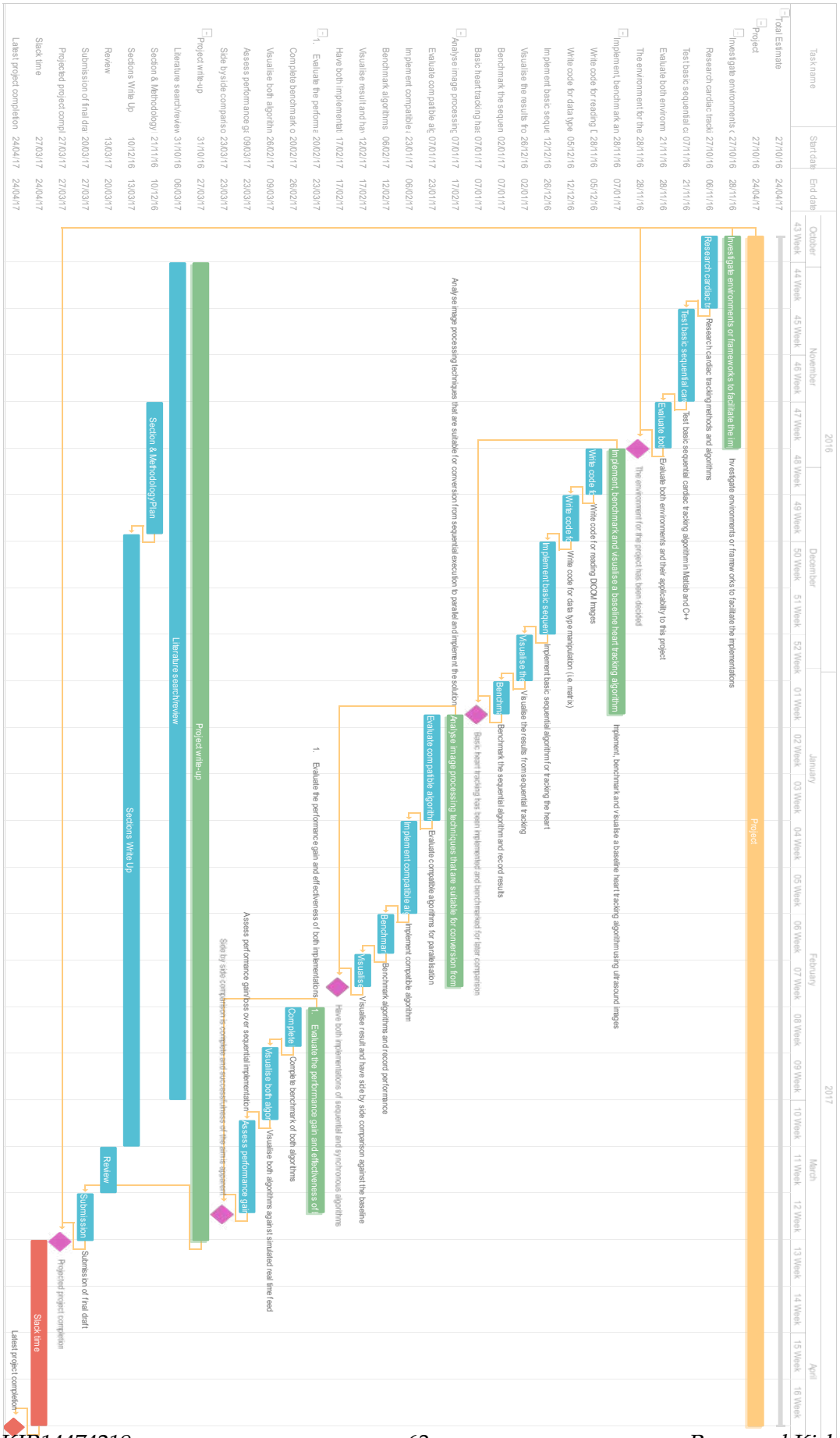


FIGURE A.2: Initial Project Gantt Chart Created with GanttPRO (Dudin *et al.*, 2017)

Appendix B

Development

B.1 Image Capture

```
FileModDate: '10-Dec-2016 19:00:38'
FileSize: 107757616
Format: 'DICOM'
FormatVersion: 3
Width: 800
Height: 600
BitDepth: 8
ColorType: 'truecolor'
TransferSyntaxUID: '1.2.840.10008.1.2.4.50'
ImageType: 'DERIVED\PRIMARY\CARDIOLOGY'
Modality: 'US'
DerivationDescription: '96'
ProtocolName: 'Free Form'
FrameTime: 19.8390
HeartRate: 96
SeriesNumber: 1
InstanceNumber: 63
SamplesPerPixel: 3
PhotometricInterpretation: 'YBR_FULL_422'
PlanarConfiguration: 0
NumberOfFrames: 505
Rows: 600
Columns: 800
UltrasoundColorDataPresent: 0
BitsAllocated: 8
BitsStored: 8
HighBit: 7
PixelRepresentation: 0
```

FIGURE B.1: DICOM Image File Extracted Information

B.2 Serial FSBM Code

```
1  inline float square(float x) {
2      return x * x;
3  }
4
5  inline int AbsoluteDifference(int a, int b) {
6      return a < b ? b - a : a - b;
7  }
8  inline float euclideanDistance(int x2, int x1, int y2, int y1) {
9      return sqrt((float)(square(x2 - x1) + square(y2 - y1)));
10 }
11
12 inline int MatrixSum(cv::Mat img, cv::Point p, int blockSize) {
13     return cv::sum(cv::abs(img(cv::Rect(p.x, p.y, blockSize,
       ↪  blockSize))))[0];
14 }
15
16 inline int MatrixSAD(const cv::Mat& curr, const cv::Mat& ref, const
   ↪  cv::Point& currPoint, const cv::Point& refPoint, const int& blockSize)
   ↪  {
17     return cv::sum(cv::abs(
18         curr(cv::Rect(currPoint.x, currPoint.y, blockSize, blockSize)) -
19         ref(cv::Rect(refPoint.x, refPoint.y, blockSize, blockSize))
20     ))[0];
21 }
22
23 inline bool IsInBounds(int x, int y, int width, int height, int bSize) {
24     return y >= 0 && y < height - bSize && x >= 0 && x < width - bSize;
25 }
26
27 cv::Vec3i ClosestNeighbour(const cv::Mat& prev, const cv::Point&
   ↪  currPoint, const int sWindow, const int width, const int height, const
   ↪  int blockSize) {
28     for (int row = -sWindow; row < sWindow; row += sWindow) {
29         for (int col = -sWindow; col < sWindow; col += sWindow) {
30             cv::Point refPoint(currPoint.x + row, currPoint.y + col);
31             if (IsInBounds(refPoint.x, refPoint.y, width, height,
                ↪  blockSize)) {
32                 return cv::Vec3i(row, col, MatrixSum(prev, refPoint,
                    ↪  blockSize));
33             }
34         }
35     }
36
37     cv::Point refPoint(currPoint.x, currPoint.y);
38     return cv::Vec3i(0, 0, MatrixSum(prev, refPoint, blockSize));
39 }
```

FIGURE B.2: Serial FSBM Motion Estimation Utility Functions

```
1  void FullExhastiveSAD(cv::Mat& curr, cv::Mat& ref, cv::Point*
   ↪  &motionVectors, cv::Point2f* &motionDetails, int blockSize, int
   ↪  stepSize, int width, int height, int wB, int hB) {
2      for (int x = 0; x < wB; x++) {
3          for (int y = 0; y < hB; y++) {
4              const cv::Point currPoint(x * stepSize, y * stepSize);
5              int idx = x + y * wB;
6
7              const int sWindow = blockSize;
8              cv::Vec3i closest = ClosestNeighbour(ref, currPoint, sWindow,
               ↪  width, height, blockSize);
9
10             float distanceToBlock = FLT_MAX;
11             int bestErr = INT_MAX, err;
12
13             for (int row = closest[0]; row < sWindow; row++) {
14                 for (int col = closest[1]; col < sWindow; col++) {
15                     cv::Point refPoint(currPoint.x + row, currPoint.y +
                       ↪  col);
16
17                     if (IsInBounds(refPoint.x, refPoint.y, width, height,
                       ↪  blockSize)) {
18
19                         err = MatrixSAD(curr, ref, currPoint, refPoint,
                           ↪  blockSize);
20
21                         float newDistance = euclideanDistance(refPoint.x,
                           ↪  currPoint.x, refPoint.y, currPoint.y);
22
23                         if (err < bestErr || (err == bestErr &&
                           ↪  newDistance <= distanceToBlock)) {
24                             bestErr = err;
25                             distanceToBlock = newDistance;
26                             float p0x = currPoint.x, p0y = currPoint.y -
                               ↪  sqrt((float)(square(refPoint.x - p0x) +
                               ↪  square(refPoint.y - currPoint.y)));
27                             float angle = (2 * atan2(refPoint.y - p0y,
                               ↪  refPoint.x - p0x)) * 180 / M_PI;
28                             motionVectors[idx] = refPoint;
29                             motionDetails[idx] = cv::Point2f(angle,
                               ↪  distanceToBlock);
30                         }
31                     }
32                 }
33             }
34         }
35     }
36 }
```

FIGURE B.3: Serial FSBM Motion Estimation Code using SAD

B.3 Parallel FSBM Code

```
1  inline int square(int x) {
2          return x * x;
3  }
4
5  inline float euclidean_distance(int x2, int x1, int y2, int y1) {
6          return sqrt((float)(square(x2 - x1) + square(y2 - y1)));
7  }
8
9  inline bool is_in_bounds(int x, int y, int width, int height, int bSize) {
10         return y >= 0 && y < height - bSize && x >= 0 && x < width -
           ↪ bSize;
11 }
12
13 inline int absolute_difference(int a, int b) {
14         return a < b ? b - a : a - b;
15 }
16
17 int sum_absolute_diff(image2d_t curr, image2d_t ref, int2 currPoint, int2
   ↪ refPoint, int bSize) {
18     int sum = 0;
19     for (int i = 0; i < bSize; i++)
20     {
21         for (int j = 0; j < bSize; j++)
22         {
23             sum += absolute_difference(
24                 read_imageui(curr, sampler, (int2)(currPoint.x + i,
                    ↪ currPoint.y + j)).x,
25                 read_imageui(ref, sampler, (int2)(refPoint.x + i,
                    ↪ refPoint.y + j)).x
26             );
27         }
28     }
29 }
30
31 int3 closest_inbound_neighbour(image2d_t prev, int2 currPoint, const int
   ↪ sWindow, int width, int height, int blockSize) {
32     for (int row = -sWindow; row < sWindow; row += sWindow) {
33         for (int col = -sWindow; col < sWindow; col += sWindow) {
34             int2 refPoint = { currPoint.x + row, currPoint.y + col };
35             if (is_in_bounds(refPoint.x, refPoint.y, width, height,
                ↪ blockSize)) {
36                 return (int3)(row, col, matrix_sum(prev, refPoint,
                    ↪ blockSize, 0));
37             }
38         }
39     }
40 }
```

FIGURE B.4: OpenCL utility Functions for FSBM Motion Estimation

```
__kernel void full_exhastive_SAD(
    __read_only image2d_t prev,
    __read_only image2d_t curr,
    const uint step_size,
    const uint blockSize,
    uint width,
    uint height,
    __global int2 * motionVectors,
    __global float2 * motionDetails
)
{
    const int x = get_global_id(0);
    const int y = get_global_id(1);
    const int2 currPoint = { x * step_size, y * step_size };

    const int wB = get_global_size(0);
    int idx = x + y * wB;

    const int sWindow = blockSize;
    float distanceToBlock = FLT_MAX;
    float bestErr = FLT_MAX, err;

    for (int row = -sWindow; row < sWindow; row++) {
        for (int col = -sWindow; col < sWindow; col++) {
            int2 refPoint = { currPoint.x + row, currPoint.y + col };

            if (is_in_bounds(refPoint.x, refPoint.y, width, height,
            ↪  blockSize)) {
                err = sum_absolute_diff(curr, prev, currPoint, refPoint,
                ↪  blockSize);

                float newDistance = euclidean_distance(refPoint.x,
                ↪  currPoint.x, refPoint.y, currPoint.y);

                if (err < bestErr || (err == bestErr && newDistance <=
                ↪  distanceToBlock)) {
                    bestErr = err;
                    distanceToBlock = newDistance;
                    float p0x = currPoint.x, p0y = currPoint.y -
                    ↪  sqrt((float)(square(refPoint.x - p0x) +
                    ↪  square(refPoint.y - currPoint.y)));
                    float angle = (2 * atan2(refPoint.y - p0y, refPoint.x
                    ↪  - p0x)) * 180 / M_PI;
                    motionVectors[idx] = refPoint;
                    motionDetails[idx] = (float2)(angle, distanceToBlock);
                }
            }
        }
    }
}
```

FIGURE B.5: OpenCL FSBM Motion Estimation Kernel Code using SAD

Appendix C

Evaluation

C.1 Extracted DICOM Data

```
DICOM-*:
    Width:          800
    Height:         600
    BitDepth:       8
    NumberOfFrames: 505
    FrameTime: 19.8390
DICOM-1:
    HeartRate: 95
    FrameTime: 19.8410
DICOM-2:
    HeartRate: 98
DICOM-3:
    HeartRate: 97
    FrameTime: 19.8410
DICOM-4:
    HeartRate: 96
    FrameTime: 19.8410
DICOM-5:
    HeartRate: 96
DICOM-6:
    HeartRate: 96
DICOM-7:
    HeartRate: 96
DICOM-8:
    HeartRate: 96
DICOM-9:
    HeartRate: 95
DICOM-10:
    HeartRate: 96
DICOM-11:
    HeartRate: 96
DICOM-12:
    HeartRate: 96
    FrameTime: 19.8410
```

FIGURE C.1: DICOM Image File Information Extracted for All Twelve Datasets

C.2 DICOM Screen Capture

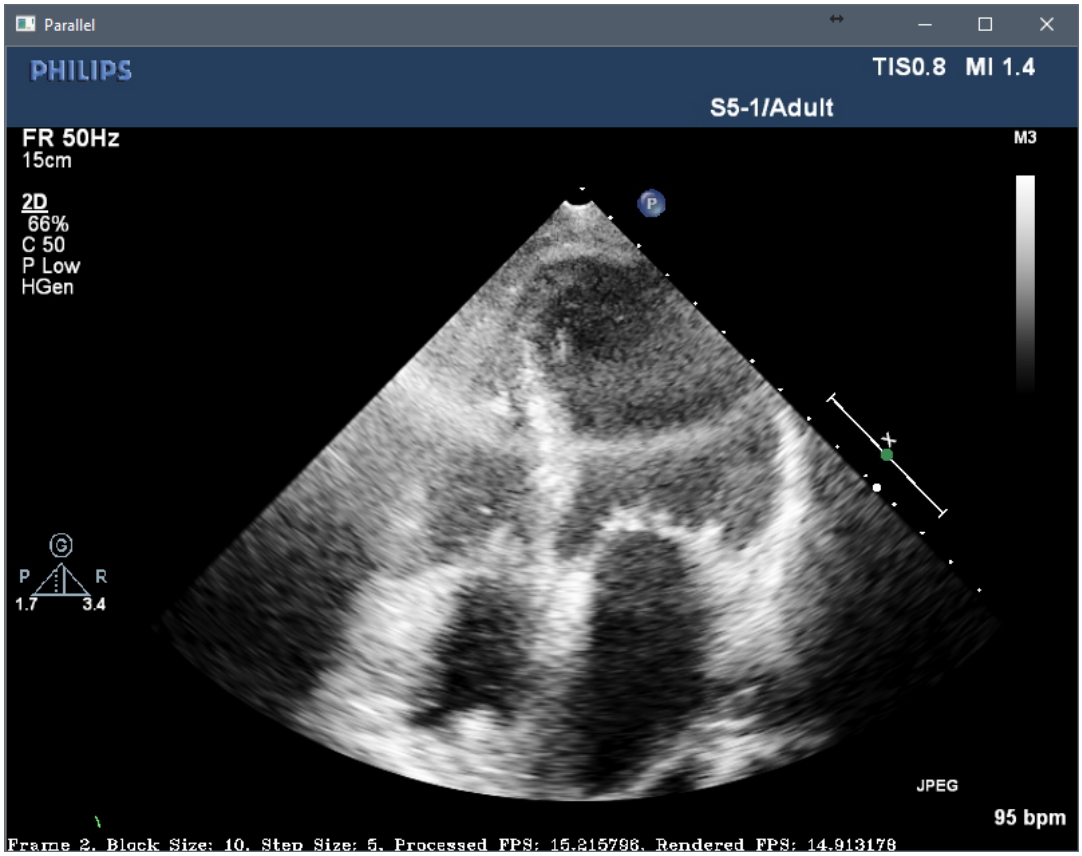


FIGURE C.2: DICOM-1

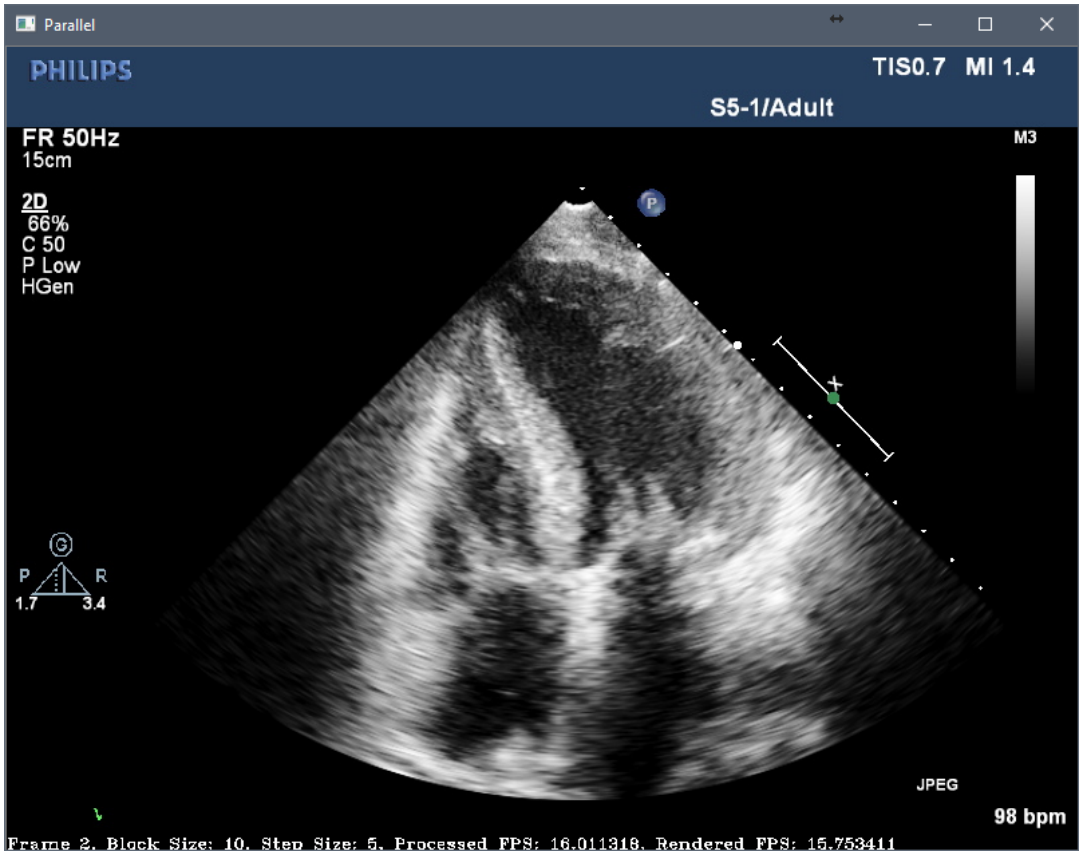


FIGURE C.3: DICOM-2

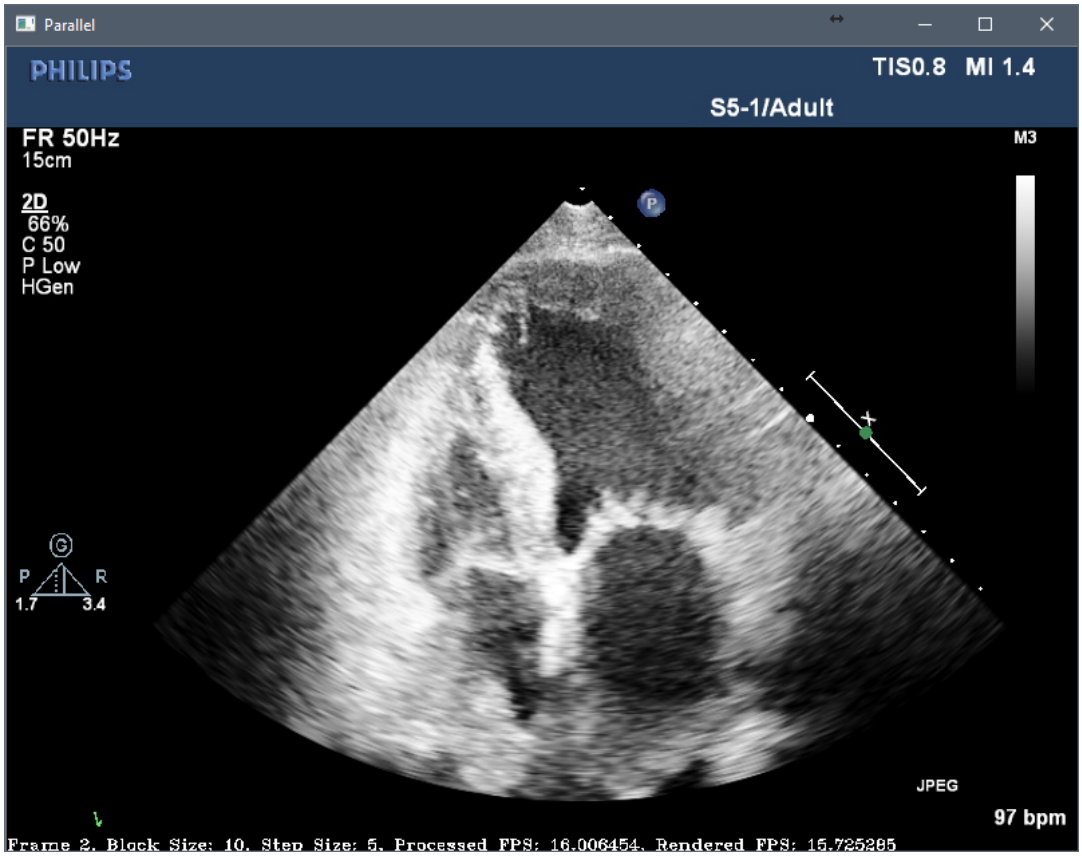


FIGURE C.4: DICOM-3

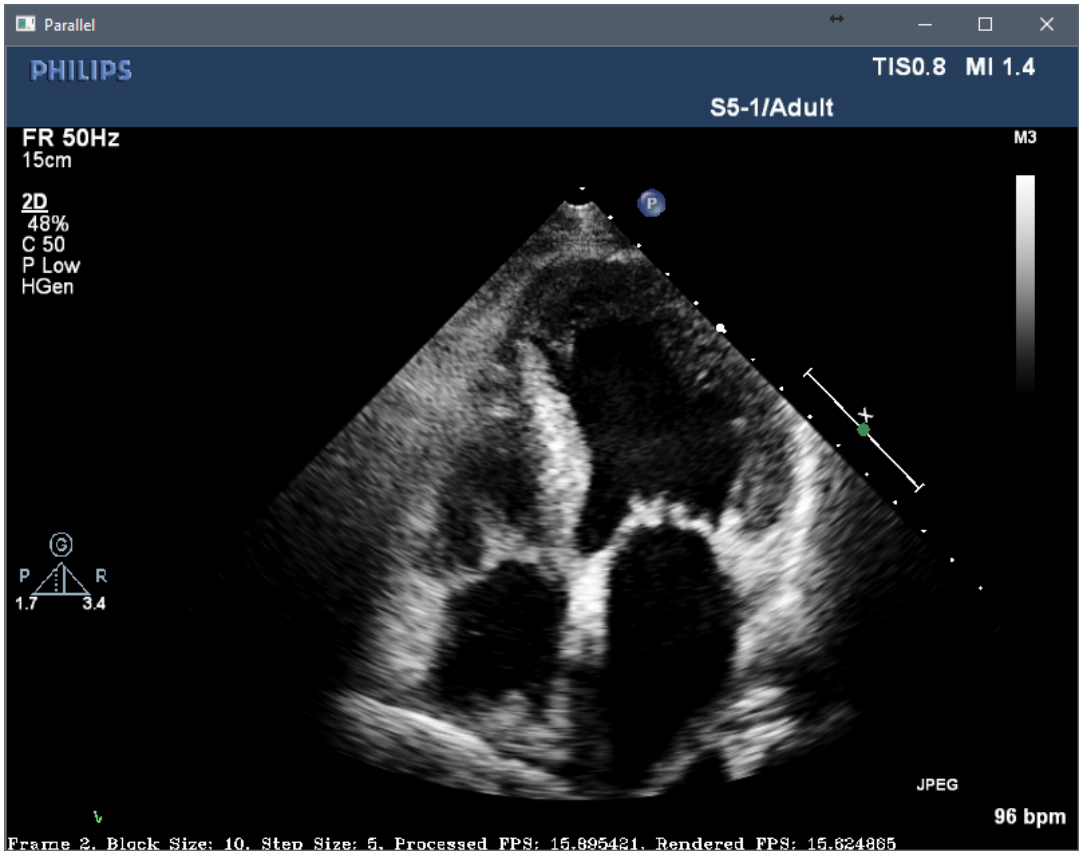


FIGURE C.5: DICOM-4

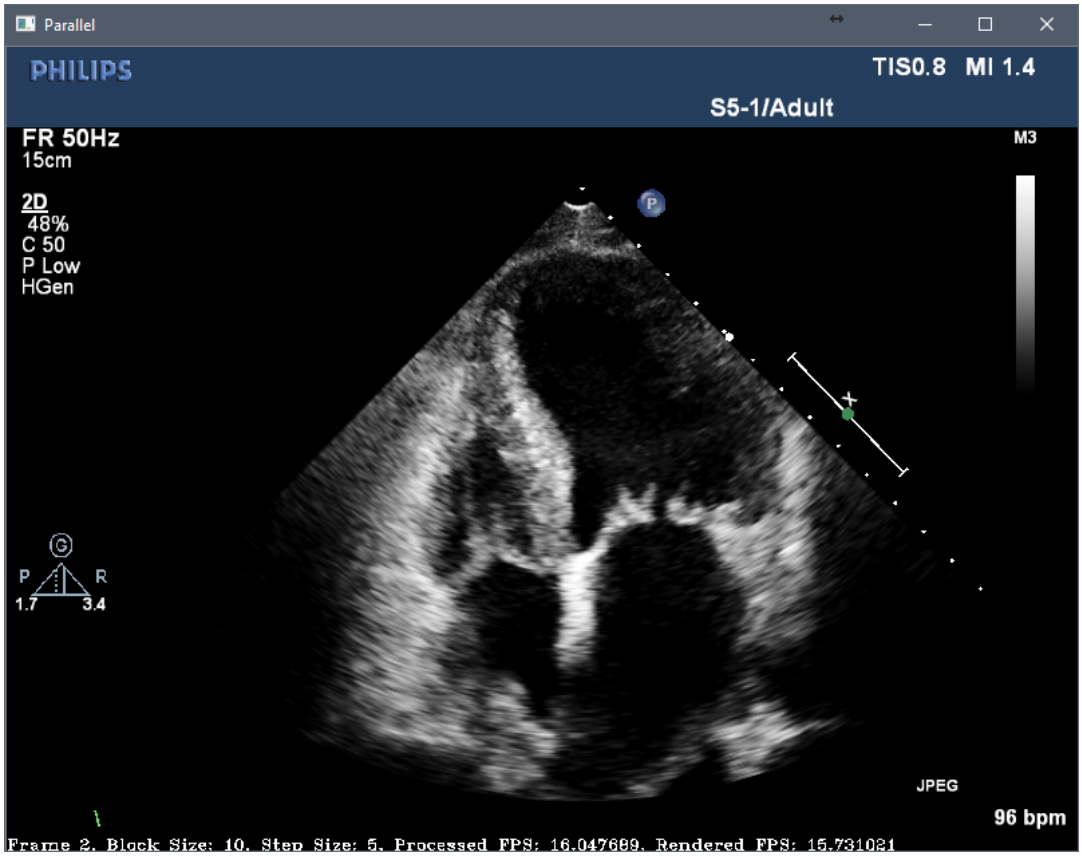


FIGURE C.6: DICOM-5

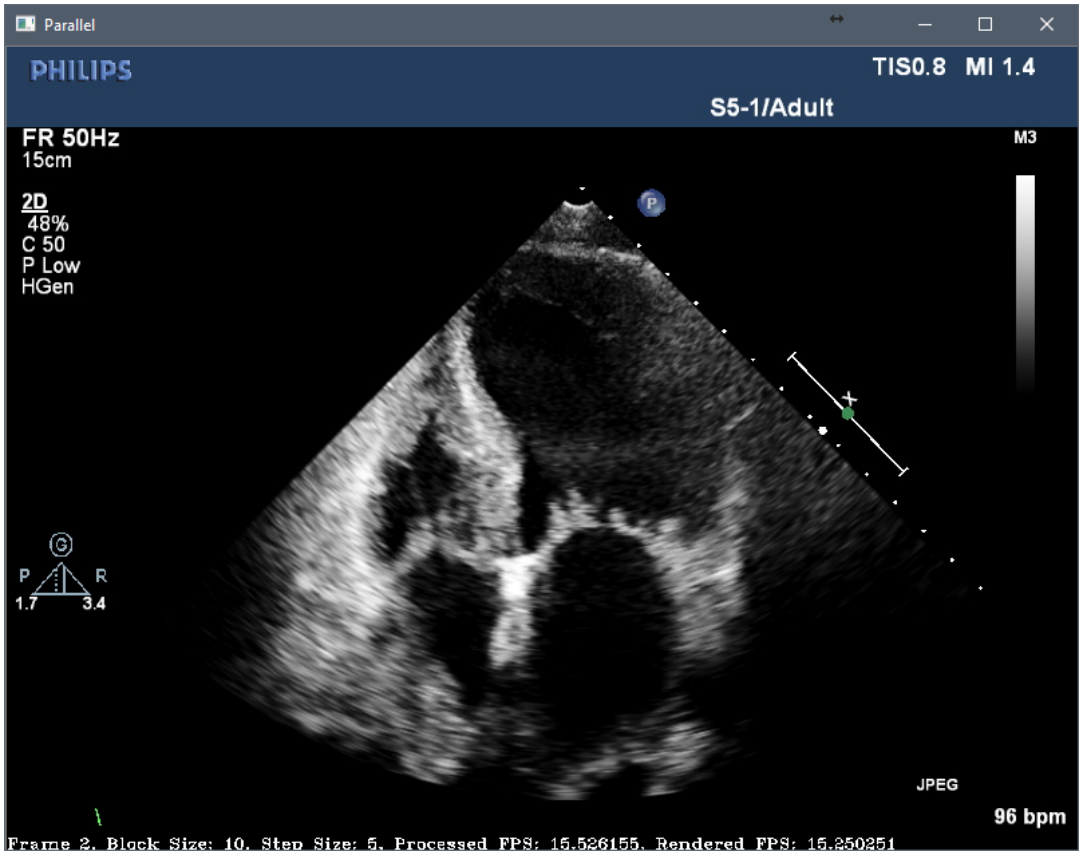


FIGURE C.7: DICOM-6

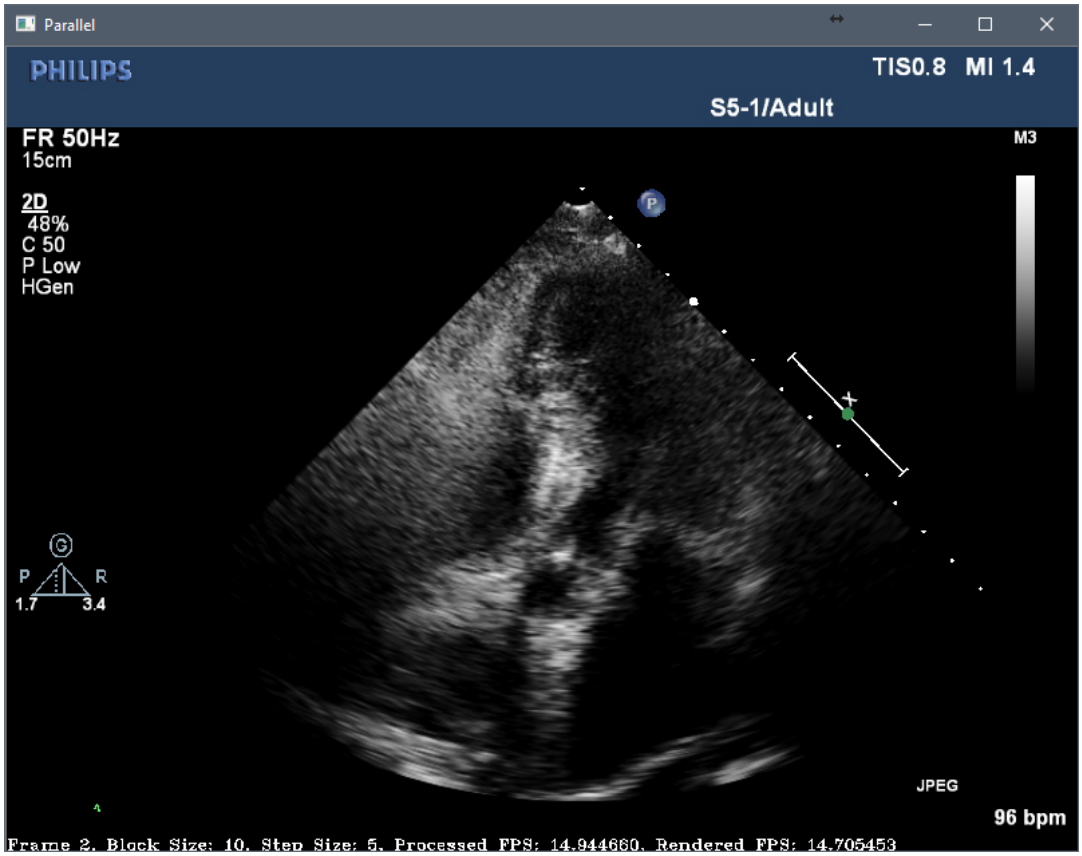


FIGURE C.8: DICOM-7

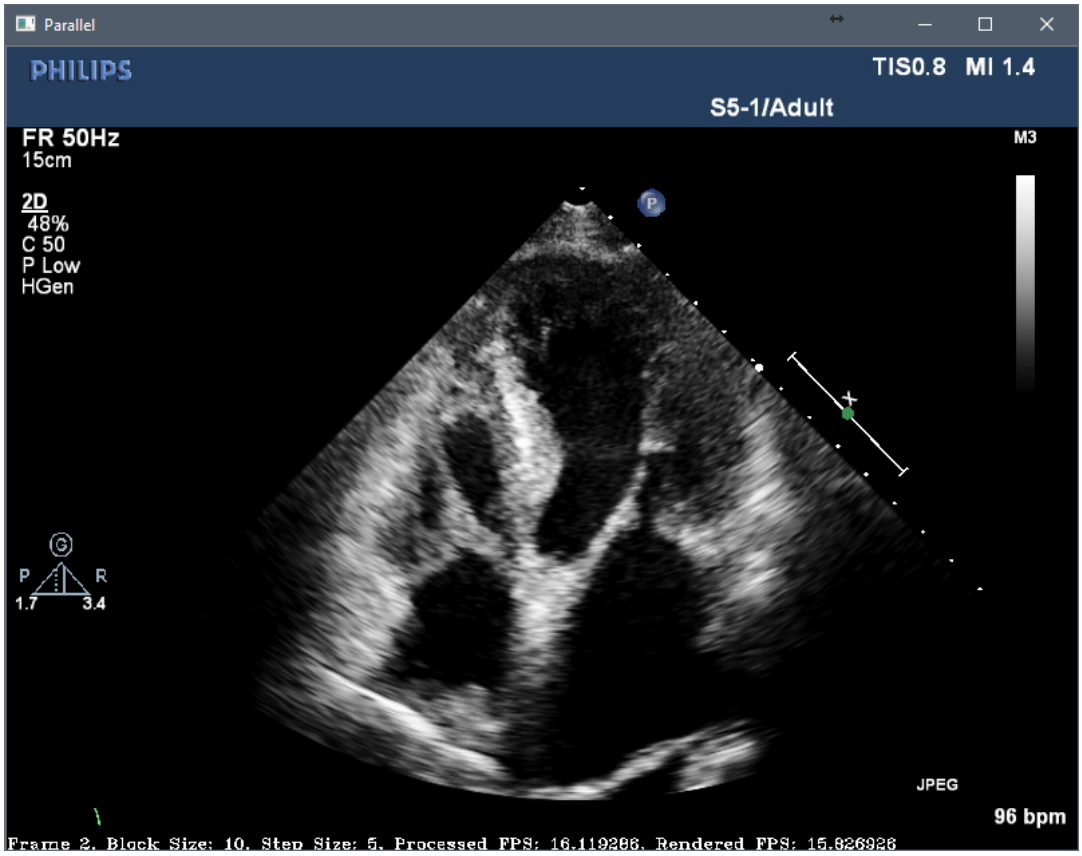


FIGURE C.9: DICOM-8

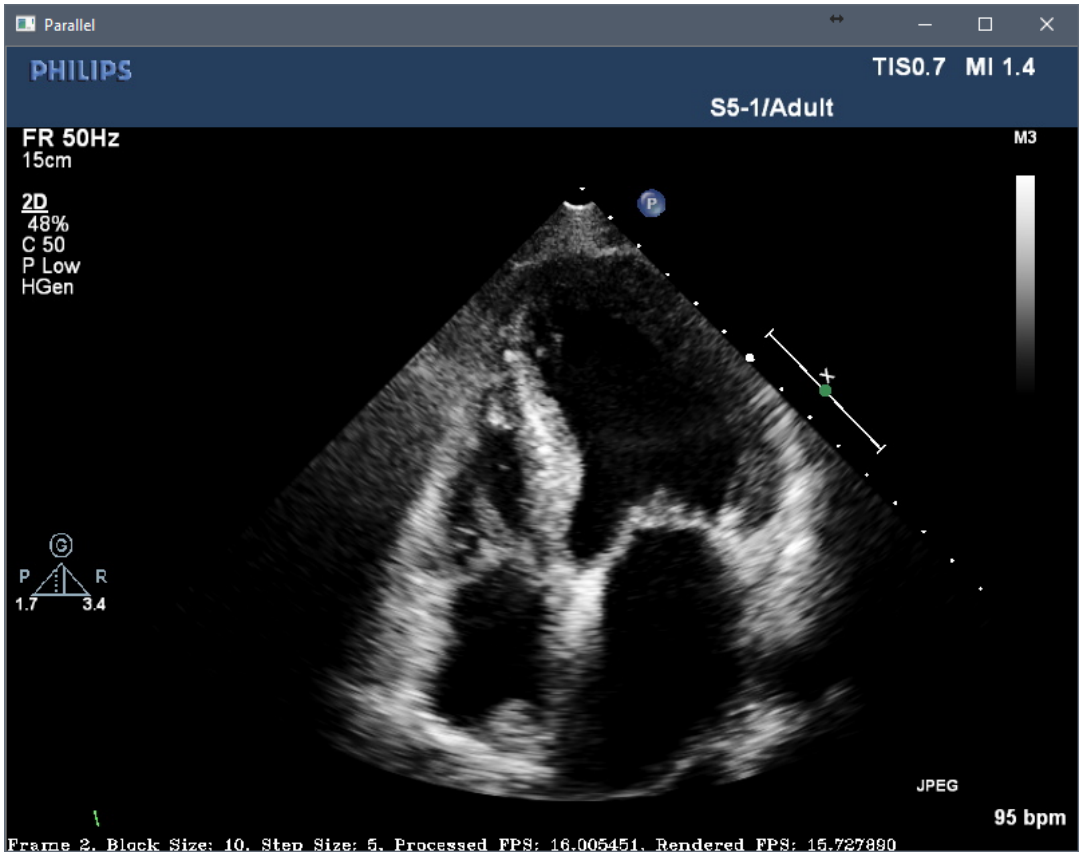


FIGURE C.10: DICOM-9

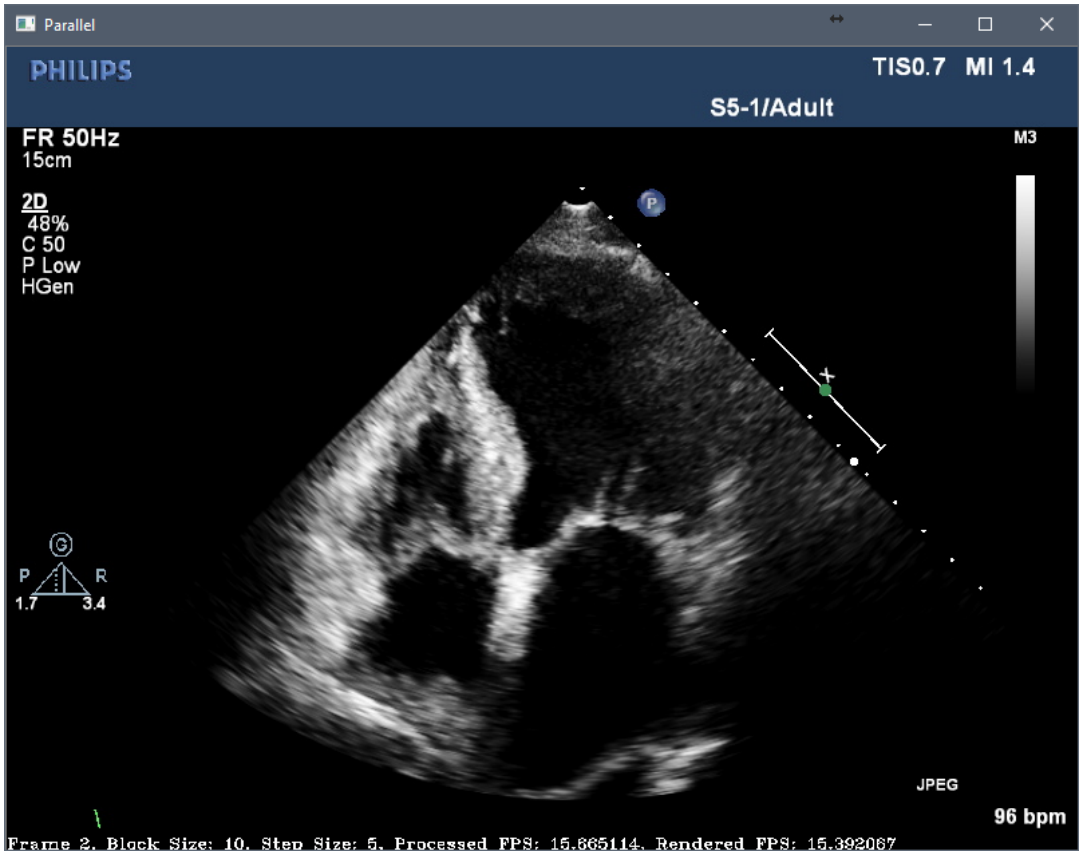


FIGURE C.11: DICOM-10

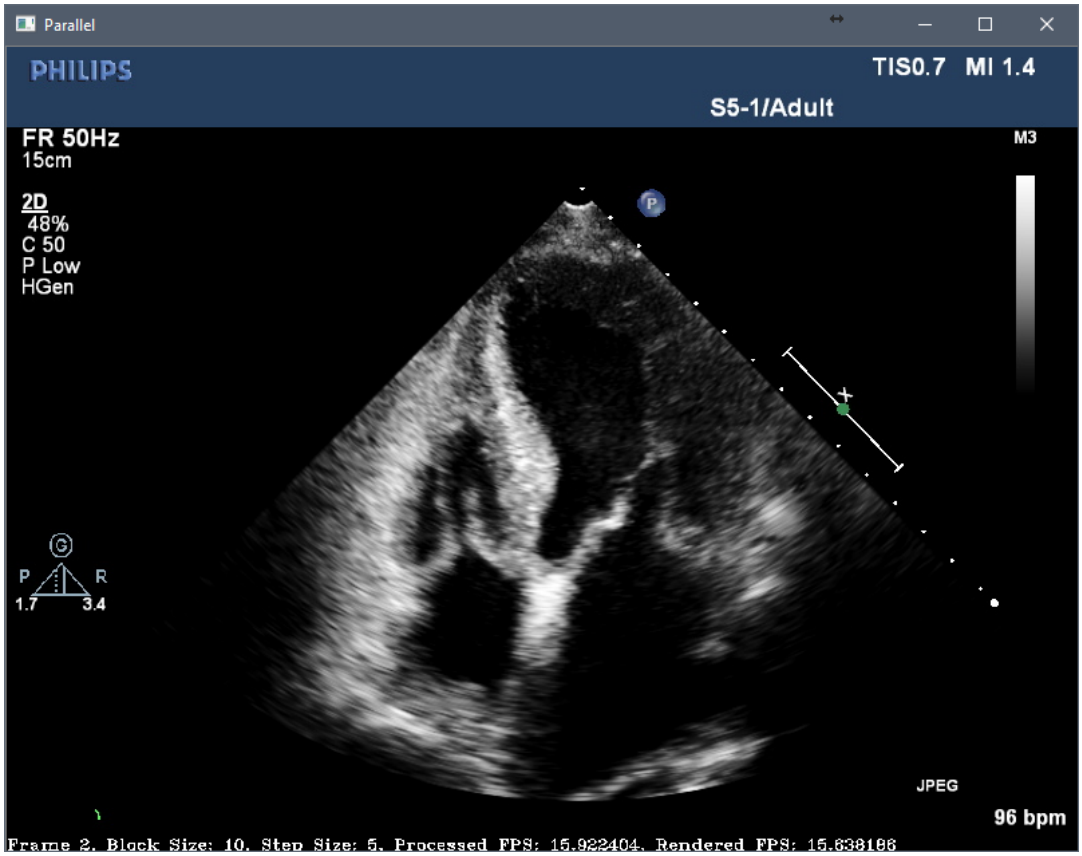


FIGURE C.12: DICOM-11

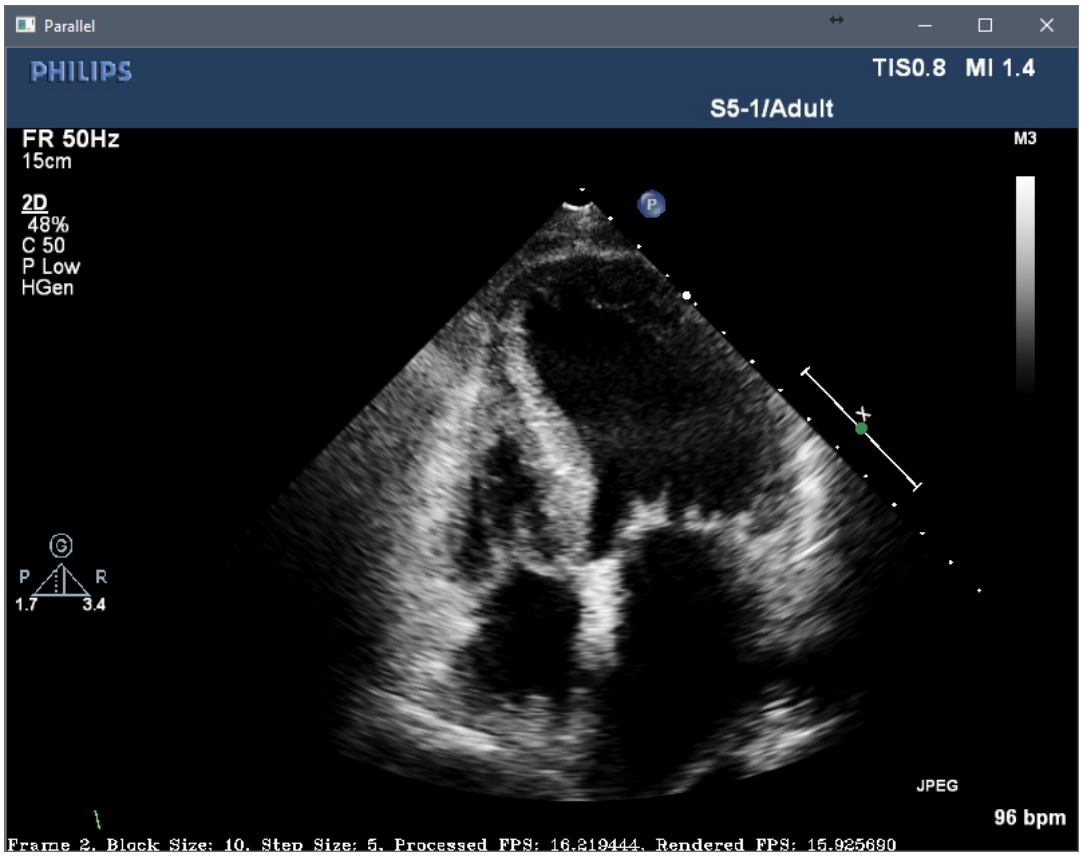


FIGURE C.13: DICOM-12

```
__kernel void ExperimentalSAD(
    __read_only image2d_t prev, __read_only image2d_t curr,
    const uint step_size, const uint blockSize, uint width, uint height,
    __global int2 * motionVectors, __global float2 * motionDetails
){
    const int x = get_global_id(0), y = get_global_id(1);
    const int2 currPoint = { x * step_size, y * step_size };
    const int wB = get_global_size(0), sWindow = blockSize;;

    int idx = x + y * wB, current_sum = matrix_sum(curr, currPoint,
    ↪ blockSize, 255);
    motionVectors[idx] = currPoint;
    motionDetails[idx] = (float2)(0, 0);

    int3 closest = closest_inbound_neighbour(prev, currPoint, sWindow,
    ↪ width, height, blockSize);
    float distanceToBlock = FLT_MAX;
    int bestErr = closest.z, err, ref_sum = 0, last = blockSize - 1;

    for (int row = closest.x; row < sWindow; row++) {
        int real_row = currPoint.x + row, real_col = currPoint.y +
        ↪ closest.y;

        if (row == closest.x) {
            ref_sum = closest.z;
        }
        else {
            int row_left = col_sum(prev, (int2)(real_row - 1, real_col),
            ↪ blockSize);
            int row_last = col_sum(prev, (int2)(real_row + last,
            ↪ real_col), blockSize);
            ref_sum = (ref_sum - abs(row_left) + abs(row_last));
        }

        for (int col = closest.y; col < sWindow; col++) {
            real_col = currPoint.y + col;

            if (col != closest.y) {
                int col_left = row_sum(prev, (int2)(real_row, real_col -
                ↪ 1), blockSize);
                int col_last = row_sum(prev, (int2)(real_row, real_col +
                ↪ last), blockSize);
                ref_sum = (ref_sum - abs(col_left) + abs(col_last));
            }
            err = abs(current_sum - ref_sum);
            int2 refPoint = { real_row, real_col };

            float newDistance = euclidean_distance(refPoint.x,
            ↪ currPoint.x, refPoint.y, currPoint.y);
            if (err < bestErr || (err == bestErr && newDistance <=
            ↪ distanceToBlock)) {
                bestErr = err;
                distanceToBlock = newDistance;
                float p0x = currPoint.x, p0y = currPoint.y -
                ↪ sqrt((float)(square(refPoint.x - p0x) +
                ↪ square(refPoint.y - currPoint.y)));
                float angle = (2 * atan2(refPoint.y - p0y, refPoint.x -
                ↪ p0x)) * 180 / M_PI;
                motionVectors[idx] = refPoint;
                motionDetails[idx] = (float2)(angle, distanceToBlock);
            }
        }
    }
}
```