Hardware Acceleration using FPGAs for Adaptive Radiotherapy

EngD Thesis

Fraser D. Robinson

Department of Biomedical Engineering

University of Strathclyde, Glasgow

March 10, 2021

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom Copyright Acts as qualified by University of Strathclyde Regulation 3.50. Due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Abstract

Adaptive radiotherapy (ART) seeks to improve the accuracy of radiotherapy by adapting the treatment based on up-to-date images of the patient's anatomy captured at the time of treatment delivery. The amount of image data, combined with the clinical time requirements for ART, necessitates automatic image analysis to adapt the treatment plan. Currently, the computational effort of the image processing and plan adaptation means they cannot be completed in a clinically acceptable timeframe. This thesis aims to investigate the use of hardware acceleration on Field Programmable Gate Arrays (FPGAs) to accelerate algorithms for segmenting bony anatomy in Computed Tomography (CT) scans, to reduce the plan adaptation time for ART.

An assessment was made of the overhead incurred by transferring image data to an FPGA-based hardware accelerator using the industry-standard DICOM protocol over an Ethernet connection. The rate was found to be likely to limit the performance of hardware accelerators for ART, highlighting the need for an alternative method of integrating hardware accelerators with existing radiotherapy equipment.

A clinically-validated segmentation algorithm was adapted for implementation in hardware. This was shown to process three-dimensional CT images up to 13.81 times faster than the original software implementation. The segmentations produced by the two implementations showed strong agreement.

Modifications to the hardware implementation were proposed for segmenting four-dimensional CT scans. This was shown to process image volumes 14.96 times faster than the original software implementation, and the segmentations produced by the two implementations showed strong agreement in most cases.

A second, novel, method for segmenting four-dimensional CT data was also proposed. The hardware implementation executed 1.95 times faster than the software implementation. However, the algorithm was found to be unsuitable for the global segmentation task examined here, although it may be suitable as a refining segmentation in the context of a larger ART algorithm.

Contents

List of Figures

List of Tables

List of Acronyms

ACP Accelerator Coherency Port

AE Application Entity

AFI AXI FIFO Interface

ART Adaptive Radiotherapy

ASIC Application Specific Integrated Circuit

AXI Advanced Extensible Interface

BRAM Block Random Access Memory

CBCT Cone Beam Computed Tomography

CLB Configurable Logic Block

CT Computed Tomography

CTV Clinical Tumour Volume

DICOM Digital Communication in Medicine

DIMSE DICOM Message Service Element

DIMSE-C Composite DIMSE

DMA Direct Memory Access

DMAC Direct Memory Access Controller

DSC Dice Similarity Coefficient

DSP Digital Signal Processing Block

EBRT External Beam Radiotherapy

ECC Edinburgh Cancer Centre

FF Flip-flop

FIFO First-in, First-out

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

GTV Gross Tumour Volume

HBM High Bandwidth Memory

HLS High Level Synthesis

HU Hounsfield Unit

ICRU International Commission on Radiation Units

IGRT Image Guided Radiotherapy

IOB Input/Output Block

IP Intellectual Property

LAN Local Area Network

LUT Look-up Table

MB Megabyte

MRI Magnetic Resonance Imaging

MV Megavolts

NRE Non-recurring Engineering

NTCP Normal Tissue Complication Probability

OAR Organs at Risk

PACS Picture Archive and Communication System

PCIe Peripheral Component Interface express

PET Positron Emission Tomography

PTV Planning Tumour Volume

QA Quality Assurance

RAM Random Access Memory

ROI Region of Interest

ROM Read Only Memory

SBRT Stereotactic Body Radiotherapy

SCP Service Class Provider

SCU Service Class User

SIMD Single Instruction Multiple Data

SoC System on Chip

TCP Tumour Control Probability

TCP/IP Transmission Control Protocol/Internet Protocol

UID Unique Identifier

VLIW Very Long Instruction Word

Acknowledgements

There are many people I would like to acknowledge for their help in the preparation of this thesis. First and foremost, I would like to thank my family, and particularly Kirsty. Without their support, encouragement and gentle cajoling this thesis would never have been completed.

I would also like to express my gratitude for the substantial amount of advice and guidance provided over the years by my supervisors, in particular Louise Crockett and Bill Nailon, and their considerable patience while reviewing the various drafts.

I have had the great fortune to have worked with a great number of colleagues during this period. Primarily those in the DSP Enabled Communications group, but also those I shared the "cupboard" with at the Edinburgh Cancer Centre and those in the Biomedical Engineering department. Their assistance was not only invaluable to the production of this thesis, but their camaraderie and fellowship made it positively enjoyable at times.

Lastly, I would like to acknowledge the staff of the Edinburgh Cancer Centre for sharing their time, expertise and resources.

Acknowledgements

Chapter 1

Introduction

1.1 Introduction

Technological advances in external beam radiotherapy in recent decades have seen an increased availability of three- and four-dimensional imaging modalities to aid diagnosis and treatment planning, and of high-quality imaging modalities at the point of treatment delivery. These advances have been coupled with improvements in the precision with which radiotherapy can be delivered, leading to more accurate treatments. Given that the goal of radiotherapy is to maximise the radiation dose to the clinical target, whilst minimising the dose to other tissues, improvements in the accuracy of treatment has resulted in better outcomes for patients, and has enabled the consideration of alternative dose regimes.

Adaptive radiotherapy (ART) is an emerging technique with the aim of further improving the accuracy of radiotherapy by adapting the treatment plan based on images of the patient's anatomy captured at the time of treatment. ART allows for greater compensation for changes in the size, position and shape of the tumour and surrounding anatomy during the course of treatment than conventional radiotherapy [1]. This is of particular value to tumours in the abdomen, where there can be considerable deformation and movement of soft tissues caused by normal physiological processes.

Plan adaptation depends on establishing a correspondence between anatomical features in the images used to create the treatment plan and the images captured at the time of treatment. Identification of bony anatomy in the two sets of images is a common initial step in establishing this correspondence, since bony anatomy tends to have fewer degrees of freedom for changing size, shape and position than soft tissues.

The amount of image data produced by the modern radiotherapy workflow, combined with the clinical time requirements for ART, necessitates automatic image analysis to establish the correspondence between the treatment planning and treatment delivery image sets and, hence, to adapt the treatment plan. Currently, the computational effort of the image processing and plan adaptation operations means they cannot be completed in a clinically acceptable timeframe, thereby preventing the use of ART in routine clinical practice.

Hardware acceleration has been shown to be effective at reducing the execution time of some algorithms compared to software executing in a general purpose processor. This is achieved by the use of customised processors, specifically designed to suit the computational requirements of the algorithm.

The aim of this thesis is to investigate the use of hardware acceleration to accelerate algorithms for the segmentation of bony anatomy in Computed Tomography (CT) scans, in order to reduce the plan adaptation time for ART.

1.2 Clinical Context

Cancer is a disease that arises from the uncontrolled growth and proliferation of cells. These malignant cells compete with healthy cells for resources, and their spread can disrupt the structure and function of healthy tissues. If allowed to continue, this can eventually result in the death of the organism.

A common treatment for many cancer types is radiotherapy, where the cancerous cells are exposed to ionising radiation. The ionising radiation causes damage to the internal structures of the cell, with the aim of destroying the cell's ability to reproduce.

Healthy cells can also be damaged by exposure to ionising radiation, and it is, therefore, imperative to the efficacy of radiotherapy that the ionising radiation is delivered accurately.

Conventional external beam radiotherapy can be delivered in a number of fractions over a period of time up to several weeks. The fractions are incorporated into a treatment plan, which defines the characteristics and geometry of the radiation beam used to deliver the treatment. The treatment plan is created some time prior to treatment delivery using images of the patient's anatomy that represent a snapshot in time.

In conventional radiotherapy, uncertainties in the position of the tumour at treatment time, relative to its position in the treatment planning image data, are accounted for by increasing the size of the target volume to encompass any anticipated movement of the tumour and thereby ensure it receives the prescribed dose of radiation. Such an approach is undesirable, as increasing the target volume increases the radiation dose received by healthy tissues, and escalates the risk of complications arising from the treatment. Increasing the accuracy of radiotherapy delivery reduces the size of the margins added to the target volume to account for positional uncertainty, and therefore, reduces the radiation dose to healthy tissues. This can enable either a decrease in the risk of complications arising from the treatment, or an escalation of the dose delivered to the tumour.

1.3 Adaptive Radiotherapy

ART seeks to improve the accuracy of radiotherapy delivery by adapting the original treatment plan based on images of the patient's anatomy captured immediately prior to, or during, treatment delivery. The aim of this approach is to ensure the intended dose of radiation is delivered to the clinical target at each fraction, whilst also avoiding irradiating healthy tissue. Unlike conventional radiotherapy, by adapting the treatment, ART can take account of interfractional changes in the size, shape and position of the tumour and surrounding anatomy, and intrafractional motion, where the tumour or surrounding anatomy move during the treatment fraction.

Cancers located in the abdomen are of particular interest for ART due to the movements and changes in shape that can be induced in the tumour and surrounding tissue by the respiratory cycle and variable filling of the organs of digestion and excretion.

Four-dimensional CT (4DCT) is a relatively recently developed imaging modality that improves the visualisation and characterisation of tumour motion, particularly that resulting from the respiratory cycle. By providing a patient-specific characterisation of tumour motion, the use of 4DCT imaging for treatment planning can reduce the margins added to the treatment target volume compared to population-based margins, which are used conventionally [2].

However, 4DCT produces much more image data than conventional CT, further adding to the computational workload of ART, making it more difficult to meet the clinical timing requirements. Furthermore, imaging artifacts can arise in 4DCT image data as a result of the interplay between the sampling frequency of the imaging system and the period of the tumour or organ motion [1, 2]. These artifacts can add to the challenge of segmenting the anatomical features in the 4DCT images.

The time taken to adapt the treatment plan is critical to the aim of ART. Confidence in the image data captured at the time of treatment delivery accurately representing the state of the patient's anatomy diminishes over time. Therfore, the plan adaptation time must be minimised. Maximum plan adaptation times of between 5 and 10 minutes have been suggested to make the technique feasible clinically [3, 4]. Although ART is an area of active research, there are few established ART techniques that are able to meet the timing requirements to make ART routine clinical practice.

ART depends on computationally intensive image processing operations to establish a correspondence between the image data used to generate the treatment plan, and the image data captured at the time of treatment delivery, from which the plan can be adapted. To reduce the time taken to adapt the treatment, and allow ART to meet the clinical timing requirements, these image processing operations need to be completed as fast as possible.

1.4 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are electronic devices containing an array of configurable logic elements and interconnects that can be programmed to implement custom circuits. These enable hardware architectures to be created specifically to implement particular algorithms. Such custom architectures have been shown to compute some algorithms, which are parallel in nature, faster than the general purpose processors, such as Central Processing Units (CPUs), commonly found in desktop computers.

The last few years has also seen the emergence of FPGA-based Systems-on-Chip (SoC), which combine FPGA fabric interconnected with hard-wired general purpose processors. These devices allow algorithms to be partitioned into different stages that can be executed on the processing architecture to which they are best-suited. So stages that are inherently sequential in nature, or require decision making, can be targeted for implementation on a processing architecture better suited to that, such as a CPU. Additionally, the combination of more conventional processing architectures tightly coupled with FPGA fabric can greatly simplify the integration of FPGA-based hardware accelerators into existing systems.

1.5 Aim and Contributions

The aim of this thesis is to accelerate image segmentation algorithms suitable for identifying bony anatomy in CT and 4DCT images of patients receiving radiotherapy for abdominal cancers. Two sets of image data are considered in this thesis, both obtained from the Edinburgh Cancer Centre (ECC), where much of this work was carried out. Three-dimensional CT image data of bladder cancer patients receiving radiotherapy is examined in Chapter 5. Modifications to the bony anatomy segmentation algorithm proposed by Haas *et al.* [5] are suggested to improve its performance and simplify its implementation in hardware. 4DCT image data of a quality assurance testing phantom

is investigated in Chapter 6. The algorithm discussed in Chapter 5 is again applied, and further modifications are suggested to improve its performance on 4DCT data. A novel algorithm based on Otsu's method [6] is also presented in Chapter 6.

The work presented in this thesis seeks to contribute to addressing the relative lack of investigation of FPGAs for hardware acceleration in ART. Specifically, it seeks to consider the application of FPGAs to image analysis problems typical in ART to assess the acceleration that may be achieved compared to execution on a CPU. This thesis uses FPGA-based SoCs to implement bony anatomy segmentation algorithms with the aim of reducing their execution time and thereby reducing the time taken to adapt a treatment plan for ART.

The data transfer overhead of moving image data between existing radiotherapy equipment and an ART hardware accelerator must be factored in to any estimation of the potential performance improvement offered by hardware acceleration. To this end, the integration of FPGA-based SoCs with existing radiotherapy equipment using the industry-standard DICOM protocol is analysed in Chapter 4.

In addressing these aims the following contributions resulted:

- a comprehensive characterisation and comparison of the DICOM protocol transfer rate on FPGA-based SoC platforms and personal computers for transferring medical image data;
- proposed novel modifications to the bony anatomy segmentation algorithm of Haas *et al.* [5], that substantially reduce the execution time of the algorithm, whilst maintaining a comparable quality of segmentation;
- developed a hardware implementation of the above novel segmentation algorithm, termed the *Simplified Haas* algorithm, which exhibits a significant improvement in execution time performance compared to the algorithm executing in software under certain conditions;
- proposed further modifications to the Simplfied Haas algorithm that significantly improve the quality of the segmentations produced when applied to 4DCT image data, and termed this algorithm the *Noise-reduced* algorithm;

- developed a hardware implementation of the Noise-reduced algorithm and demonstrated its execution time performance to be superior to that of the same algorithm executing in software;

- developed a hardware implementation of a novel image segmentation algorithm, termed the *4DCT Optimal Threshold* algorithm, to segment 4DCT images into three classes, based on Otsu's method [6] and a three-dimensional mean filter; demonstrated the execution time performance to be superior to the same algorithm executing in software and fully discussed the limitations of the algorithm for its intended application.

1.6 Outputs

The following outputs have been produced as a result of this work:

- Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications. Crockett L, Ramsay C, Northcote D, Robinson F, Stewart R,
  Strathclyde Academic Media (2019);

- Hardware acceleration of automated 4DCT analysis. Robinson FD, Crockett LH, Nailon WH, Stewart RW, McLaren DB,
  Presented at the 6th Annual Scientific Meeting of the Scottish Radiotherapy Research Forum (ScoRRF 2017);

- High-level synthesis for medical image processing on Systems on Chip: A case study. Robinson FD, Crockett LH, Nailon WH, Stewart RW,
  2016 26th International Conference on Field Programmable Logic and Applications (FPL), 1-2 (2016) doi: 10.1109/FPL.2016.7577390;

- Hardware accelerated segmentation of CT images for adaptive radiotherapy. Robinson FD, Crockett LH, Nailon WH, Stewart RW, McLaren DB,
  Presented at the SINAPSE Annual Scientific Meeting (2016);

- Hardware accelerated image processing to enable real-time adaptive radiotherapy.
  Robinson FD, Crockett LH, Nailon WH, Stewart RW, McLaren DB,
  Presented at The Royal Society of Medicine & IET conference on the future of medicine — the role of doctors in 2025 (2016).

1.7 Thesis Outline

The remainder of this thesis is arranged as follows.

Chapter 2 extends the brief introduction to radiotherapy and ART provided here, and sets the work presented in this thesis in the context of the clinical problem.

A more detailed description of hardware acceleration, FPGAs and FPGA-based SoCs is given in Chapter 3, along with an overview of some of the tools used in this thesis to develop custom hardware.

The DICOM standard and protocol are introduced in Chapter 4. The data transfer performance of two off-the-shelf, FPGA-based SoC platforms using the DICOM protocol is also characterised and compared with the performance of a desktop computer. Furthermore, a comprehensive analysis of the factors affecting the recorded performance is provided.

Chapter 5 examines the bony anatomy segmentation algorithm proposed by Haas *et al.* [5] in the context of three-dimensional CT images of bladder cancer patients. A number of modifications are suggested to this algorithm to improve its execution time performance and to implement the algorithm in hardware. An original hardware implementation of this algorithm is developed on an FPGA-based SoC, and a comparison is made, in terms of execution time and segmentation quality, with the original algorithm.

Chapter 6 considers segmenting high density target objects in 4DCT images of a quality assurance phantom. A novel algorithm to segment the image data into three classes, based on Otsu's method [6] and a three-dimensional mean filter, is proposed and developed in hardware. Its performance, in terms of execution time and segmentation quality, are discussed, as are its limitations. Modifications to the algorithm proposed

in Chapter 5 to improve the quality of the segmentation produced using 4DCT data are put forward. A hardware implementation of this algorithm is developed and its performance, in terms of execution time and segmentation quality, is reviewed.

Concluding remarks and a review of the aims of this thesis are provided in Chapter 7, along with suggestions for future work.

Chapter 2

Adaptive Radiotherapy in Pelvic Cancers

This chapter outlines the need for accelerated image processing in ART in pelvic cancers. The use of conventional radiotherapy as a treatment for cancer is discussed, including the importance of accuracy in treatment delivery. ART is introduced and the current limitations hindering its clinical implementation are presented. The potential for ART in pelvic cancers, such as bladder and prostate cancers, is also discussed.

2.1 Cancer

Cancer is a disease caused by poorly controlled cell growth and proliferation, leading to an accumulation of cells that form a malignant neoplasm or tumour. Usually, the cells composing a tumour are the daughter cells of a single malignant cell, which arose from an actively proliferating cell type [7, 8].

Cancer cells are metabolically very active and compete with healthy cells for space, oxygen and nutrients. At the same time, cancer cells tend not to perform the normal functions of the progenitor cell type efficiently.

The competition with healthy cells for resources can also precipitate changes in the extracellular environment within the tumour, such as areas of depleted oxygen concentration or *hypoxia*. This in turn can cause cancer cells to express factors that

encourage *angiogenesis*, or the formation of new blood vessels, to serve the tumour with oxygen and nutrients, further promoting growth of the tumour [8]. As the size of the tumour increases, it can invade the surrounding healthy tissue, competing with the normal cells and disrupting the function of the tissue.

If the disease is able to progress, the death of the organism can follow as the function of vital organs is impaired by the non-functional cancer cells supplanting healthy cells, or through the loss of healthy cells, due to competition for oxygen and nutrients from cancer cells [7, 8].

The formation of *metastases*, or secondary tumours in remote organs of the organism, represents a significant stage in the progression of the disease. It is often used as a factor to assess disease progression clinically. Once the tumour has spread to distant organs within the body it is significantly more challenging to control therapeutically [7,8]. Indeed, the majority of human deaths from cancer are caused by metastatic disease [8].

The most recent global survey estimated that there were 18.1 million new cases of cancer diagnosed in 2018 and 9.6 million deaths from cancer [9, 10]. In the UK, there are around 367 000 new cases of cancer diagnosed each year and around 165 000 cancer deaths [11].

2.2 Radiotherapy

Radiotherapy is used to control a cancerous tumour or tumours through the application of ionising radiation. In this context, the term control means to remove the ability of any of the cancerous cells to reproduce and proliferate.

Tumour Control Probability

Radiotherapy uses ionising radiation to inflict irreparable damage on the DNA of cancer cells. The dose of radiation given to the patient is measured in units of Gray (Gy) where 1Gy is equivalent to 1J of energy deposited per kilogram of tissue. The likelihood of achieving tumour control is termed the Tumour Control Probability (TCP).

Normal Tissue Complication Probability

Clearly the application of ionising radiation to the patient has the potential to damage healthy cells as well as cancerous ones. The likelihood of inducing toxicity in healthy tissues is termed the Normal Tissue Complication Probability (NTCP) and limits the radiation dose that can be delivered to the cancerous cells.

A low NTCP can be tolerated in some normal tissues, depending on the severity of the complication, in order to increase TCP [12]. NTCP values are empirically derived and can be influenced by factors such as the sensitivity of the cells to irradiation, the volume of an organ that is irradiated and the functional structure of the organ [12].

Therapeutic Ratio

The therapeutic ratio is the ratio between TCP and NTCP. The ratio depends on the radiation dose delivered, the type of cells in the tumour, the location of the tumour relative to the surrounding healthy tissue and the type of healthy tissue [13]. Figure 2.1(a) shows an example of a high therapeutic ratio where a high TCP can be achieved while keeping NTCP low. Figure 2.1(b) shows an example of a low therapeutic ratio where a reasonably high TCP is only possible at the expense of a high NTCP.

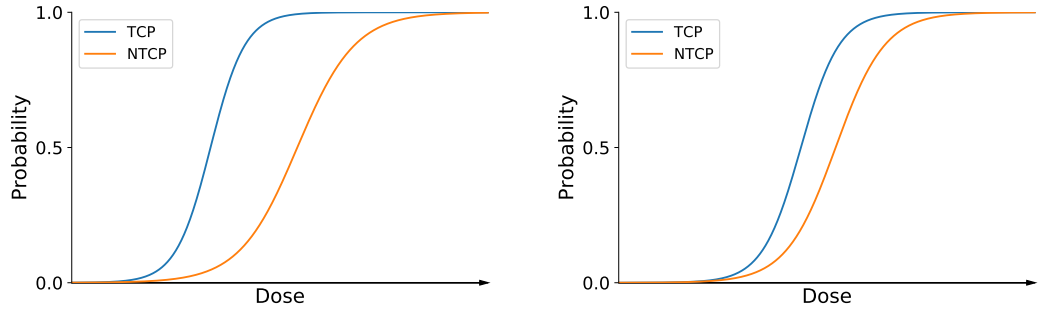


Figure 2.1: Example TCP and NTCP curves for (a) high therapeutic ratio and (b) low therapeutic ratio

Need for Accuracy in Radiotherapy

The idealised radiotherapy would deliver an infinitely high dose of ionising radiation to the tumour whilst simultaneously avoiding any irradiation of normal tissue. This can be seen from the graphs in Figure 2.1, where the TCP approaches unity with increasing dose to the tumour, while the NTCP tends to zero with decreasing radiation dose to normal tissue.

The exposure of normal tissue to ionising radiation is unavoidable with current radiotherapy techniques. However, improving the accuracy of radiotherapy helps to minimise the exposure of normal tissues to ionising radiation whilst maintaining the dose to the tumour, thereby increasing the therapeutic ratio of the treatment.

2.2.1 External Beam Radiotherapy

There are two broad approaches to radiotherapy depending on the position of the radiation source relative to the patient: External Beam Radiotherapy (EBRT) and brachytherapy [14]. In brachytherapy, the radiation source is implanted into the patient as close to the tumour as possible to expose the lesion to the highest intensity of radiation. Healthy tissue surrounding the tumour is exposed to the radiation that is not absorbed by the tumour.

In EBRT, the radiation source remains external to the patient and the radiation is directed toward the tumour. Healthy tissue is exposed to the radiation passing through the patient en route to the lesion and to the radiation that is not absorbed by the tumour. The work presented here has focussed on EBRT.

Linear Accelerators

Different forms of ionising radiation can be used in EBRT with differing radiation properties and ease of use. Most EBRT currently delivered in the UK uses linear accelerators to produce x-ray photons. Figure 2.2 shows a simple linear accelerator.

Linear accelerators allow some of the properties of the x-ray beam produced to be configured. They also enable the output beam to be collimated and shaped and to modulate its intensity to better suit the clinical target [15].

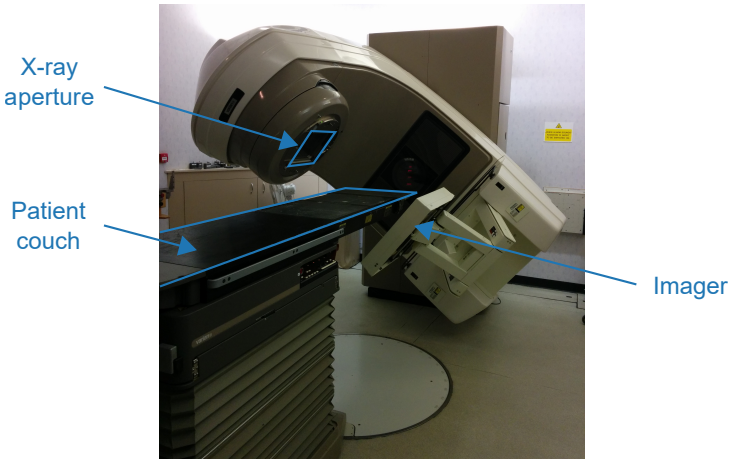


Figure 2.2: Simple linear accelerator

The pattern of energy absorption by the patient's tissue depends on the energy of the photons in the x-ray beam [14]. Figure 2.3 shows energy deposition in water, which is approximately analogous to soft tissue, with beam energy. Treatments are typically delivered at a single beam energy.

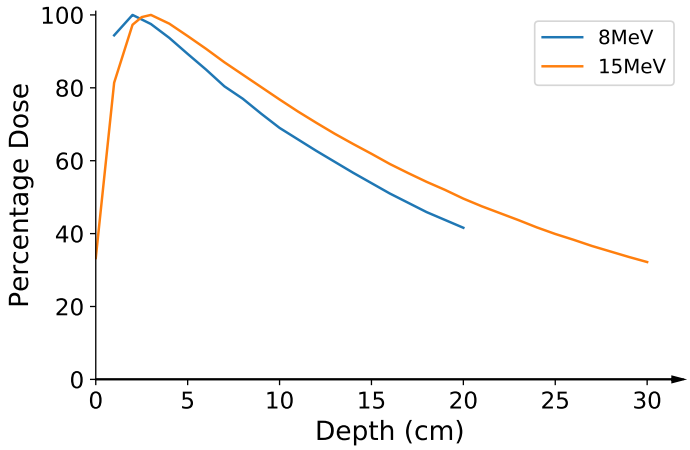


Figure 2.3: Dose, normalised to the maximum, against depth in water for x-ray beams with 8MeV and 15MeV energies [16, 17]

It can be seen from Figure 2.3 that a substantial proportion of the dose is deposited over a large range of depth in the patient's tissues. It is clear from this that precisely delivering the radiation dose to the tumour without also depositing significant doses

in healthy tissue in the path of the beam is not possible. However, the effect can be mitigated by delivering the prescribed dose of radiation to the tumour using multiple beams with different trajectories to vary the normal tissue that is in the path of the beam.

Protons and Heavy Ions

There is growing interest in the use of protons and heavy ion forms of ionising radiation in radiotherapy. These have the advantage of depositing their energy in a much more precise manner compared to photons, reducing the amount of dose received by healthy tissue on the entry and exit paths of the beam [18].

The precision with which these forms of radiation deposit dose also means that geometric inaccuracies in targeting the cancerous lesion can result in much larger differences in the dose distribution to the lesion and healthy tissues than would be the case for photons. These forms of radiation therefore require a high level of geometric accuracy in their use for radiotherapy.

2.2.2 Fractionation

In order to improve the therapeutic ratio, radiotherapy is often delivered in fractions of 1.8-3Gy around 24 hours apart over the course of several weeks [19–21]. Each fraction causes both reparable and irreparable damage to the cells in the cancerous lesion and the healthy tissue exposed to the radiation.

By delivering the treatment in fractions, the healthy tissue cells have the opportunity to repair damage between fractions. The tumour cells also have the opportunity to repair damage, however, they are typically less effective at doing so than normal cells and so the normal tissue benefits more than the cancerous tissue [13].

The radiosensitivity of tumour cells varies depending on which phase of the cell cycle they are in and how well oxygenated they are. Tumour cells with a low radiosensitivity at one fraction may have a higher radiosensitivity at subsequent fractions because they are in a different phase of the cell cycle or their oxygen supply has improved [13].

The size of each fraction and hence the total length of the course of radiotherapy must be balanced by the risk of repopulation of tumour cells exceeding the rate at which they are killed and the treatment failing to achieve tumour control [13].

Hypofractionation

Cancers of some tissues have been found to benefit from hypofractionated radiotherapy [21]. A hypofractionated treatment delivers radiotherapy in fewer fractions than conventional radiotherapy but delivers a higher dose per fraction.

Stereotactic Body Radiotherapy (SBRT) is a form of hypofractionated treatment that has been found to produce better tumour control than conventional radiotherapy in some cancers [21]. Fractionation regimes of between 33.5Gy and 38Gy in four or five fractions have been proposed [21].

The high doses delivered per fraction in SBRT increase the potential for geometric inaccuracies to cause toxicity in normal tissues. Highly conformal and accurate treatments are therefore required.

2.2.3 Conventional Radiotherapy

Figure 2.4 shows the simplified treatment pathway used in conventional radiotherapy.

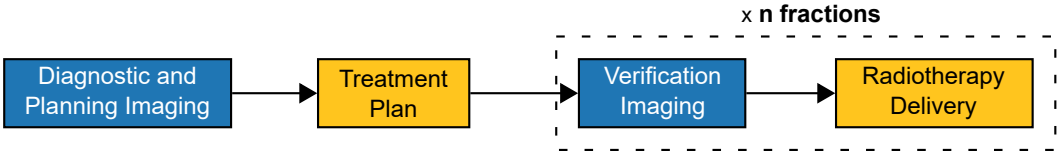


Figure 2.4: Treatment pathway for conventional radiotherapy

Before treatment can begin a treatment plan must be created. This plan defines the parameters with which to configure the treatment machine in order to deliver the prescribed dose to the target. The plan is formulated using a model of the patient's anatomy constructed from image data of the patient.

Diagnostic and Planning Imaging

Computed Tomography (CT) imaging tends to be the main imaging modality currently used in radiotherapy planning [21, 22]. CT imaging constructs a three dimensional image volume of the patient anatomy from a series of measurements of x-ray beam attenuation as the beam passes through the patient.

CT data sets tend to provide good contrast between bony and soft tissue and can also be used to estimate the electron densities of the tissues in the patient. The estimates of electron densities are essential to the accuracy of the dose calculation made using the patient model [22, 23].

Other imaging modalities can also be used to help improve the patient model for creating the treatment plan. Magnetic Resonance Imaging (MRI) is a modality that can provide much better contrast between different soft tissues than CT. This can be particularly useful for identifying and delineating different soft tissue structures [22,23].

Functional imaging modalities, such as Positron Emission Tomography (PET), where the information obtained is not anatomic but pertains to the activity within the imaged tissue, are sometimes used for diagnosing and staging the disease prior to commencing treatment. The information from these modalities can be used to aid in the determination of the extent of the disease [22, 23].

Treatment Plan

Using the pre-treatment imaging, the radiation oncologist defines the volumes to target with radiation. They also define any normal tissues in the vicinity of the planned treatment whose radiosensitivity or function warrants particular consideration to avoid irradiating. These normal tissues are termed Organs At Risk (OAR) and the radiation dose to these structures constrains the planned treatment [24, 25].

The targets to receive radiotherapy are defined in the International Commission on Radiation Units and Measurements (ICRU) Reports 50, 62 and 83 [24–26]. There are three main volumes of interest in radiotherapy planning:

- Gross Tumour Volume (GTV);

- Clinical Target Volume (CTV);
- Planning Target Volume (PTV).

The GTV identifies the extent of the tumour that is visible in images or is palpable [24–26].

The CTV includes tissues surrounding the GTV to which there is suspected to be microscopic spread of the disease [24–26].

The PTV adds a margin to the GTV and CTV to ensure that the GTV and CTV receive the prescribed dose even when there is variation in their position due to organ motion, changes in the shape, size and position of organs or deviations in set-up [24–26].

The relationship between the three volumes is illustrated in Figure 2.5.

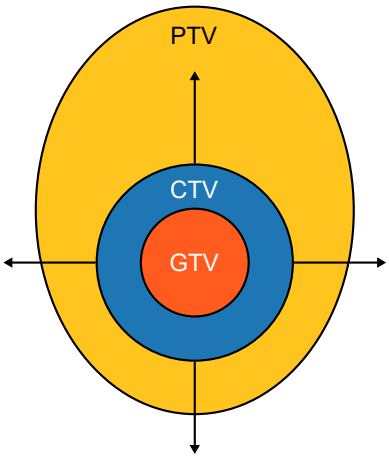


Figure 2.5: Illustration of the relationship between the GTV, CTV and PTV [24–26]

The radiation oncologist prescribes a dose of radiation and a fractionation regime from which the treatment plan is created. The treatment plan is created by selecting the linear accelerator parameters to deliver the prescribed dose to the PTV while avoiding irradiating the OARs. Reference images can be generated from the treatment plan to aid verification that the patient anatomy at the time of treatment delivery closely matches the patient model used to plan the treatment [24].

Radiotherapy Delivery

For treatment delivery, the patient is set-up to replicate their position when the treatment planning scan was made in order to accurately match the model of the patient used to create the treatment plan. As the ability of radiotherapy treatment machines to precisely deliver highly conformal dose distributions has improved, the need to deliver treatments accurately has increased.

Highly conformal dose distributions with steep dose gradients at the perimeter of the PTV allow higher doses to be planned to the PTV while maintaining the same dose to the surrounding normal tissue. However, increasing dose to the PTV increases the dose to the normal tissue within the PTV, thereby increasing the NTCP for this tissue. Therefore, it is desirable to reduce the CTV-PTV margin, which requires more accuracy in treatment delivery.

Furthermore, sudden changes in dose between the PTV and surrounding tissue increase the impact of geometric inaccuracies during treatment delivery. In an inaccurate treatment, some volumes intended to be within the PTV will receive a much lower dose than expected, reducing TCP, and some volumes not intended to be within the PTV will receive a much higher dose than expected, increasing NTCP.

Image Guided Radiotherapy

The need to increase the accuracy of radiotherapy delivery and improvements in the imaging technology available within the treatment room have led to the development of Image Guided Radiotherapy (IGRT).

IGRT covers a broad range of techniques. The general principle, however, is to use image data acquired at the time of treatment delivery to identify discrepancies in the geometry of the patient anatomy relative to the planning model and, thus, to mitigate the effects of these on the treatment.

Using IGRT allows the patient set-up to be adjusted to compensate for patient set-up errors and changes in the position of the PTV and surrounding organs. By reducing the geometric uncertainties at treatment delivery, the CTV-PTV margin can be reduced in many cases.

Cone Beam Computed Tomography

A number of imaging modalities have been proposed and are in use for IGRT, including Cone Beam CT (CBCT). CBCT is similar to conventional CT but typically images have a larger volume per acquisition plane than conventional CT.

The image quality produced by CBCT tends to be lower than that of conventional CT [22]. Some systems use the treatment x-ray beam to perform CBCT in a process called Mega-Voltage (MV) CBCT. However, better image contrast is obtained using a dedicated imaging x-ray beam in a process called Kilo-Voltage (kV) CBCT [22]. Figure 2.6 shows a linear accelerator with a kV CBCT system. kV CBCT is the system predominantly used for IGRT at ECC and will simply be referred to as CBCT in the remainder of this thesis.

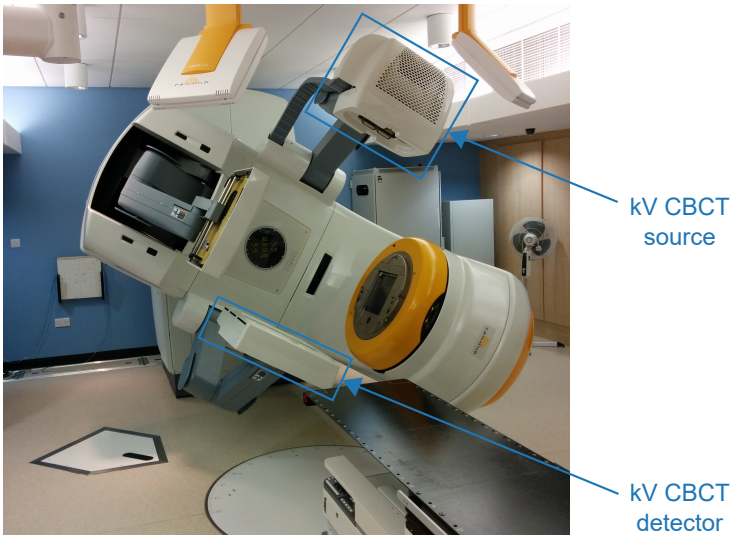


Figure 2.6: Linear accelerator with kV CBCT system

2.3 Adaptive Radiotherapy

Although IGRT can compensate for some changes in patient anatomy at treatment delivery, IGRT cannot compensate for changes in the size and shape of organs or structures, nor for changes in the position of structures relative to one another. In these scenarios, it may not be possible to accurately replicate the dose distributions to the PTV and normal tissues that were calculated in the original treatment plan.

An approach to improve the situation, termed Adaptive Radiotherapy (ART), is to use the image data of the patient's anatomy at the time of delivery to adapt the original treatment plan, either at the current fraction or subsequent fractions, to produce a dose distribution that better matches the objectives of the course of treatment.

ART is an emerging technique and optimal strategies have yet to be established, however the general concept is illustrated in Figure 2.7.

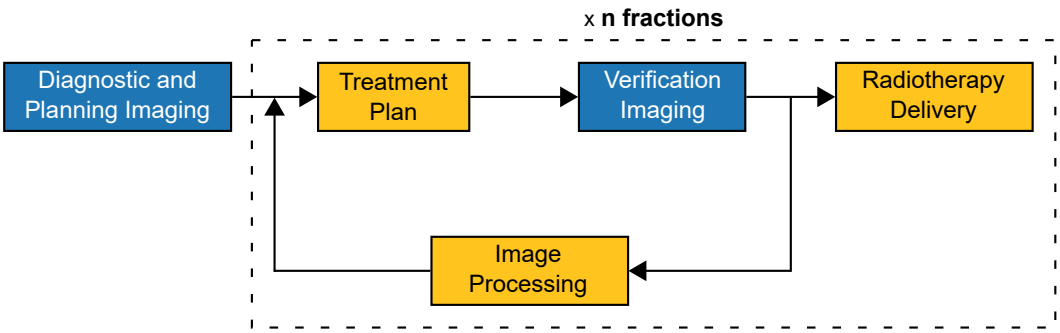


Figure 2.7: Treatment pathway for ART

The diagnostic and planning imaging are acquired in much the same way as with conventional radiotherapy, typically several days prior to the delivery of the first treatment fraction. Likewise, the initial treatment plan also tends to be created in a similar manner to standard radiotherapy.

It is at the delivery of a radiation fraction that the ART process diverges most significantly from conventional radiotherapy. In ART, images of the patient's anatomy acquired around the time of treatment delivery are compared to those used to create the original treatment plan. Any changes in the patient's anatomy identified by this comparison can then be used to adapt the original treatment plan in order to deliver the

prescribed radiation dose more accurately. However, the comparison of the planning and treatment image data, and the adaptation of the original treatment plan are both non-trivial tasks that substantially increase the effort involved in delivering an ART fraction compared to a standard radiotherapy fraction.

Two broad approaches that have been proposed are online and inline ART [27]. In online ART, the treatment plan is adapted based on the image data acquired immediately before treatment delivery. This approach can compensate for interfraction organ motion and deformation.

Inline ART is more ambitious, with the aim of periodically or continuously imaging the patient during treatment delivery and adapting the treatment in real-time. This approach is predicated on the availability of suitable real-time imaging techniques during treatment delivery.

ART techniques using real-time planar x-ray imaging have been demonstrated [28], however interest in inline ART has particularly grown given recent advances in treatment machines integrated with MRI capable of producing volumetric image data during treatment delivery [29, 30].

2.3.1 ART Workflow

A range of techniques have been suggested for online ART. This section provides an overview of a typical online ART fraction and briefly reviews the more prominent published approaches.

The workflow for each online ART fraction generally follows the same five steps:

1. Patient set up and image acquisition;
2. Re-contouring;
3. Plan adaptation or re-optimisation;
4. Quality Assurance (QA) testing of adapted plan;
5. Treatment delivery.

Patient Set Up and Image Acquisition

At the outset of a fraction, the patient is positioned in the treatment machine so as to replicate, as closely as possible, their position when the treatment planning images were acquired. New images of the patient's anatomy are captured to be used to adapt the treatment plan.

Confidence in the accuracy of the image data acquired at the start of the fraction, in terms of representing the geometry of a patient's anatomy, degrades over time due to the likelihood of intrafractional organ motion [31, 32]. The time taken to perform steps 2-5 is therefore critical in achieving the aim of ART, and these steps should be performed as fast as possible [3, 4].

In order to adapt the original treatment plan to accommodate the information obtained from the treatment delivery images, the anatomical structures of interest (target volumes and OARS) must be identified on treatment delivery images and an appropriate correspondence established with those in the planning images. The original treatment plan can then be adapted based on this correspondence.

Typically, the structures of interest are delineated on the planning model manually on a slice-by-slice basis and this process takes 30-60 minutes, depending on the location of the tumour [24]. Substantial changes in the geometry of the patient's anatomy can take place during this timeframe, for example, due to normal physiological processes, such as digestion, or the patient changing position. Therefore, automated image processing is essential for ART.

Re-contouring

Many of the online ART techniques proposed use deformable image registration to transfer the contours from the planning model to the image data acquired at the start of each fraction, and require manual verification and adjustment [29, 30, 33, 34]. The approach suggested in [4] copied the contours from the planning model to the image data acquired at the start of each fraction.

Image segmentation, which is the focus of this work, can be used to identify salient anatomical features to base the registration on [35–38]. Such an approach to re-contouring is illustrated in Figure 2.8. It is envisaged that the output of the segmentation work presented in this thesis would form the input to the next stage of a deeper image processing pipeline, which would complete the task of re-contouring.

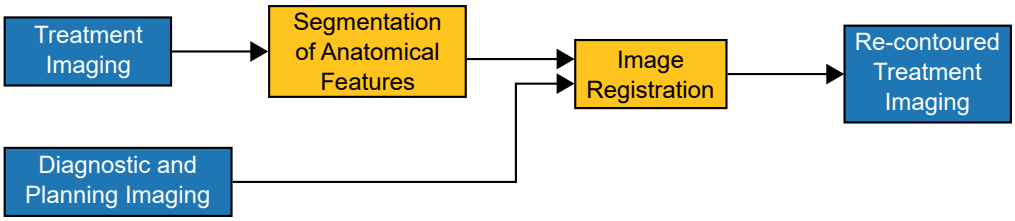


Figure 2.8: Illustration of the envisaged re-contouring processing pipeline

With the exception of [4] and [34], the time taken specifically to carry out the re-contouring step was not reported. In [4] re-contouring was reported to take around three minutes, while in [34] the mean re-contouring time was ten minutes and ranged between five and twenty-two minutes.

Plan Re-optimisation

Plan re-optimisation using the same number and direction of radiation beams and the same dose objectives as the original plan, but re-optimised based on the contours from the latest image data, is widely suggested [3, 4, 29, 30, 33, 34].

A typical plan re-optimisation time of several minutes is reported in the majority of cases, with times of 2.5 minutes reported in [3], 3 minutes in [4] and around 5 minutes in [30].

In [34], the time for plan re-optimisation was not specifically reported, however the time for plan re-optimisation and QA testing was, with a median time of 14 minutes and a range between 8 minutes and 40 minutes.

There is a substantial range in the times reported by [34], however they also reported some of the causes for increased time, which included the operator not being familiar with the planning tools, contouring errors not being noticed until the plan was evaluated and the need to make a significant modification to the treatment plan.

QA Testing

In many of the online ART approaches proposed, the QA testing of the re-optimised treatment plan included an independent, automated dose calculation [29, 30, 33, 34]. The time taken specifically for this calculation to be performed was not reported, although [34] did report the time for plan re-optimisation and QA testing.

An independent dose calculator has been developed specifically aimed at QA testing for ART [39]. This aimed to produce a tool that would be able to calculate dose using a Monte Carlo method in the timeframe of several minutes or less, based on the ART time reported in [33]. In order to meet the timing requirements, the software was optimised for execution on a Graphics Processing Unit (GPU), rather than the more ubiquitous CPU, since the hardware architecture of a GPU was better suited to the algorithm [39]. The time reported for a full dose calculation with their system was 2.3 minutes [39].

Treatment Delivery

The treatment machine is configured with the parameters from the re-optimised treatment plan and the delivery of radiation to the patient can begin.

2.3.2 State of Online ART

Two main approaches have so far been adopted for online ART. The first, and thus far more prevalent, is the plan library approach [40]. In this approach, multiple plans are created at the treatment planning stage, with each plan based on an anticipated variation in the patient's anatomy. At each treatment fraction, the plan re-optimisation stage requires the selection of the plan best suited to the patient's anatomy at the time of the fraction. Such an approach is limited to tumour sites where there are frequent

and predictable interfractional changes in the anatomy [40]. The image quality of CBCT tends to be sufficient for the images acquired at the time of treatment for this approach, making it widely accessible to most modern radiotherapy institutions [40].

The second approach is daily online planning, where the original treatment plan is re-optimised at each treatment fraction using the image data acquired at that fraction. This approach is not limited to tumour sites where the interfraction motion is predictable, however, it is more resource intensive than the plan library approach [40]. It also tends to require higher quality treatment images than the plan library approach, currently necessitating more specialised radiotherapy equipment, such as MRI-linacs [40, 41].

Online ART is still a relatively uncommon technique, although there is demand to broaden its usage [40]. There are a number of barriers to achieving this. At present, the main barrier is the increased staffing levels required. Increasing the automation of the re-contouring and plan re-optimisation stages are likely to be required to address this [40].

Another challenge to the widespread adoption of online ART is the availability of high quality imaging at the time of the treatment fraction [40]. ART is typically attempting to address changes in soft tissue. The imaging modalities currently commonly available at the time of treatment, such as CBCT, are not best suited to visualising these changes, thus hindering the clinical implementation of online ART. Although improving the quality of CBCT imaging is an area of active research, at present MRI provides much superior soft tissue contrast [40, 41]. The availability of MRI at the time of treatment, with the advent of devices such as MRI-linacs, may prove to be a watershed moment in the adoption of online ART.

Online ART also requires new tools that are fast and efficient to simplify the re-contouring, plan re-optimisation and QA testing workflow. These are required to minimise the increases in treatment time required for online ART [40, 41].

2.3.3 Inline ART

Techniques that extend the online ART paradigm through the use of imaging acquired during treatment delivery have been proposed [28, 29, 42]. The use of real-time imaging places higher demands on the execution time of the automated image processing proposed for ART in order to process the image data as it is produced. However, the tighter planning margins resulting from techniques such as online ART also increase the need to address intrafractional changes in radiotherapy [41].

The work presented in [42] proposed to extend online ART to adapt the treatment plan between each radiation beam in a fraction. They utilised orthogonal two dimensional MRI data obtained during treatment delivery to characterise the motion of the tumour and to reconstruct the dose that was delivered [42]. The remaining beams in the fraction would then be adapted to reflect the most recent characterisation of the tumour motion and to compensate for any inaccuracies in the dose delivery in the preceeding beams [42]. An orthogonal set of two dimensional MRI data was able to be produced in 360ms [42]. The treatments simulated in [42] were planned with six beams and the average time for plan adaptation between each beam was 25.6s, much faster than plan adaptation times reported elsewhere. Overall, the simulated mean fraction delivery time was less than two minutes longer than if a non-adaptive approach was taken, 18.5 minutes versus 16.7 minutes [42].

Similarly to the work presented in [42], the technique proposed in [28] used two dimensional imaging acquired during treatment delivery to adapt the treatment. However, they used x-ray imaging and tracked the tumour by dynamically changing the position of the treatment beam aperture [28]. In addition, if the tumour moved beyond a certain range the treatment beam would be held off, or gated, until the tumour returned closer to the original position [28]. Tracking the tumour using the image data was simplified by implanting the area containing the tumour with fiducial markers that have high contrast compared to the surrounding tissue in x-ray images. The x-ray images were generated at a rate of 10Hz [28].

The technique proposed in [29] also used treatment beam gating to deliver the treatment. In this case, the gating control was based on three dimensional MRI data obtained during treatment delivery [29]. The rate at which the image data were produced was not reported, however, [30] reported monitoring treatment delivery using three dimensional MRI data that was generated in 7s.

2.4 ART in Pelvic Cancers

Cancers located in the pelvic cavity are particularly well-suited for the application of ART. This is due to the variability in position, size and shape of the tumour and surrounding organs caused by processes such as respiration and variable bladder and bowel filling [19, 43, 44]. Bladder and prostate cancer are two such cancer types that have been specifically considered in this thesis.

2.4.1 Bladder Cancer

Bladder cancer comprises around 3% of cancer cases globally and in the UK [9, 45]. In 2017 it was the eleventh most common type of cancer diagnosed in the UK [45]. It accounted for 2% of cancer deaths globally in 2018 [9] and 3% of cancer deaths in the UK in 2017 [45].

Bladder Cancer Anatomy

The urinary bladder is a muscular bag located in the anterior portion of the pelvis, as shown in Figures 2.9 and 2.10.

The bladder is bounded anteriorly and laterally by the pubic symphysis and the bones of the pelvis and is attached to these by ligaments [7, 19]. The superior surface is covered by a layer of peritoneum, the folds of which help to stabilise the position of the bladder. Ligaments also attach the bladder to the abdomen [7, 19]. In females, the posterior of the bladder is adjacent to the rectovaginal septum, as shown in Figure 2.9, while in males, it is adjacent to the rectum, as shown Figure 2.10.

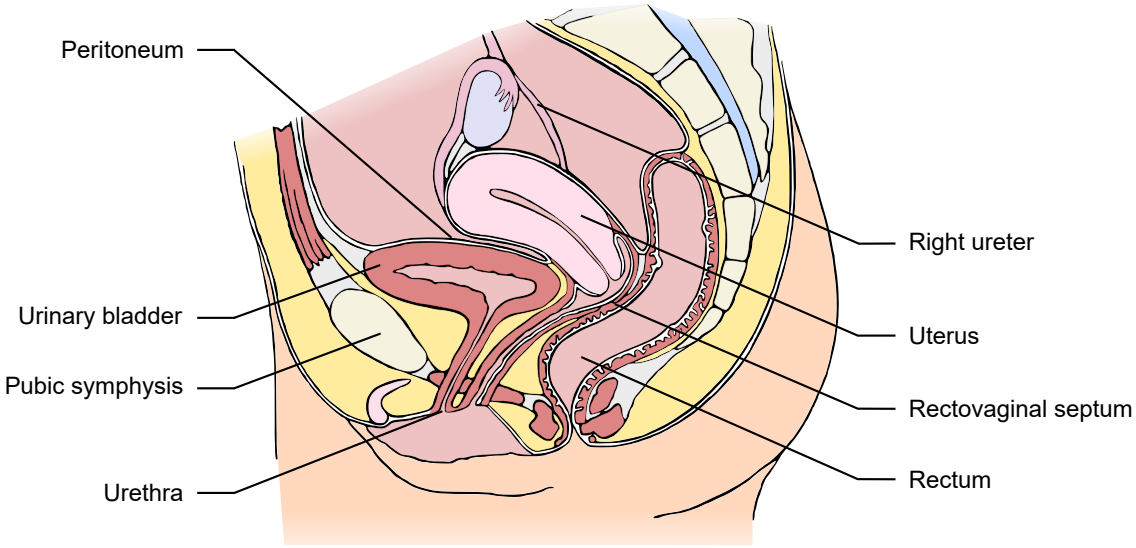


Figure 2.9: Sagittal section of female pelvis

The size, shape and position of the bladder is highly variable due to changes in the filling of the bladder itself and the filling and motion of surrounding organs. Following micturition, the bladder typically contains around 10ml of urine, while when full it typically contains around 500ml, but can contain up to 1l [7, 19].

The urethra enters the bladder at the most inferior point of the bladder called the neck of the bladder [7, 19]. The region of the bladder between the openings of the ureters and the neck of the bladder is called the trigone [7, 19].

Most bladder cancers in developed countries are located in the trigone area of the bladder, followed by the lateral and posterior walls and the bladder neck [19]. Distant metastases stemming from a bladder cancer tumour are most commonly found in the lungs, bone and liver [19].

Radiotherapy for Bladder Cancer

In the UK, radiotherapy forms part of the primary treatment for 21% of bladder cancer patients [45].

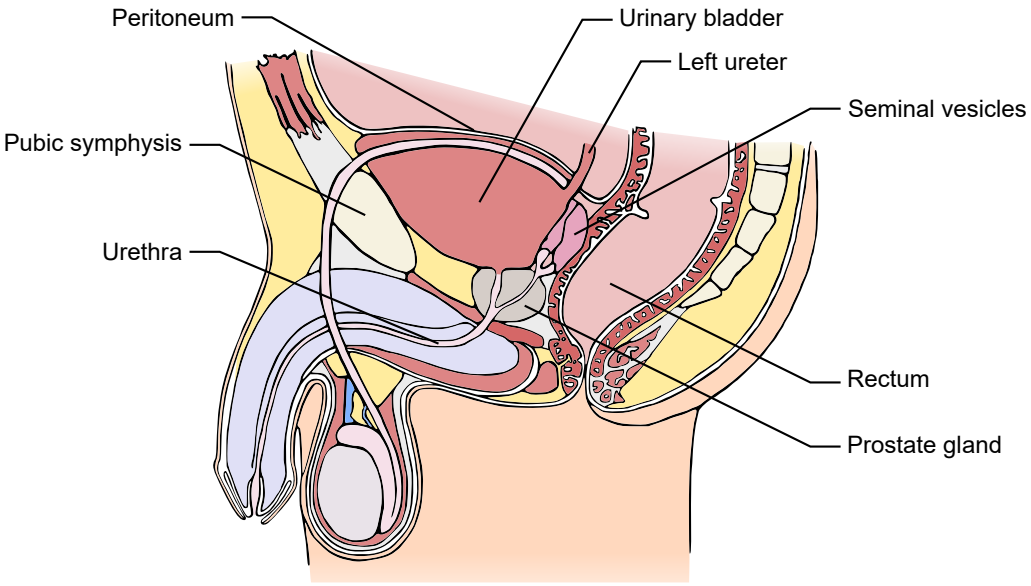


Figure 2.10: Sagittal section of male pelvis

Most bladder cancer treatments identify the whole bladder as the CTV, although some specifically identify the tumour region, at least for an escalated dose [19]. Without the high soft tissue contrast offered by MRI, accurate localisation of the tumour on images can be challenging [19].

An assortment of dose and fractionation schedules have been proposed, ranging from 50Gy to 67.5Gy in fractions of between 1.8Gy and 3Gy [19].

The OARs typically identified for radiotherapy of bladder cancer include the bowel, rectum and the bladder itself [19].

The CTV-PTV margin for conventional radiotherapy of bladder cancer tends to be quite large due to the variability of the bladder position. Margins of between 10mm and 35mm have been reported [19, 43, 46]. ART therefore has the potential to greatly improve radiotherapy for bladder cancer by increasing the accuracy of the treatment, and thereby allowing a reduction in the CTV-PTV margin.

Based on their observations of bladder intrafractional motion, [3] suggested a time of no more than ten minutes between image acquisition and the completion of fraction delivery for online ART in bladder cancer.

These time requirements appear to be shorter than the median time for re-contouring, plan adaptation and QA testing of 26 minutes reported by [33] for treating patients with pelvic malignancies. These operations therefore need to be accelerated in order for ART to become a routine clinical procedure.

Given the foreseeable nature of interfractional anatomical changes caused by variable bladder filling, bladder cancer has been a popular target for online ART using the plan library approach [40, 47]. The RAIDER clinical trial [48] is a large scale international clinical trial of online adaptive radiotherapy for muscle-invasive bladder cancer currently being conducted that uses this approach. For those patients receiving ART as part of this trial, a library of three treatment plans is created using image data of the patient with variable bladder filling. The most appropriate plan is selected based on the size of the bladder at each treatment fraction.

2.4.2 Prostate Cancer

Prostate cancer is the most common form of cancer found in males in the UK. Around 48 500 new cases were diagnosed in 2017, comprising 26% of cancer diagnoses in men [49]. It also accounted for around 12 000 deaths in the UK in 2017, which was around 14% of male deaths due to cancer, making it the second most common cause of cancer deaths in males [49].

Prostate Cancer Anatomy

The prostate gland is about 4cm in diameter and is composed of glandular tissue, smooth muscle fibres and connective tissue. It is located between the apex of the bladder and the urogenital diaphragm, as shown in Figure 2.10. The prostate gland encircles the urethra and the posterior is adjacent to the rectum [7, 20].

The seminal vesicles are also glands, which enter the prostate gland at the postero-superior surface [7, 20], as shown in Figure 2.10. The seminal vesicles are found either side of the midline, posterior to the bladder and anterior to the rectum and are surrounded by connective tissue [7, 20].

In cases of prostate cancer there are commonly found to be multiple lesions within the prostate [20]. Disease progression can result in the tumour extending beyond the prostate gland into the seminal vesicles and surrounding tissue. Bladder neck and rectum involvement are risks if the disease continues to progress [20]. Distant metastases resulting from prostate cancer most often form in the skeleton, liver and lungs but can sometimes form in the brain and other sites [20].

Radiotherapy for Prostate Cancer

Radiotherapy is part of the primary treatment of 30% of prostate cancer patients in the UK [20].

The CTV used in prostate cancer radiotherapy can vary depending on the risk categorisation of the disease. For low risk disease, the entire prostate gland tends to be identified as the CTV since prostate cancer is often found to be multi-focal [20]. For patients with higher risk disease the CTV can extend to the seminal vesicles and regional lymph nodes [20].

Radiotherapy with curative intent for prostate cancer is typically delivered in 1.8Gy to 2Gy fractions to total doses between 66Gy and 74Gy [20]. Although, SBRT has been found to produce better tumour control than conventional radiotherapy in prostate cancer, with fractionation regimes of between 33.5Gy and 38Gy in four or five fractions being proposed [21].

The OARs typically identified for prostate cancer radiotherapy include the bladder, rectum, femoral heads and occasionally the bowel [20].

The position of the prostate gland varies with changes in bladder and rectal filling [50]. PTV margins of between 5mm and 12.5mm have been reported [20, 44, 51]. These properties make prostate cancer a good candidate for ART.

For online ART in prostate cancer, [4] proposed a time of between five and ten minutes between image acquisition and the start of radiation delivery based on similar intervals for clinically implemented IGRT.

Again, these time requirements appear to be shorter than the median time necessary for plan adaptation reported by [33], further demonstrating the need for acceleration of the plan adaptation operations to enable ART.

2.5 Clinical Image Storage and Communication Infrastructure

Clinically, medical image data tends to be stored in a centralised archive called a Picture Archiving and Communication System (PACS) [52]. The PACS is typically connected to image acquisition machines such as CT and MRI scanners, treatment planning and review workstations and treatment machines by a TCP/IP based Local Area Network (LAN). An illustration of this is shown in Figure 2.11. This enables the image acquisition equipment to store image data in the PACS and for the workstations and treatment machines to retrieve image data from the PACS.

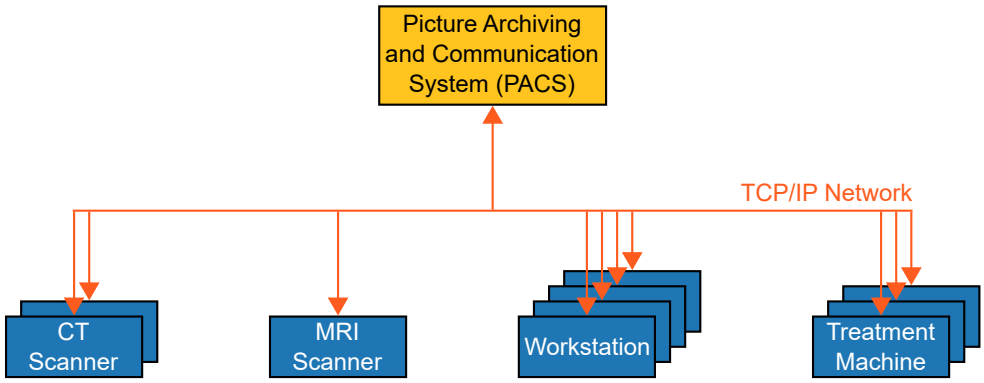


Figure 2.11: Illustration of a typical clinical local area network

2.5.1 DICOM

To facilitate compatibility between equipment from different manufacturers, a standard for storing and communicating medical image data called the Digital Imaging and Communications in Medicine (DICOM) standard is used [52].

2.5.2 DICOM Information Model

The DICOM standard can be considered in two parts: an information model and a communications protocol. Dealing with the information model first, image data in the DICOM format is typically arranged into *studies* and *series*. A *study* tends to consist of all of the image data associated with a single course of treatment for a patient and is composed of one or more series. A *series* is typically the image data obtained from a single image acquisition procedure, for example a CT scan [52]. The main imaging modalities currently used in radiotherapy, CT and MR, acquire data in three spatial dimensions, which is commonly represented by a set of DICOM files, each containing the image data for a single transverse slice [52]. Figure 2.12 provides a simplified illustration of this model.

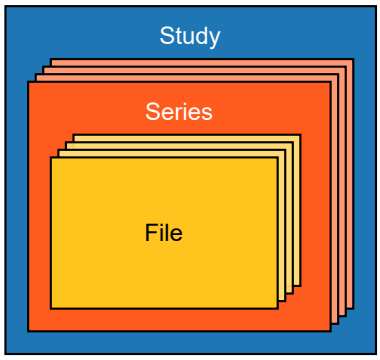


Figure 2.12: Illustration of the DICOM information model

All of the image data acquired for a single course of radiotherapy would typically be contained in a single study. That study would consist of one or more series, one for each imaging procedure the patient has undergone.

In the course of an ART fraction, two sets of image data must be transferred to the system adapting the original treatment plan:

(i) image data captured prior to the fraction, including the image data used to create the original treatment plan;

(ii) image data captured at the start of the fraction to be used to adapt the original treatment plan.

Figure 2.13 illustrates the transfer of these image data in the context of an ART fraction.

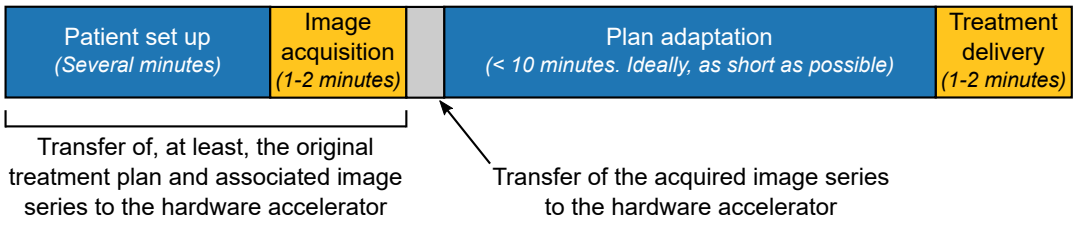


Figure 2.13: Illustration of image data transfer in the context of an ART fraction

The image data captured prior to the fraction will, at the very least, consist of the image series used to create the original treatment plan. Depending on the approach adopted for plan adaptation, however, it could include all of the image data acquired prior to the fraction, as a study. As can be seen from Figure 2.13, there is a considerable window of time in the workflow of an ART fraction during which this data could be transferred. Therefore, the transfer rate of this data is less likely to be critical to the feasibility of hardware acceleration for ART.

The transfer rate of the image series acquired during the fraction, on the other hand, is likely to be critical to the feasibility of hardware acceleration for ART. As is also shown in Figure 2.13, the time taken to transfer this image data delays the plan adaptation process, and increases the critical time interval between image acquisition and treatment delivery.

2.5.3 DICOM Communications Protocol

The communication of data using the DICOM standard occurs between what are termed Application Entities (AE) [52]. In radiotherapy, the PACS and treatment machine are two such AEs commonly involved in exchanging data using the DICOM standard.

The first step in communicating data between two AEs using the DICOM standard is to negotiate an association between the AEs. The negotiation establishes an agreed method of serialising the DICOM data to be communicated that is common to both AEs.

The communications protocol portion of the DICOM standard is based on services provided by the DICOM Message Service Elements (DIMSE), implemented by the AEs [52]. The class of DIMSE most relevant to ART are the composite DIMSE (DIMSE-C), since image files are composite information objects.

The DIMSE-C class of services are shown in Table 2.1. Typically, DIMSE-C services involve two AEs. For each service, one AE plays the role of the Service Class Provider (SCP), while the other is the Service Class User (SCU). The SCP implements the service requested by the SCU.

Table 2.1: DIMSE-C services [52]

DIMSE-C service	**Description**
C-ECHO	Verifies communication between two AEs
C-STORE	Sends an information object from one AE to be stored by another
C-FIND	Requests information about information objects stored by another AE
C-GET	Retrieves information objects stored by another AE using the C-STORE service
C-MOVE	Instructs another AE to send stored information objects to this AE, or a third party AE, using the C-STORE service

2.5.4 Data Transfer Process

The process of transferring data from non-volatile storage in one system to non-volatile storage in another typically involves moving the data in a number of intermediary steps. The data from non-volatile storage is first read into memory, from whence it is arranged for transmission over the network connecting the two systems using the appropriate protocols. Data received from the network is initially buffered in memory before being written to non-volatile storage.

The performance of the intermediary steps affects the overall performance of the data transfer.

Direct Memory Access Controller

Compared to memory, access speeds for non-volatile storage tend to be very slow [53]. Rather than tie-up the CPU waiting for data to be transferred between memory and non-volatile storage it is common to use a Direct Memory Access Controller (DMAC) to handle the data movement. The CPU programs and initialises the DMAC to perform the data transfer, allowing the CPU to perform other tasks in the meantime. Once the data transfer is complete, the DMAC can indicate this either by generating an interrupt or by setting a signal that is polled by the operating system [53].

Reading and Writing Non-volatile Storage

The data held on non-volatile storage is usually accessed in units of adjacent bytes called blocks. The size of a block is determined by a combination of the filesystem used to organise data on the disk and the physical properties of the non-volatile storage device. However, performing non-volatile storage access entirely on individual blocks can lead to very poor data transfer performance [53]. Instead, the operating system attempts to combine requests for adjacent blocks of data and service them in one access to the storage device.

This strategy greatly improves the performance of accessing data that is stored sequentially on the non-volatile storage device, as files tend to be. To enable this, requests made to read or write data on the non-volatile storage device on many common operating systems, such as Linux-based ones, are not serviced immediately, but are scheduled in a queue. When the operating system comes to service the requests in the queue, it is able to look for and combine requests for adjacent blocks on the device to be serviced in a single access [53].

Read operations on non-volatile storage are generally more critical for system performance than write operations. This is because there is likely to be some operation carried out on the data being read from storage that cannot proceed until the data is in memory. Therefore, many common operating systems, such as Linux-based ones, tend

to defer writing to non-volatile storage by buffering data in memory longer than for read operations. This leads to a greater number of read requests, for smaller amounts of data, being serviced on average than write requests [53].

2.6 Conclusion

Radiotherapy is frequently used in cases of pelvic cancer. The therapeutic ratio of radiotherapy is inherently bound to the accuracy with which the ionising radiation can be delivered to the tumour.

Recent developments in radiotherapy techniques that seek to increase the therapeutic ratio, such as the use of protons or hypofractionation, require even greater accuracy than conventional techniques. This is particularly the case for pelvic cancers, such as bladder and prostate cancer, that are highly susceptible to changes in the position of the tumour and surrounding anatomy.

ART aims to improve the accuracy of radiotherapy by adapting the treatment plan at the time of delivery based on images of the patient's anatomy acquired immediately before or during treatment delivery. This requires fast image processing as part of the adaptation process.

To make ART clinically viable for pelvic cancers, such as bladder and prostate cancer, the plan adaptation process must take less than ten minutes.

Currently, the plan adaptation process cannot be completed within the time constraints required for ART. The plan adaptation process, including image processing, therefore needs to accelerated. One possibility for achieving this is to implement the adaptation process in custom hardware.

DICOM is an industry standard for the storage and communication of medical image data. It is likely that a system intended to accelerate image analysis for ART would need to implement a portion of the DICOM standard in order to integrate and communicate with existing radiotherapy equipment.

Chapter 3

Hardware Acceleration using FPGA and Systems on Chip

The timing requirements of ART present a challenge for processing the image data acquired during each fraction to adapt the treatment plan using conventional CPUs.

A potential solution to this problem is to use alternative processing architectures to the traditional CPU that are better suited to the specific processing task, thereby performing the processing faster. This approach is called hardware acceleration.

This chapter briefly outlines the general principles of hardware acceleration. FPGAs are introduced as programmable logic devices that can be configured to create hardware accelerators. SoCs that combine FPGA with other processing architectures on a single device are also presented, along with their potential to be used for ART. The use of FPGAs for accelerating image processing and analysis algorithms is discussed, with specific consideration given to medical applications.

3.1 Principles of Hardware Acceleration

Hardware acceleration seeks to improve the performance of an algorithm by implementing it on a more appropriate architecture than a traditional CPU. This section provides an overview of CPU hardware architecture and introduces alternative processing architectures that are commonly used for hardware acceleration. The advantages and disadvantages of each of the architectures is briefly discussed.

3.1.1 CPU

The distinction between different processing architectures is not always clear-cut [54]. However, conventional CPU hardware architecture is typically designed to optimise serial code execution with low latency. This enables CPUs to perform a wide variety of tasks well by optimising the single thread performance [55–57]. Powerful arithmetic and logic units tend to be used to keep the latency of computations low [57]. To reduce memory access latency, CPUs often use larger caches than other architectures [56,57]. CPUs also tend to have more complex control hardware to enable sophisticated execution control features, such as out-of-order execution and branch prediction [56,57].

In hardware acceleration, gains in performance are generally made by performing more operations in parallel than on a traditional CPU, although some improvement may also be gained by the reduction or complete eradication of fetching instructions from memory and decoding them [58].

Parallelism can be obtained at different granularities: from fine-grained data parallelism, where the same operation is performed on multiple data elements in parallel, to task-level parallelism, such as pipeline parallelism, where multiple stages of the algorithm execute concurrently with each operating on the output of the previous stage [59].

The scope for hardware acceleration and the most appropriate processing architecture is determined by the characteristics of the algorithm. Some algorithms are inherently sequential in nature and offer few opportunities for exploiting parallelism to improve performance compared to execution on a CPU. Others offer ample oppor-

tunities to process multiple data elements in parallel, while some are best suited to task-level parallelism. More complex algorithms are likely to contain sub-routines that are best suited to a mixture of hardware architectures [55, 60, 61].

Many modern CPU architectures provide support for exploiting parallelism to improve processing performance. These include having multiple processing cores and including specific Single Instruction Multiple Data (SIMD) hardware to apply a single instruction to multiple data elements in parallel. However, the complexity of CPU architecture typically limits the number of processing cores that can be implemented in a single CPU device while still satisfying thermal and power requirements [55, 56].

CPU clusters have also been used to exploit task-level parallelism, although the communication overhead involved with this arrangement can limit the acceleration that can be achieved.

3.1.2 Alternative Processing Architectures

GPU

GPU architectures are massively parallel. Generally, they are composed of an array of multi-processors, which can each execute a task in parallel [55–57, 62]. Each multi-processor is itself composed of an array of processing elements with a SIMD architecture [55–57, 62]. The processing elements within a multi-processor tend to share a memory cache, however, there is typically no cache memory shared between multi-processors [57]. The processing elements are usually much simpler than those used in CPUs, with less powerful arithmetic and logic units and less complex control hardware. This makes the processing elements in GPUs more energy efficient, but also gives them greater latency than those in a CPU [56, 57]. The GPU architecture trades-off the low latency design of the CPU for greater parallelism in order to achieve greater throughput in circumstances where this parallelism can be exploited [55–57].

GPU architectures are well suited to performing operations on multiple data elements in parallel and suit algorithms where the computationally intensive tasks can be performed in this way.

GPUs are programmed using software. Development for GPUs is similar to that for CPUs, with extensive support for high-level languages. This makes migrating CPU implementations to GPU simpler than with some other architectures [55, 62].

Digital Signal Processors

As their name suggests, digital signal processors are processors whose architecture has been optimised for performing operations commonly used in digital signal processing algorithms. Specifically, their architecture tends to be centred around performing multiply-accumulate operations efficiently. This makes them well-suited to implementing operations such as digital filters and fast Fourier transforms [54,63]. They typically have separate buses for instructions and each of the data operands, allowing instructions and the data the instructions will operate on to be fetched from memory simultaneously. This, coupled with optimised processing elements, such as fast multipliers, can enable digital signal processors to perform a complete multiply-accumulate operation each clock cycle [54, 63, 64].

Many modern digital signal processors exploit greater parallelism by having multiple sets of processing elements. Each set of processing elements can be used to perform the same operation on multiple sets of data concurrently in a SIMD configuration. Alternatively, multiple instructions can be packed into a Very Long Instruction Word (VLIW). Each instruction in the VLIW is then executed in parallel on a separate set of processing elements with its own data operands [54, 63–65].

Digital signal processors are frequently used in applications with hard real-time requirements. Deterministic operation, rather than minimum execution time performance, has therefore tended to be favoured [54, 63, 64]. Their relatively low cost and power consumption has also made them a popular choice of processor for applications sensitive to these parameters [54, 64, 65].

Development for digital signal processors is comparatively simple, as they are programmable by software [54,63,64]. Although there are compilers that allow applications for digital signal processors to be developed in high-level languages, such as C, they are often programmed in assembly to fully optimise their performance. This can make development for a digital signal processor more labour intensive than for a CPU [54,64].

Customised Hardware

An alternative approach to hardware acceleration from the use of general purpose processors, such as CPU, GPU and Digital Signal Processors, is to design a hardware architecture specifically optimised for the algorithm to be accelerated.

Customised hardware enables the most appropriate levels of parallelism to be used for each sub-routine in the algorithm [59] to extract the best performance. However, developing an Application-Specific Integrated Circuit (ASIC) is time-consuming and expensive, with high up-front Non-Recurring Engineering (NRE) costs [59, 66]. Unless the developed ASIC is produced in very high volume, or is being developed for an application with extreme operating requirements, these costs can prove prohibitive [66, 67].

In these situations, FPGAs can offer a solution in the form of highly configurable hardware with much lower NRE costs than an ASIC [59, 66, 67].

3.2 Hardware Acceleration in ART

Hardware acceleration for ART has been the subject of considerable research. Much of the existing research in this area has focussed on the use of GPUs [68], although other platforms have also been considered.

GPU

A number of researchers have used GPUs to accelerate image registration algorithms for transferring contours from planning image data to the image data obtained at the time of treatment delivery. Registration of two-dimensional with three-dimensional images

was accelerated in [69, 70]. Deformable image registration algorithms were accelerated in [71–73]. The work in [71] showed a speed-up of between 34 and 39 times using a GPU compared to a multi-threaded CPU implementation.

Treatment plan re-optimisation strategies that leverage GPU acceleration have been proposed in [3, 42, 74–76]. The approaches presented in [3, 42] were based on GPU-accelerated radiation dose calculation algorithms. Dose calculation algorithms, which can form the basis of plan re-optimisation and QA testing in ART, have been another popular target for GPU acceleration [68,77]. A speed-up of around 200 times for a dose calculation algorithm running on a GPU compared to a CPU was reported in [78]. A similar dose calculation algorithm aimed at proton-based ART was reported to have been accelerated using a GPU in [27]. Another approach to dose calculation, based on Monte Carlo simulations, was proposed for acceleration using GPUs in [39, 42, 79].

Cluster Computing

Cluster computing has also been investigated for accelerating ART algorithms, most recently in the form of performing dose calculations using cloud computing [77, 80–82]. Cloud computing services provide large amounts of computational resources on an on-demand basis [77,80,82]. These services were used to accelerate Monte Carlo simulation based dose calculation algorithms in [77, 80–82]. The performance of the algorithms was found to scale linearly with the number of computing nodes employed [77, 81, 82], with a 1258 times speed-up compared to single-threaded CPU implementation when using 240 nodes [81].

FPGA

Compared to GPUs, there has been relatively little research published on using FPGAs for hardware acceleration in ART.

A number of researchers have investigated accelerating radiation dose calculation algorithms using FPGAs [83–86]. A dose calculation algorithm for radiotherapy was simulated for implementation on an FPGA in [83] and was estimated to achieve a speed-up of around 20 times compared to a CPU. Similarly, a speed-up of around 20 times

compared to a CPU implementation was reported for an FPGA-based dose calculation in [86]. The work presented in [84] exploited an heavily pipelined architecture for a Monte Carlo simulation based algorithm suitable for dose calculation for small sites on an FPGA. They reported speed-ups compared to a CPU of between approximately 350 and 500 times [84].

Another Monte Carlo simulation based dose calculation algorithm was implemented on FPGA in [85]. Although in this case the specific application being considered was photodynamic therapy, there are clear similarities with dose calculations for radiotherapy. Compared to a CPU implementation, their FPGA implementation was found to complete the computation 28 times faster [85].

The work presented in this thesis seeks to contribute to addressing the relative lack of investigation of FPGAs for hardware acceleration in ART. Specifically, it seeks to consider the application of FPGAs to image analysis problems typical in ART to assess the acceleration that may be achieved compared to execution on a CPU.

3.3 FPGAs

FPGAs are devices composed of an array of configurable logic. The logic can be programmed to implement a wide variety of circuits. Moreover, the logic can be repeatedly reconfigured to entirely change the function of the circuits implemented on the FPGA.

Broadly, FPGA consist of three elements:

- configurable logic;
- configurable interconnects;
- configurable *Input/Output Blocks* (IOB).

Blocks of configurable logic can be used to implement arbitrary functions, with the connections between blocks of configurable logic being customised by programming the interconnects. IOBs can be configured for the appropriate input and output signal types with interconnects being set to route these to and from the relevant logic.

Configurable Logic

The configurable logic tends to be organised hierarchically. At the lowest level there is simple combinational logic, most often implemented using *Look-Up Tables* (LUT) in modern FPGAs, 1-bit registers or *Flip-Flops* (FF), multiplexers and fast carry logic [59,66,67,87]. The fast carry logic provides dedicated connections between neighbouring LUTs to enable the implementation of more complex functions, requiring multiple LUTs, such as arithmetic functions [59, 87]. In addition to arbitrary logic functions, LUTs can be used to implement *Random Access Memory* (RAM), *Read Only Memory* (ROM) or shift registers [59, 88].

A set number of these basic resources are provided in each elementary logic unit in the FPGA, and each block of configurable logic contains a given number of elementary logic units [59, 87].

The terminology used to describe these elements vary between FPGA manufacturers. For the Xilinx (Xilinx Inc., San Jose, CA.) FPGA devices used in this work, elementary logic units are termed *Slices* and blocks of configurable logic are termed *Configurable Logic Blocks* (CLB).

A programmable switch matrix next to each CLB controls the connections made between the elements within the CLB and between the elements in the CLB and other resources in the FPGA fabric [88].

Figure 3.1 shows a schematic representation of the structure of FPGA fabric.

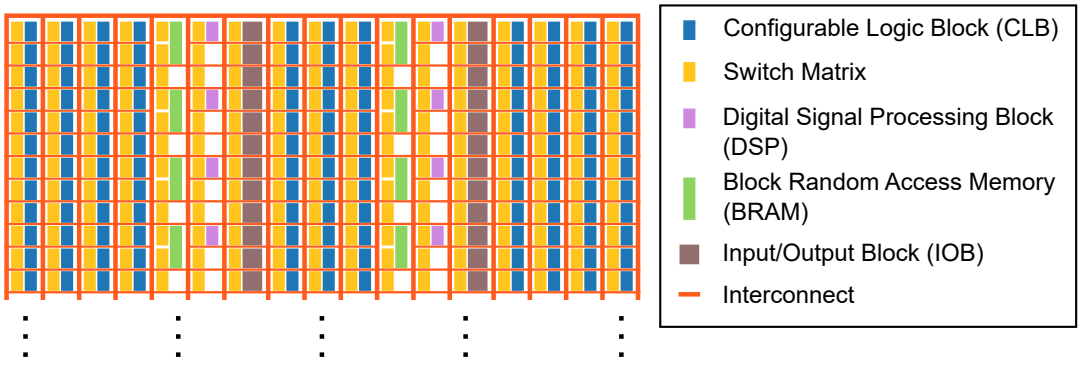


Figure 3.1: Schematic representation of FPGA fabric structure [88]

Specialised Resources

In addition to the general purpose resources, there are often more specialised resources made available in FPGA fabric, such as *Block RAM* (BRAM) and *Digital Signal Processing Blocks* (DSP) [59, 67, 87, 88].

BRAM resources provide fast access, dense RAM [59], but can also be used as ROM and *First-In-First-Out* (FIFO) buffers [88]. The dimensions of the BRAM tend to be configurable, to some extent, to help make the most efficient use of the storage space available for the word length of the data being stored [59,87,88]. For example, a BRAM able to store 2048 18-bit values may also be configured to store 1024 36-bit values.

DSPs are dedicated multipliers and adders for implementing fast arithmetic functions that consume less power than their counterparts implemented using the general purpose resources in the FPGA fabric [59, 67, 87, 88].

The work presented here makes use of FPGA-based devices manufactured by Xilinx that are based on two different families of FPGA fabric: the Artix-7 [89] and Kintex Ultrascale+ families [90]. The composition of resources in an individual slice varies between the two families, as shown in Table 3.1. Furthermore, the number of slices in a CLB varies, with the Artix-7 fabric having two slices per CLB and the Kintex Ultrascale+ fabric having only one, as shown in Figure 3.2.

Table 3.1: Resources per slice [88, 91]

Resource	Artix-7	Kintex Ultrascale+
6-input LUTs	4	8
FFs	8	16

3.3.1 FPGAs for Hardware Acceleration

Compared to the same circuit implemented using an ASIC, the FPGA implementation tends to require more transistors, operates at a slower clock frequency and consumes more power [59,67]. However, the reconfigurability of FPGAs means they are applicable

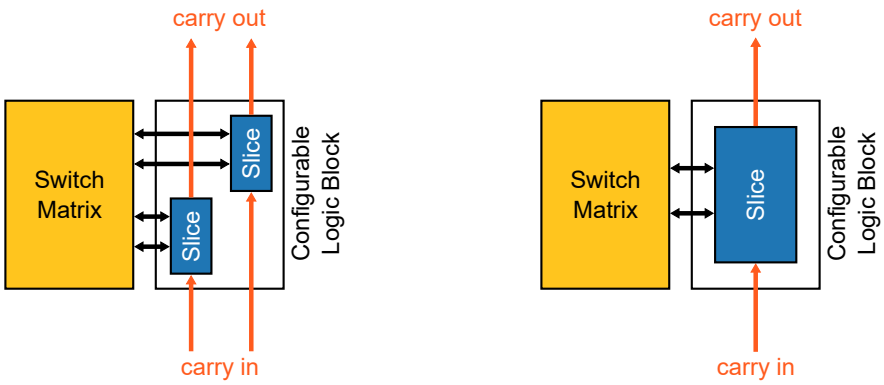


Figure 3.2: CLB structure for (a) Artix-7 [88] and (b) Kintex Ultrascale+ [91] FPGA fabric

for a wide range of applications and are mass-produced by FPGA vendors. The NRE costs of the FPGA device are therefore spread across the large production volume, making FPGAs more cost-effective for product developers than ASICs at lower volumes.

Advances in integrated circuit technology have tended to increase the number of devices that need to be produced to make the production of an ASIC more cost-effective than using FPGAs [59, 67]. In this context, hardware accelerators for ART are a relatively low volume market, making FPGAs a more cost-effective option than ASICs.

In addition, design errors are simpler to resolve later in the design cycle using FPGAs than with ASICs, reducing the design effort and making the time-to-market shorter for FPGA devices [59, 66, 67].

FPGAs can also be easily upgraded in the field to extend the lifetime of the system [59, 66, 67], enabling new and improved algorithms to be used on already deployed systems. In ART, different algorithms better suited to specific clinical sites or treatments could be loaded into an FPGA-based accelerator as required.

FPGAs Versus General Purpose Processors

FPGA implementations do not require to fetch and decode instructions; the instructions are implemented in the hardware. This provides an opportunity for improved processing performance compared to CPU and GPU architectures, which expend clock cycles performing the instruction fetch and decode operations [58, 92], although these architectures do tend to have much faster clock rates than FPGAs [92–94].

The reconfigurable hardware of FPGAs makes them well suited to implementing a variety of levels of parallelism.

Hardware can be replicated in order to perform fine-grained data parallelism by applying the same operation to multiple data values in parallel [59, 67]. This mimics the parallelism obtained using SIMD engines or GPU, although FPGA are unlikely to be able to match the scale of data parallelism offered by GPU [59] and, again, tend to operate with a much slower clock rate [93].

FPGA also have the flexibility to implement multiple different instructions in parallel. At a coarser-grained level, this enables FPGA to implement task-level parallelism in the form of pipeline-processing [58, 59, 67, 87, 93].

In a pipeline-processing architecture, an algorithm is decomposed into a series of stages, where the output of one stage forms the input to the next. By implementing each stage of the process in separate hardware the input data can be streamed between each stage. The next input data element can be streamed into the first processing stage after the previous element has passed to the second stage, which executes concurrently with the first stage processing the next data element [59, 67, 87]. The concept is illustrated in Figure 3.3.

Deep processing pipelines can be efficiently implemented in hardware [59, 67], enabling algorithms to be implemented with high data throughput [93].

FPGA Clock Frequency

The performance of an FPGA design is heavily influenced by the physical mapping of the circuit onto the resources available in the FPGA fabric [66]. The delay for a signal to propagate between two synchronous logic elements in the design determines

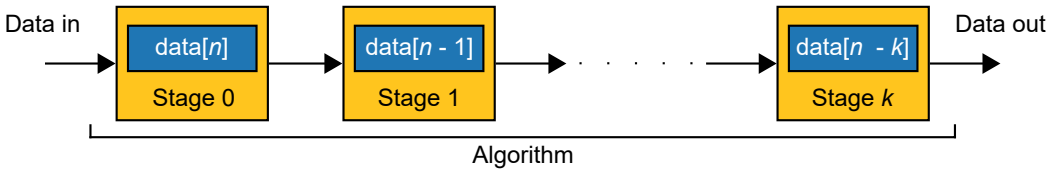


Figure 3.3: Dataflow within a pipeline-processing architecture

the maximum frequency of the clock signal used by the synchronous logic elements. This in turn influences the performance of all parts of the circuit sharing the same clock signal.

The longest propagation delay between two synchronous logic elements in a clock domain is termed the critical path. The propagation delay of a signal path depends on the routing of the path and the combinational logic elements in the path. The routing of a signal path is constrained by the physical location of the appropriate resources in the FPGA fabric and the available interconnects between them.

The critical path can be shortened by inserting an additional register between the two original synchronous logic elements, splitting the propagation delay between the new register and the original elements. This adds an additional clock cycle of latency to the signal due to the additional register, but enables a higher throughput for the entire clock domain by enabling a higher clock frequency to be used [59].

3.4 FPGA-Based SoC

Although many algorithms benefit from exploiting the parallelism offered by FPGA designs, some operations are inherently sequential in nature and are best suited to a serial processor [67, 95, 96]. Serial processors can be implemented in FPGA fabric, however the performance of these processors is limited by the clock frequency on the FPGA [67], which is typically much lower than the clock rate for conventional CPUs [93].

FPGA-based SoCs have been introduced relatively recently to address these limitations. These devices combine FPGA fabric with dedicated processors, such as CPU, GPU and real-time processors, on a single device. The FPGA fabric is often termed *programmable logic*, with the portion of the device containing the general purpose processors being termed the *processing system*, as shown in Figure 3.4.

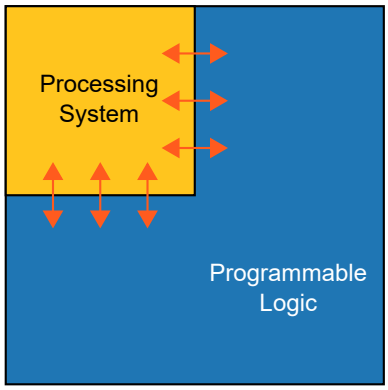


Figure 3.4: Schematic illustration of FPGA-based SoC structure depicting heavily interconnected processing system and programmable logic

Typically, the programmable logic portion of the SoC device is effectively a conventional FPGA with its own external interfaces. Likewise, the processing system generally includes external interfaces for peripherals and memory, memory caches, clock generators and interconnects [88].

Critically, the programmable logic and processing system in an SoC have a range of interfaces and interconnects between them. These enable systems to be created that exploit resources on both the processing system and in programmable logic by allowing signals and data to pass efficiently between the two parts of the device [88].

The composition of FPGA-based SoC devices varies between device manufacturers and models. The work presented here used devices manufactured by Xilinx (Xilinx, Inc., San Jose, CA). Two devices were used in this work: the Zynq-7000 XCZ7020 [88, 89] and the Zynq UltraScale+ XCZU9EG [91, 97], although only the processing system of the Zynq UltraScale+ device was utilised in this work. Table 3.2 summarises the

processors available in the processing systems of these two devices. The programmable logic of the Zynq-7000 XCZ7020 device is composed of Artix-7 FPGA fabric and the number of resources available is presented in Table 3.3.

Table 3.2: Processing system processors for SoC devices used in this work

Zynq-7000 XCZ7020	**Zynq UltraScale+ XCZU9EG**
667MHz dual core Arm Cortex-A9 CPU	1.5GHz quad core Arm Cortex-A53 CPU
	600MHz dual core Arm Cortex-R5 CPU
	667MHz Arm Mali-400 MP2 GPU

Table 3.3: Programmable logic resources on Zynq-7000 XCZ7020 device

Resource	**Number**
FFs	106,400
LUTs	53,200
36Kb BRAMs	140
DSPs	220

3.4.1 Processing System and Programmable Logic Interconnections

There are a number of interconnections between the processing system and programmable logic on the Xilinx devices. The main interconnections for passing data and control information between accelerators in programmable logic and the processing system are based on the *Advanced eXtensible Interface* (AXI) standard [88,91]. There are also other signals passing between the processing system and programmable logic, providing low-level control signals, such as interrupts and reset signals [88].

On the Zynq-7000 device there are a total of nine AXI interconnections. These include four *general purpose* interconnections, each with a 32-bit wide data bus and an estimated throughput of 400MB/s at a clock frequency of 100MHz, which are suitable for low and medium rate communications [88, 89, 98].

There are also four high performance interconnections, each with a data bus up to 64-bits wide, an estimated throughput of 800MB/s at a clock frequency of 100MHz and incorporating a FIFO buffer to support high rate communications. These are referred to as *AXI FIFO Interfaces* (AFI). The four AFIs share two dedicated ports on the processing system's memory controller for accessing off-chip memory [88, 89, 98].

The final AXI interconnection is an *Accelerator Coherency Port* (ACP), which has access to the CPU caches in the processing system and dedicated hardware to enforce cache coherency for data transferred across this interconnection [88, 89]. The ACP interconnection has a 64-bit wide data bus and similar throughput to each of the AFIs [89], but competes with the processing system CPUs for access to off-chip memory [89]. The ACP interconnection is therefore better suited to transferring data between the caches of the processing system and programmable logic. This limits the amount of data that can be transferred to avoid cache thrashing [89], where so much data is being written to cache that an excessive number of cache misses occur due to the high turnover of data in the cache. A cache miss occurs when the required data is not present in the cache.

In addition to the high speed serial processing capacity, the general purpose processors in the processing system of an SoC provide access to an ecosystem of software, such as operating systems, utilities and applications, that can significantly reduce the design effort required to implement a system on the SoC [88].

3.5 FPGA Design Methodologies

The effort required to design a system for implementation on an FPGA tends to be greater than that required when targeting a general purpose processor, such as a CPU or GPU [59,62,66]. The increasing capacity of FPGA devices has enhanced the capabilities of a single device, but has also increased the design effort required to exploit this capacity with improved functionality [95, 99].

FPGA-based SoCs add further design challenges with their more complex and heterogeneous architectures [100, 101] and the need to efficiently partition the system between the different processing architectures [88].

IP Cores

A common approach to mitigate the burden of designing a system for implementation on an FPGA is to develop and use *Intellectual Property (IP) cores*. IP cores are designed as discrete units that implement a specific functionality with a generic interface to promote interoperability with other IP cores. This approach encourages design reuse and can simplify system design for FPGAs by reducing the system to a set of interconnected IP cores.

Hardware Description Languages

Traditionally, circuit designs for FPGA implementation have been created using relatively low-level *Hardware Description Languages* (HDLs). Although HDL design techniques give the designer fine-grained, cycle accurate control of the system design, they are also time-consuming and prone to errors, and using them requires specialised skills [59, 100, 102]. This is in contrast to general purpose processors, such as CPU and GPU, where algorithms can be implemented using widely-known and high-level programming languages, and presents a significant barrier to the adoption of FPGA-based devices.

3.5.1 High-Level Synthesis

High-Level Synthesis (HLS) is an alternative approach to FPGA development aimed at increasing design productivity and the accessibility of FPGA-based devices [59, 88, 95, 99–101]. HLS uses an automated tool to convert a system specification from a high-level language into a low-level description suitable for implementation on an FPGA [59, 100]. By raising the abstraction level, specification of the system becomes simpler, with the details of the system being generated automatically by the tools, under the direction of the designer. This enables system designs to be produced more quickly than with traditional design methods [88]. The simplified system specification also reduces the amount of code required to be written manually, thereby reducing the risk of coding errors [99].

Verification of system designs can also be simplified through the use of HLS [100, 101]. Some HLS tools are able to automatically generate testbenches for synthesised hardware from software testbenches, removing the need to manually create an hardware testbench and enabling the sharing of test data [99].

The ability to rapidly implement systems increases the scope for design exploration, including different configurations of partitioning the system between hardware and software [99, 100].

The trade-off for the raised abstraction level obtained with HLS is a loss of fine-grained control over the hardware design. Instead, the designer is, to an extent, reliant upon the automated synthesis tools to efficiently implement the finer details of the circuit [88]. However, the relative abundance of programmable logic resources available with modern FPGA devices means that some implementation inefficiency can be tolerated in order to achieve signicant reductions in the design effort required.

HLS Languages

HLS tools exist that are capable of synthesising hardware designs from a number of high-level languages. Most tools support C-like languages, such as C and C++ [95, 100, 101]. C and C++ tend to be well known to software and hardware engineers alike, increasing the accessiblity of the HLS tools [95, 100]. In addition, a great many algorithms exist in these languages, simplifying their use [88,101]. C and C++ are commonly used for implementing embedded software [101], simplifying the exploration of system partitioning for systems targeted at SoC devices, as functionality can be easily moved between software and hardware [100, 101].

System-level Tools

The functionality of the HLS tools can be extended by automating the integration of the synthesised hardware into the system. For systems implemented on FPGA-based SoCs, the integration of custom hardware implemented in programmable logic with other non-configurable hardware modules and system software is often the most challenging task [96, 101], so there is strong motivation for simplifying it.

Such a tool, specifically the SDx development environment (Xilinx, Inc., San Jose, CA), was used in some of the work presented in this thesis. As well as performing HLS, this tool automatically produces software drivers and additional hardware needed to integrate the synthesised hardware with the rest of the system [91].

3.6 FPGA Acceleration of Image Analysis

The use of FPGAs to accelerate computationally intensive image analysis and processing algorithms has been widely reported [55, 60, 61, 92, 94, 102–114]. In some applications, the performance of algorithms on FPGAs has been shown to be superior to CPUs [92, 102, 103, 105–107, 110, 112] and GPUs [60, 61, 94, 104, 111, 114].

Suitability of FPGA Architecture

The architecture of FPGAs is inherently well-suited to some image processing applications, particularly low- and medium-level processing tasks characterised by high levels of parallelism, such as segmentation and classification [55, 67, 87, 92, 103]. FPGAs enable the exploitation of the data parallelism in images by allowing multiple copies of hardware to process multiple sub-regions of an image concurrently [61, 67, 92, 102, 103, 105, 107, 108, 110, 114].

They also allow the exploitation of functional parallelism through the implementation of deep processing pipelines, with separate hardware for each processing stage that is able to run concurrently [61, 67, 87, 92, 104, 106–109, 113, 114].

Data Access Patterns

FPGA-based hardware accelerators that use processing pipelines are often most efficiently realised by streaming data to the accelerator. This enables the latency of accessing data from external memory to be mitigated, to some extent, by processing data already retrieved from memory concurrently with accessing new data [67, 92, 107, 113].

In general, accelerators perform best when the number of times data is accessed from external memory is minimised [87,94,103]. This makes local image processing operations, that is ones where the operation can be completed using data from a relatively small neighbourhood, better suited for implementation on FPGAs. The relatively small amount of local data can be cached in memory resouces within the FPGA and accessed more quickly than from external memory [87, 106, 111].

Global image processing operations, on the other hand, require data from the entire image, which can exceed the amount of data that can be efficiently stored locally on the FPGA [106]. This can make the implementation of global operations on FPGAs compared to CPUs more variable, depending on the data and access patterns required by the algorithm [87].

Limiting Factors

Two common limiting factors for the performance of FPGA-based hardware accelerators for image analysis are the amount of resources available in the FPGA and the bandwidth between the FPGA and external memory. The bandwidth between the FPGA and external memory can limit the rate at which the accelerator consumes and produces data [61, 92, 94, 103, 105]. The amount of resources available in the FPGA, on the other hand, restricts the amount of hardware that can be implemented, and thus the level of parallelism that can be achieved, by limiting the number of processing stages or instances of processors that can be implemented in parallel [61,92,94,103,107].

3.6.1 Medical Image Analysis

FPGAs have been proposed for accelerating image processing and analysis tasks in a range of medical applications. Much of this work has tended to focus on broad classes of image processing and analysis tasks, such as the work presented in [112]. They demonstrated that the computation of a local image filter could be accelerated by 14.3 times compared to a CPU using FPGAs in the context of processing biomedical images. A pipeline of image filters was also accelerated using FPGAs in [61], achieving an increase in performance of more than 5 times relative to a CPU.

In addition to local filtering, [112] also demonstrated a 27.3 times speed-up of a K-means clustering algorithm, commonly used in image classification. Other authors have also reported accelerating medical image classification algorithms using FPGAs. A brain tissue classification algorithm for MRI was reported to achieve a more modest acceleration of 5 times compared to a CPU in [107].

Image Registration

Medical image registration is another area where FPGAs have been considered for acceleration. Free-form deformation, an important component of a deformable registration algorithm, was shown to run more than twice as fast using an FPGA as on a CPU [105]. Furthermore, it was reported that the performance of the FPGA-based algorithm was limited by the bandwidth to memory. If the memory bandwidth were sufficiently increased, the performance of the FPGA-based algorithm was expected to be more than 3 times as fast as the CPU implementation [105].

A deformable registration algorithm for registering CT images was reported in [106]. They reported a more impressive acceleration using an FPGA of around 30 times compared to a CPU.

Acceleration of MRI registration was investigated by [61]. In practice, the authors found that their FPGA implementation was unable to match the performance of the algorithm on either a CPU or GPU. However, simulations they performed indicated that using a much larger FPGA with greater memory bandwidth would improve the performance of the algorithm relative to both the CPU and GPU implementations [61].

Image Reconstruction

Image reconstruction is a necessary task in many medical imaging modalities, including CT and MRI. An algorithm for simultaneously reconstructing and segmenting CT images was implemented on an FPGA in [110] and demonstrated to execute 9.24 times faster than on a CPU. The same work, however, also showed that a GPU implementation of the algorithm was around 3 times faster than the FPGA implementation, albeit with greater power consumption [110].

The work presented in [111] used an image reconstruction algorithm to demonstrate their FPGA-based programming model for medical imaging applications. They showed that their FPGA implementation was able to process image frames at twice the rate of their CPU implementation and meet their real-time requirements [111]. The authors also demonstrated that their FPGA implementation was able to process image frames at around 1.4 times the rate of their implementation on an embedded GPU [111].

The time required for image reconstruction in three-dimensional ultrasound computed tomography has been reported as limiting its use clinically [114]. The authors in [114] described the performance of their algorithm, which has signal processing and image reconstruction stages, implemented on FPGA and GPU platforms. The FPGA platform was shown to perform the signal processing portion of the algorithm 10 times faster than a CPU and the image reconstruction 15 times faster [114]. The GPU platform, however, was found to outperform the FPGA platform by 2.5 times for the image reconstruction stage, whereas the FPGA platform outperformed the GPU platform by 1.6 times for the signal processing stage [114].

Specific Applications

In addition to the work considering FPGA acceleration of broad classes of image processing and analysis, there has also been work on accelerating image processing for specific medical applications using FPGAs. Real-time image processing for endoscopic diagnosis using FPGAs was reported in [109]. FPGA acceleration of image processing for localisation microscopy was shown by [104] to improve performance by 225 times compared to a CPU. The authors also demonstrated a more modest acceleration of 5 times for electron tomography using an FPGA compared to a GPU [104].

These examples illustrate the potential for FPGA acceleration of image processing and analysis in medical appplications to improve performance and meet real-time requirements. The examples also illustrate that determining the optimal computing architecture for an algorithm is non-trivial and highly application-specific.

Although some of these examples could be related to algorithms pertinent to ART, particularly those for image registration, there does not appear to be any literature where FPGA acceleration of image processing and analysis is considered specifically in the context of ART. The work presented in this thesis seeks to address this by considering the application of FPGA acceleration for image processing and analysis specifically for ART.

3.7 Selecting a Processing Architecture for Hardware Acceleration in ART

The selection of the most appropriate architecture for hardware acceleration in any application, including ART, is heavily dependent on the characteristics of the algorithms being accelerated. This section discusses the advantages and disadvantages of digital signal processors, GPUs and FPGAs as hardware accelerators in the context of ART.

3.7.1 Digital Signal Processors

Digital signal processors are well-suited to accelerating applications where the computationally intensive load can be implemented as operations commonly used in digital signal processing algorithms. Furthermore, applications suited to digital signal processors that exhibit data-level parallelism would map well onto digital signal processors that support SIMD operation [63,64]. Similarly, those that exhibit task-level parallelism may gain an advantage from using digital signal processors that support VLIW [63]. The advantages gained from exploiting such parallelism need to be traded-off against any overhead incurred by the requirement to have data specifically arranged in memory to make effective use of the architecture [63, 64].

Digital signal processors are frequently used in applications where low power consumption and cost are critical requirements [54, 64]. However, given the high capital cost and energy requirements of radiotherapy equipment, it is doubtful that the cost and power consumption of hardware accelerators for ART are going to be significant factors.

3.7.2 GPUs

GPUs are designed to accelerate algorithms with a high degree of data-level parallelism. They work most effectively when performing SIMD operations on data with no inter-dependencies [62]. Modern GPUs that contain an array of multi-processors can also achieve task-level parallelism by executing different tasks on each of the multi-processors concurrently. In algorithms with a high degree of data-level parallelism, GPUs are able to generate greater throughput than a CPU, despite typically using processing elements with higher latency. This is achieved by the GPU processing more data operands simultaneously. In algorithms where there is a low degree, or no, data-level parallelism, much of the parallel architecture of a GPU will be idle, and the low latency processing of a CPU is likely to produce greater throughput.

Even when processing algorithms with a high degree of data-level parallelism, there are some aspects of the GPU architecture that can limit the acceleration that can be achieved. GPUs do not generally support the same extent of fine-grained control flow that CPUs do. Branching within algorithms can cause SIMD threads to diverge, requiring that each branch be executed in series [55]. Synchronisation following a set of parallel computations is also not generally as efficient on a GPU as a CPU [55, 56]. This means that algorithms that consist of a lot of short parallel computations do not map well to the GPU architecture, as the synchronisation overhead begins to dominate the execution time [56].

The memory shared by processing elements within a multi-processor on a GPU tends to be relatively fast. However, it is also typically much smaller than the memory caches used in CPUs [57]. Any data that needs to be shared between multi-processors on a GPU, or does not fit into the memory dedicated to a single multi-processor, is limited to using the slower global memory shared between multi-processors. This can limit the performance of algorithms that need to make a lot of memory accesses or have irregular memory access patterns [56, 62]. Some GPUs do provide efficient hardware support for gathering operands for SIMD operations from multiple memory locations and scattering the results likewise. This can help to reduce the overhead incurred from the need to arrange data in memory to fit the SIMD operation [56].

Compared to digital signal processors, the architecture of GPU processing elements bears more similarity with that of CPUs. Unlike digital signal processors, the processing elements in a GPU are not specifically optimised for digital signal processing operations and there are not typically separate data and instruction buses. This, coupled with extensive support for high-level programming languages, makes the migration of traditional software implementations to GPUs relatively straightforward [55, 62].

3.7.3 FPGAs

Unlike either digital signal processors or GPUs, which are programmed using software, FPGAs are conventionally programmed using HDLs. While this provides fine-grained control over the implementation on the FPGA, development in HDLs also tends to be much slower and more labour-intensive than software [59, 62, 66]. HLS tools have been introduced to allow FPGA development in software programming languages, however, these are not yet as widely adopted or mature as those for digital signal processors or GPUs.

The reconfigurability of FPGAs enables the architecture to be tailored to the degree of parallelism required by the algorithm by exploiting both data-level and task-level parallelism [55, 62]. For algorithms that are predominantly data-parallel in nature however, an FPGA is unlikely to be able to match the execution time performance provided by the massively parallel architecture and fast clock rate of a GPU.

FPGAs are particularly suited to exploiting task-level parallelism in algorithms where data streaming can be used [55, 62]. These algorithms can be implemented as a series of connected processing engines where the data flows from one engine to the next, as shown in Figure 3.3 on page 51. Each processing engine itself can also be designed to take advantage of any parallelism within the stage of the algorithm that it implements. This approach tends to work well where the control and data flow are relatively simple. However, in algorithms that require complex control and data flow, the implementation of the controller in the FPGA resources can limit the performance of the algorithm [55, 62].

The performance of FPGA-based hardware accelerators in applications with large input datasets can also be limited by the availabilty of memory resources on the FPGA [55, 62]. Much of the performance advantage achieved from implementing a data streaming algorithm on an FPGA is obtained by minimising the number of costly off-chip memory accesses by buffering the data within the processing pipeline [55]. If there are insufficient memory resources on the FPGA to do this, then the data must be accessed from off-chip memory, which may detrimentally affect the execution time performance of the algorithm.

3.8 Conclusion

Hardware acceleration uses alternative processing architectures to conventional CPUs to reduce the time taken to execute algorithms. Applying this to algorithms for ART has the potential to enable the treatment plan to be adapted within the clinical time constraints.

FPGAs are programmable logic devices that can be configured to create hardware accelerators. Algorithms implemented on FPGAs can obtain performance advantages over CPU implementations for some algorithms, or portions of algorithms, chiefly through exploiting greater levels of parallelism.

Selecting the processing architecture that will give the optimal execution time performance for a given algorithm is a non-trivial task. For algorithms that are inherently serial in nature, the low-latency optimised architecture of the CPU is likely to provide the best performance. For algorithms with a high degree of data-level parallelism, the massively parallel architecture of a GPU may provide better performance than a CPU. Digital signal processors should also be considered in algorithms with a high degree of data-level parallelism. However, they are only likely to provide better execution time performance than a GPU when the algorithm can be implemented from operations for which the digital signal processor architecture is optimised, such as multiply-accumulate operations. FPGAs are well suited to exploiting task-level parallelism by implementing deep pipelines of processing engines connected in series. This suits algorithms where data streaming can be used and the control and data flow are relatively simple. FPGAs

can also take advantage of data-level parallelism, albeit not typically to the same extent as GPUs or digital signal processors. In addition, the reconfigurability of FPGAs allows the degree of parallelism to be tailored to the algorithm being implemented.

The emergence of devices with heterogeneous architectures also opens up the possibility of sub-dividing algorithms into stages and executing each stage on the processing architecture best suited to it.

Hardware acceleration for ART has been proposed previously, although much of this research has sought to use GPUs rather than FPGAs. Acceleration of algorithms that may be suitable for ART using FPGAs has been demonstrated, but this has tended to focus on radiation dose calculations.

The work presented in this thesis seeks to contribute to the relative lack of consideration of FPGAs for hardware acceleration and, specifically, their application to the image analysis problems typical in ART. Much research has been published that establishes the suitability of FPGAs for accelerating some image analysis algorithms, including for medical imaging applications. However, their application to accelerating image analysis algorithms specifically in the context of ART does not appear to have been considered previously.

Chapter 4

DICOM Transfer Rates on FPGA-based SoC Platforms

In the context of accelerating image analysis for ART, data needs to be transferred between the existing digital imaging infrastructure and the accelerating processor. Much of this information will be in the form of images, captured during the patient's treatment pathway, although it is also likely to include information about the planned radiotherapy.

A critical parameter for hardware acceleration is the overhead incurred from the need to transfer data to and from the specialised hardware. Any performance improvements gained from accelerating the algorithm must be traded-off against the additional time needed to transfer data to and from the accelerator in order to justify its use. It is therefore important when considering the viability of any system intended to accelerate adaptive radiotherapy to establish how the system would integrate with the existing radiotherapy equipment and to estimate the anticipated communication overhead.

This chapter presents the results of work carried out to characterise the communication overhead between existing radiotherapy infrastructure and an FPGA-based SoC for accelerating adaptive radiotherapy. The digital imaging infrastructure was modelled and the time taken to send and retrieve data from it was measured for two different FPGA-based SoC development boards. In all simulations, a desktop computer with a similar specification to those used for clinical radiotherapy work was used for

comparison. Clinical image data from prostate cancer patients was used to characterise transfer rates. Activity levels within the development boards and desktop computer were recorded during the tests to help identify factors limiting performance and to propose solutions.

This chapter is organised as follows. Initially, relevant previous work is discussed. A detailed description of the new work together with results is given.

4.1 Relevant Work

Any system intended to accelerate image analysis for ART would be required to integrate and communicate with existing radiotherapy equipment. As was discussed in Section 2.5 on page 34, using the DICOM standard for the communication of medical image data is highly likely to fulfil this requirement.

Generally, there is a lack of widely accepted, objective standards for satisfactory transfer rates of DICOM images in the clinical setting. Indeed, there is not a great deal of literature dealing with the topic of DICOM transfer rates at all. Much of the literature tends to base the criteria for assessing acceptability of DICOM transfer rates on the tolerance of clinicians to wait for images to be transferred.

The guidelines for purchasing and acceptance testing of PACS equipment produced by the American Association of Physicists in Medicine and the Radiological Society of North America do not give clear objective standards for the DICOM transfer rates that should be achievable with PACS equipment [115]. Instead, the subjective assessment of the time clinicians are content to wait for image data to be tranferred is used. This value is likely to vary between clinicians and between institutions with varying workloads.

Authors in [116] also based their assessment of DICOM image transfer rates primarily on the tolerance of clinicians to wait for data to be presented. Although, they additionally suggested the more objective standard of being able to display two or more full resolution images in less than 2s.

An exception to the foregoing is the work presented in [117], which was carried out to assess the requirements and feasibility of installing a PACS in a moderately sized radiology department. They considered the required throughput of a PACS in terms of

the rate at which image data were produced and reviewed prior to the installation of the PACS. They found that a peak throughput of 0.5MB/s would be sufficient to serve the departmental workload at that time. This work was carried out in 1996, however, and there has generally been an increase in the usage of medical imaging data since then, calling into question the validity of this data rate for current clinical requirements.

Rates for transferring image data over a network using the DICOM protocol were reported in [118]. The paper authors reported mean transfer rates of between 3.6 and 3.9MB/s for MR and CT images. Their work was conducted in the context of adapting existing grid computing communications mechanisms to better handle the DICOM protocol and data. They offered no comment on the rates achieved using the plain DICOM protocol compared to the rates that would be expected on a clinical LAN. However, the rates achieved with the plain DICOM protocol were superior to those achieved with their proposed communication mechanisms and to those achieved with the best performing alternative grid computing communications mechanism for handling the DICOM protocol and data [118].

Perhaps the most relevant published work is by the Oracle Corporation where they demonstrated the performance of retrieving and inserting DICOM image data into their database application [119, 120]. One of the scenarios tested in their white paper used a single server, running the database application, connected by five 1Gb/s Ethernet connections to a single client. Both the client and server were server-grade computers. The server had two 3.2GHz quad core processors, 48GB of memory and 8TB of non-volatile storage made up of 112 HDDs. The client had two 2.8GHz quad core CPUs, 64GB of memory and around 2TB of non-volatile storage. The bandwidth to non-volatile storage on the server was 8Gb/s, while in the client it was 2Gb/s.

The DICOM data used in the test was around 2TB in size, composed of 20,080 studies, around 2.4 million images from six modalities. Of these studies, 3,000, making up 64GB, were MR or CT image data, which are the most relevant modalities for the work presented here and current radiotherapy clinical practice. The MR and CT image data included in the dataset are described further in Table 4.1.

Table 4.1: MR and CT image data used in the Oracle Corporation white paper [119]

Average Study Size (MB)	Images per Study			Image Size (MB)		
	Min.	Max.	Mean	Min.	Max.	Mean
22	16	1024	64	0.06	4.2	0.36

Image data were retrieved from and inserted into the database at the study level. Image retrieval and insertion could occur in parallel for each study. Six instances of the retrieval and insertion applications also ran concurrently on the client during the respective retrieval and insertion tests [119].

Retrieval rates of 1100 images per second, equating to a data rate of 380MB/s, were reported [119,120]. This rate was limited by the available network bandwidth [120]. For image insertion, a rate of 1564 images per second, or 353MB/s, was achieved [119,120]. The limiting factor for this rate was not reported [119,120].

These transfer rates are much higher than any comparable rates reported elsewhere in the literature. These rates are due, at least in part, to the high performance hardware that was used, and reflect that the purpose of the work was to demonstrate the capability of Oracle's database application, rather than representing typical transfer rates that would be seen in a contemporary clinical setting.

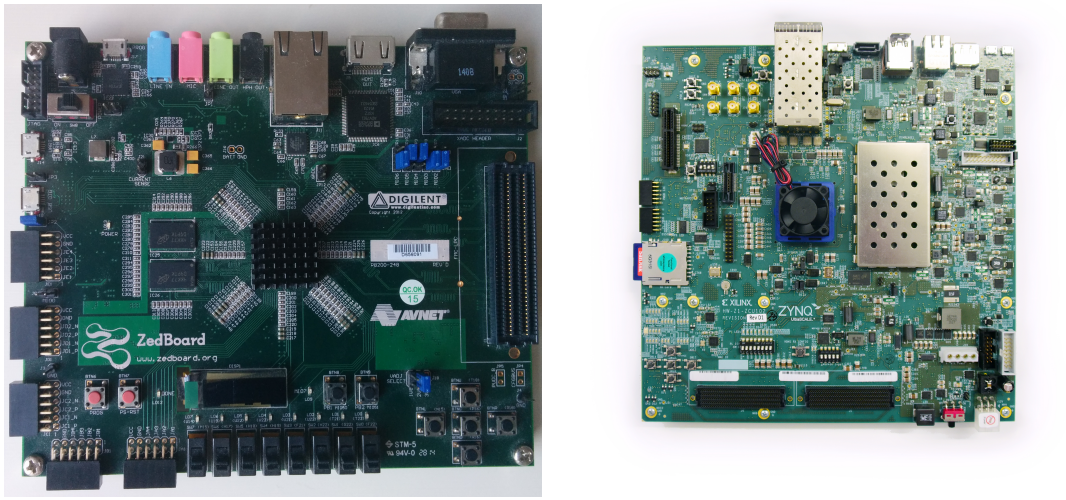
4.2 Materials and Methods

4.2.1 Hardware Used

The PACS used in radiotherapy clinics was modelled using the Orthanc DICOM server software [121]. Orthanc is an open-source DICOM store software application. It implements a database for storing DICOM files and the DICOM image storage and retrieval protocols of interest here. It has previously been used by its creators to implement DICOM stores within the clinical environment. The Orthanc software was run on a laptop computer, the specifications of which are summarised in Table 4.2. The Ubuntu 14.04 operating system was used.

Three platforms were used to test the rate at which DICOM data could be transferred to and from the Orthanc DICOM store. Two of these platforms, the Avnet ZedBoard (Avnet, Inc., Phoenix, AZ) and the Xilinx ZCU102 (Xilinx, Inc., San Jose, CA), were FPGA-based SoC development boards. The third platform used was a desktop computer with a specification comparable with modern workstations used clinically in radiotherapy departments.

The ZedBoard platform, shown in Figure 4.1(a), used a Xilinx Zynq-7000 XC7Z020 SoC device (Xilinx, Inc., San Jose, CA), which incorporated a dual core Arm Cortex-A9 CPU (Arm Ltd., Cambridge, UK) with FPGA fabric containing around 85,000 logic cells. Further details are provided in Table 4.2.



(a) (b)

Figure 4.1: FPGA-based SoC development boards. The (a) ZedBoard platform, and (b) ZCU102 platform [122].

The ZCU102 platform, shown in Figure 4.1(b), used a Xilinx Zynq Ultrascale+ XCZU9EG SoC device (Xilinx, Inc., San Jose, CA), which incorporated a quad core Arm Cortex-A53 processor (Arm Ltd., Cambridge, UK), a dual core Arm Cortex-R5 real-time processor (Arm Ltd., Cambridge, UK), an Arm Mali-400 MP2 graphics processing unit (Arm Ltd., Cambridge, UK) and FPGA fabric containing around 600,000 logic cells. The application used to characterise the data transfer rate to and from the

PACS model was executed on the Arm Cortex-A53 processor as it was believed this would provide the best performance from the device. Further details of the ZCU102 platform are provided in Table 4.2.

The specifications of the desktop computer used are also summarised in Table 4.2.

Table 4.2: System specifications

System	Processor	Memory	Non-volatile storage
PACS model	2.6GHz dual core Intel i5	4GB 1600MHz DDR3	80GB SATA 2.6, 3Gb/s HDD ext4 filesystem, 4kB block size
ZedBoard	667MHz dual core Arm Cortex-A9	512MB 533MHz DDR3	8GB class 10 SD card ext4 filesystem, 4kB block size
ZCU102	1.5GHz quad core Arm Cortex-A53	4GB 2133MHz DDR4	8GB class 10 SD card ext4 filesystem, 4kB block size
Desktop with HDD	4GHz quad core Intel i7	32GB 2400MHz DDR4	1TB SATA 3.1, 6Gb/s HDD ext4 filesystem, 4kB block size
Desktop with SD	4GHz quad core Intel i7	32GB 2400MHz DDR4	8GB class 10 SD card ext4 filesystem, 4kB block size

Each of the platforms tested ran a Linux-based operating system. This gave access to a wealth of existing software that was used to simplify the task of characterising the rate at which data could be exchanged with the PACS model using the DICOM protocol. Ubuntu 16.04 was used as the operating system on the desktop computer. For the ZedBoard and ZCU102 platforms, their respective Linux images were those supplied with the Xilinx SDSoC 2016.3 development environment software (Xilinx, Inc., San Jose, CA).

The FPGA-based SoC development boards used here were chosen primarily for their availability. As they are general-purpose boards, they have not been designed, and do not feature optimisations for the tasks required in ART. The performance measured

using these boards is, therefore, likely to be lower than that which could be attained using an FPGA-based SoC board specifically developed for ART. However, the effort required to design and produce such a board precluded this approach.

4.2.2 Measuring Transfer Rates

The software applications created to perform the tests presented here made use of the DICOM Toolkit (DCMTK) libraries (OFFIS e.V., Oldenburg, Germany). DCMTK is a set of open-source C++ libraries that implement large parts of the DICOM standard. Version 3.6.1 of DCMTK was used here.

The DCMTK libraries were used to establish a connection with the PACS model using the DICOM protocol over a 1Gb/s full duplex TCP/IP Ethernet connection. They were also used to implement the C-STORE DIMSE-C service in the case of sending data to the PACS model, and the C-MOVE DIMSE-C service in the case of receiving data from the PACS model. In the case of receiving data from the PACS model, the SCU of the C-STORE service was implemented by an application supplied with the DCMTK libraries.

The experimental setup is shown in Figure 4.2.

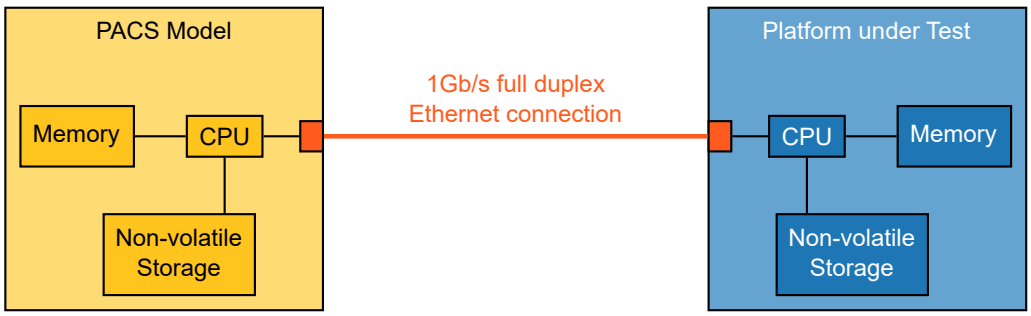


Figure 4.2: Experimental setup showing the interconnection between the PACS and the platform being tested.

Sending Data to the PACS

A C++ application was written to test the rate at which the data could be sent from the platform under test to the PACS model. This was tested for sending data on both a series- and study-wise basis. Fifty series and fifty studies were transferred for the series- and study-wise tests, respectively.

First the application sought to establish an association with the Orthanc software running on the PACS model using the DICOM protocol. Once an association was established, the application acted as the SCU, requesting the C-STORE service from the Orthanc software to store the data being sent to the PACS model. For each series or study to be transmitted, a C-STORE request was issued for each file belonging to it in turn. Once all of the data for the test had been sent, the application sent a request to release the association established with the PACS model. The flow of the application in terms of the DICOM protocol is shown in Figure 4.3.

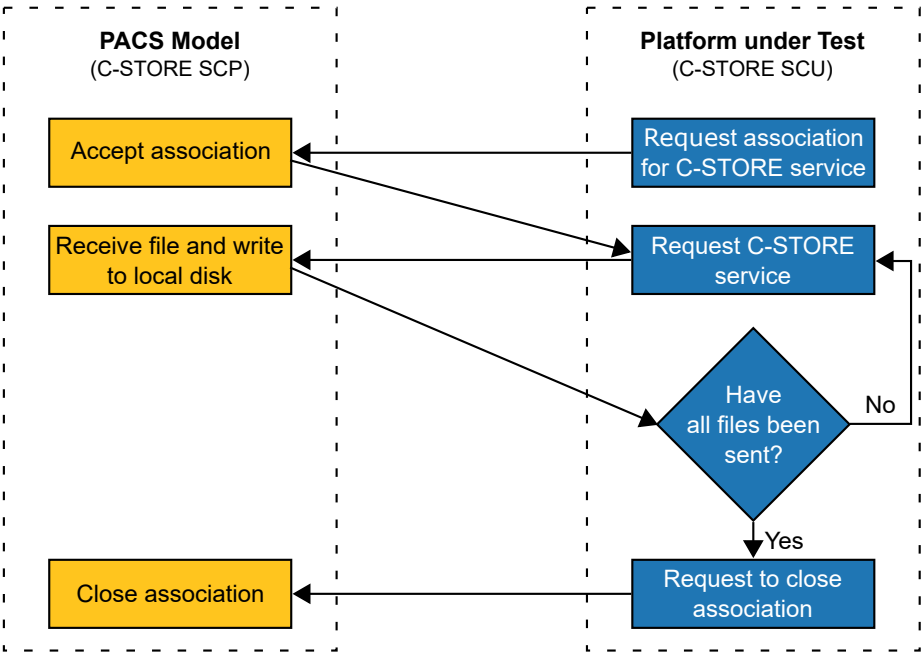


Figure 4.3: DICOM protocol for sending data to the PACS model

For each series or study, the time taken to send all of the files and receive responses from the PACS model was measured by recording a timestamp from a system clock before issuing the first C-STORE request, and recording another timestamp upon receiving the response for the final file sent. The difference between these two timestamps was used as an approximation of the time taken to transfer the series or study to the PACS model. The amount of data contained in each series or study was known and an estimate of the data transfer rate could therefore be made by dividing the amount of data transferred by the time measured to send it. The programmatic flow of the application is shown in Algorithm 1. The source code for the study-wise test is also shown in Appendix B.

Algorithm 1 Sending DICOM data to the PACS model

```
1: association ← REQUEST ASSOCIATION
2: if association = rejected then
3:     return error
4: end if
5: for all studies/series to be sent do
6:     start ← TIMESTAMP
7:     for all files in study/series do
8:         response ← SEND C-STORE REQUEST(file)
9:     end for
10:    stop ← TIMESTAMP
11:    if response = succeeded then
12:        duration ← stop − start
13:    end if
14: end for
```

Retrieving Data from the PACS

A similar C++ application was written to test the rate at which data could be received by the platform under test from the PACS model.

This application added an extra layer of complexity compared to the one used to test sending data to the PACS model. The PACS model and platform under test both had to play the role of SCU and SCP for the two different DIMSE-C services involved. The platform under test acted as the SCU for the C-MOVE service with the PACS

model acting as the SCP. However, this then precipitated the PACS model to make a C-STORE service request to the platform under test to store the data. This process is shown in Figure 4.4.

The C-STORE SCP on the platform under test was provided by the *storescp* application included as part of the DCMTK libraries.

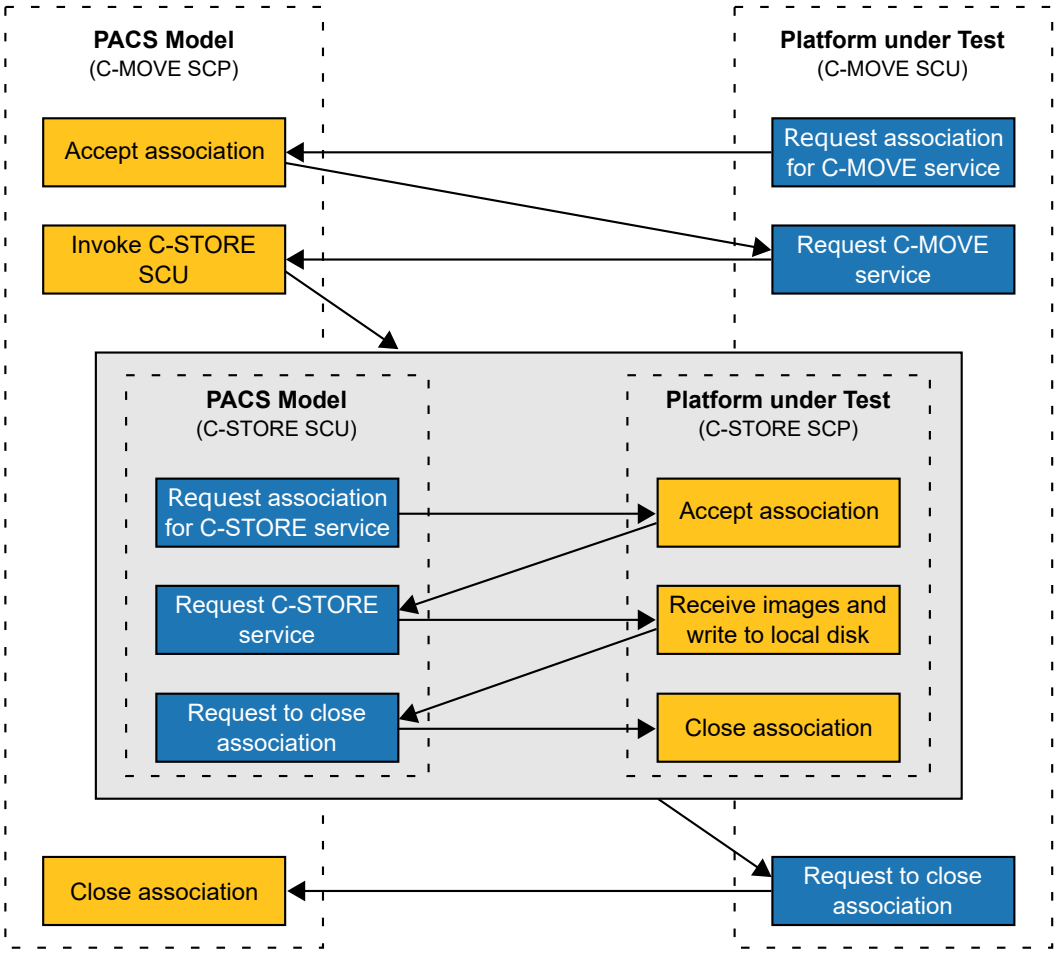


Figure 4.4: DICOM protocol for receiving data from the PACS model

Again, the application was tested for receiving data on a series- and study-wise basis, transferring fifty series and fifty studies, respectively. The time taken to transfer the data to the platform under test was estimated by the application recording a timestamp immediately before issuing the C-MOVE request, and recording another immediately after receiving the response to the request from the PACS model, as shown in Algo-

rithm 2. The difference between the two timestamps was taken as an approximation of the time taken to send the data to the platform under test. This, together with the amount of data contained in each series or study being known, allowed an estimate of the transfer rate to be made. The source code for the C-MOVE SCU for the series-wise test can be found in Appendix B.

Algorithm 2 Receiving DICOM data from the PACS model

```
1: association ← REQUEST ASSOCIATION
2: if association = rejected then
3:     return error
4: end if
5: for all studies/series to be sent do
6:     start ← TIMESTAMP
7:     response ← SEND C-MOVE REQUEST(study/series)
8:     stop ← TIMESTAMP
9:     if response = succeeded then
10:        duration ← stop − start
11:    end if
12: end for
```

Image Data

The DICOM image data used for this work was drawn from the PROSTATE-DIAGNOSIS collection [123] of The Cancer Imaging Archive (TCIA) [124]. This is a publicly available collection of MR images obtained from 92 patients diagnosed with prostate cancer. The collection comprises 92 studies containing a total of 368 series and 32,537 image files.

At the commencement of this work, this was the largest collection of images composed purely of either CT or MR images for prostate or bladder patients hosted on TCIA. A publicly available dataset was chosen to help facilitate repetition or extension of the work presented here by others.

Tests were carried out to characterise the transfer rates when sending or receiving data on both a study-wise and series-wise basis. In the study-wise case, fifty studies were pre-selected at random from the 92 available in the collection. Likewise, fifty series were pre-selected at random from the 368 available in the collection for the series-wise case.

Random selection of a subset of the studies and series available minimises efficiencies that may arise from accessing data sequentially and better reflects the transfer patterns likely to be seen in the clinical environment. The studies and series to be transferred were randomly pre-selected so that the data transferred between the test platforms and the PACS model was repeatable. This was done in order to simplify the comparison in performance between the platforms.

When testing sending data from the platform under test to the PACS model, the database maintained by the Orthanc software was initially empty. When testing sending data from the PACS model to the platform under test, the database maintained by the Orthanc software on the PACS model contained the entire PROSTATE-DIAGNOSIS collection.

Any differences found in the mean transfer rates between platforms were tested for statistical significance. A significance level of 5% was chosen for this, and all other significance tests performed. The results from this are provided in Section 4.3.2, starting on page 85.

The linear correlation between transfer rate and transferred series or study size was tested [125,126]. This was done in order to establish whether any correlation may have arisen due to efficiencies obtained through transferring larger or smaller amounts of data. The statistical significance of the correlation coefficients was also tested [126]. These results are discussed in Section 4.3.3 on page 90.

4.2.3 Monitoring System Activity

In addition to the timestamps used to estimate the time taken to transfer data, the system activity of the platforms under test was monitored in greater detail using the *sysstat* software package [127]. The *sysstat* software package is open source software for

recording and monitoring the activity of computer systems that use a Linux-based operating system. The software monitors and records a range of metrics concerning system utilisation and performance including CPU utilisation, memory usage, input/output (I/O) activity and network interface statistics. It is capable of recording these metrics at frequencies up to 1Hz.

The *sysstat* software was run concurrently with the applications to transfer data to and from the PACS model. The time intervals at which the *sysstat* software recorded system activity data were chosen empirically with a view to obtaining multiple samples of system activity during the transfer of each DICOM series or study, while avoiding collecting an excessive amount of data.

System activity data was recorded at five second intervals when sending and receiving DICOM studies and series to and from the PACS model using the ZedBoard and ZCU102 platforms. In all other cases, system activity data was recorded at one second intervals, the shortest interval afforded by the *sysstat* software.

The statistics obtained by the *sysstat* software that were of particular interest were those pertaining to CPU, non-volatile storage and network activity.

CPU

The CPU utilisation statistics were examined with a focus on the amount of available CPU time that was occupied by the applications performing the DICOM transfer tests and the amount spent waiting for non-volatile storage I/O requests to be serviced.

The applications performing the DICOM transfer tests were the only user applications executing during the tests and therefore, the percentage of CPU time spent operating at the user level recorded by *sysstat* was assumed to be a measure of the percentage of CPU time spent executing the applications.

The *sysstat* software recorded:

(i) the percentage of time the CPU spent idle;

(ii) the percentage of time the CPU spent idle but during which there was an outstanding I/O request to be serviced.

The percentage of time the CPU spent idle was used as an indication of how much spare processing capacity was available during the tests. The time spent with the CPU idle and an outstanding I/O request to be serviced was used as an indication of the time the system spent stalled, awaiting I/O.

Network

Network utilisation was recorded as the percentage of the available network bandwidth that was used. In the case of the full-duplex network connection used here, this figure represented the utilisation of the transmit or receive channel, whichever had the greatest utilisation.

Non-volatile Storage

For the tests where data was sent from the platform under test to the PACS model, the average rate of reading data from non-volatile storage was examined, while, for the tests where the data was sent from the PACS model to the platform under test, the average rate for writing data to non-volatile storage was evaluated.

The mean values calculated for the read and write speeds excluded zero values to avoid spuriously low averages for these statistics.

Sampling Rates

As touched on earlier, the *sysstat* statistics were sampled at different rates on various platforms in some of the tests. Moreover, the diverse transfer rates on certain platforms meant that the duration of each test varied between platforms, complicating the comparison of statistics between them.

In order to simplify the comparison, a mean value for each statistic was calculated from all of the recorded samples pertaining to each study or series transferred. Instead of plotting these mean values against real time, they were plotted against the sequence in which the study or series that they related to were transferred. This approach was applied to the data collected for network and non-volatile storage I/O. In the case of the CPU utilisation data, the statistics were computed for the duration of each test.

No samples were recorded during the transfer of some series because they took less than the time interval between *sysstat* samples to complete. Statistic values for these series were interpolated, using linear interpolation, from the statistics for the series transferred immediately before and after.

4.3 Results and Discussion

The results obtained are presented and discussed in this section. First, the selection of the image data used is discussed, followed by a presentation and examination of the measured transfer rates. The recorded system activity data are given and analysed. Consideration is also given to the experimental set-up, before, finally, the results are discussed in the context of the literature and ART.

4.3.1 Data Selection

The data used for testing the transfer rates on each of the platforms were randomly selected subsets of image data from the TCIA PROSTATE-DIAGNOSIS collection. Two subsets were selected: a set of fifty studies and a set of fifty series. The specific studies and series selected are given in Appendix A.

The data used were image data acquired as part of the diagnostic pathway only of prostate cancer patients. They are, therefore, not specifically representative of the imaging data that would typically be acquired for patients receiving radiotherapy. There are some features, however, that recommend them as a model of the image data for patients receiving radiotherapy for genitourinary cancer.

There would be a clear similarity with the image data acquired to diagnose patients being treated with radiotherapy for genitourinary cancer given that it is the same anatomy that is being examined.

There are multiple series per study, mimicking the temporally diverse image acquisition that is characteristic of modern IGRT. This is in contrast to the dataset created to model a clinical workload in [119], where it appears that most of the studies in that

dataset consist of a single series. This is likely to be indicative of the fact that the work in [119] was not targeted specifically at radiotherapy, but at the medical imaging field in general.

The image data utilised in this work contains only MR data, whereas current clinical radiotherapy typically uses a substantial amount of CT image data. However, in terms of series and image sizes, modern MR and CT data are analogous. Moreover, there is growing interest in making greater use of MR imaging within the radiotherapy treatment pathway.

The size of the dataset employed in this study was relatively small for modelling the workload of a typical radiotherapy institution. The entire dataset contained imaging data for 92 patients, which would be approximately equivalent to one day's workload for two treatment machines.

The atypically small amount of data contained in the PACS model is likely to have enabled the database to respond faster than if it had contained more data. This could indicate that the transfer rates measured here may be optimistic compared to what would be expected in the clinical setting. On the other hand, however, the hardware used to model the PACS here was much lower in performance than the dedicated server-grade equipment typically used in a modern radiotherapy institution. Indeed, the HDD of the PACS model used here would have been too small to have contained even 5% of the 2TB reference workload proposed in [119]. This being the case, it is possible that the transfer rates observed in the work presented here are pessimistic compared to what would be expected in the clinical setting.

Ideally, the tests described here would have been performed using a clinical PACS complete with clinical workload, however, gaining access to such a set-up for research purposes is challenging.

To ensure the results obtained using the sampled data could be reasonably generalised to the whole collection, the distributions of the study and series sizes in the collection and in each of the samples were compared.

Study Sizes

Figure 4.5 shows the histograms of study sizes for the PROSTATE-DIAGNOSIS collection and the randomly selected subset of studies used in the study-wise tests here.

The mean study size for the PROSTATE-DIAGNOSIS collection and the randomly chosen sample are shown in Table 4.3, along with the range and average number and size of images in each study.

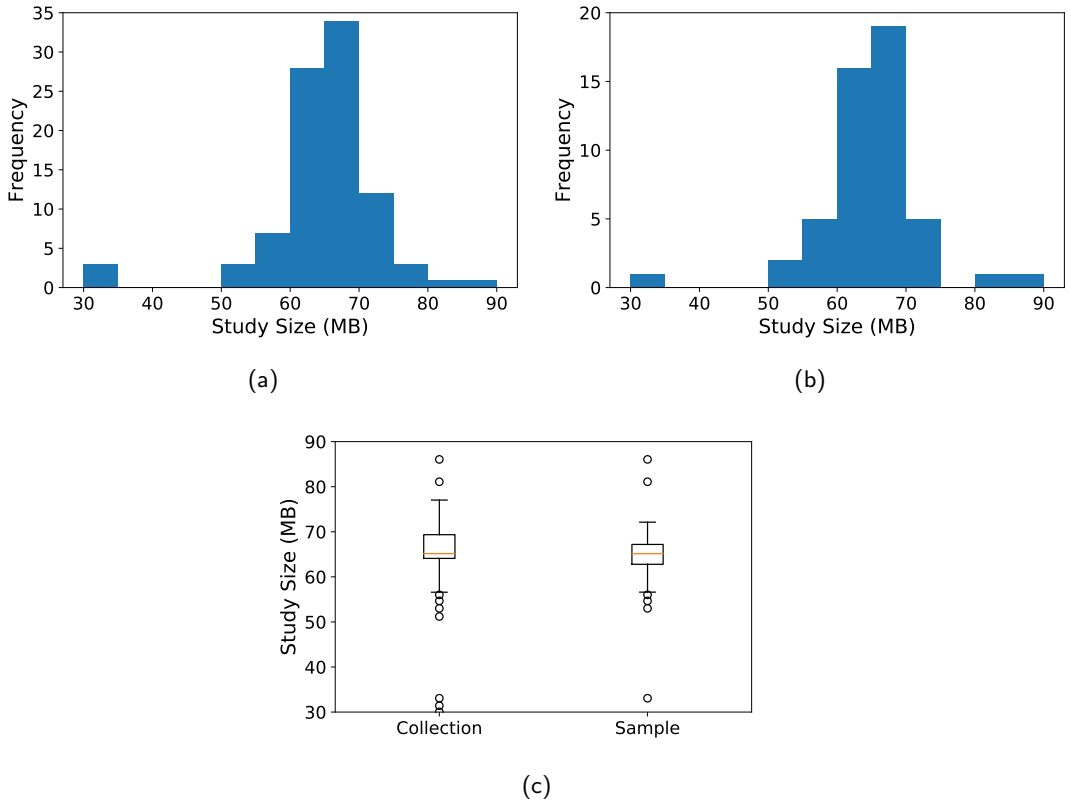


Figure 4.5: Study size histograms for (a) the PROSTATE-DIAGNOSIS collection, (b) the randomly selected sample, and (c) the PROSTATE-DIAGNOSIS collection and randomly selected sample as boxplots

The similarity in shape of the study size histograms for the PROSTATE-DIAGNOSIS data and the randomly selected sample shown in Figure 4.5(a) and (b) indicates that the sizes of the studies in the randomly selected sample closely mimic those in the collection as a whole. This is further reinforced by the similarity of the boxplots shown

Table 4.3: Study size statistics for PROSTATE-DIAGNOSIS collection and randomly selected sample

	Average Study	**Images per Study**			**Image Size (MB)**		
	Size (MB)	**Min.**	**Max.**	**Mean**	**Min.**	**Max.**	**Mean**
Collection	64.91	146	468	353.27	0.12	0.56	0.18
Sample	64.66	164	468	352.08	0.12	0.56	0.18

in Figure 4.5(c) where the collection and sample have similar medians and ranges of study sizes. This would suggest that it is reasonable to generalise the results obtained using the sample to what would likely be obtained if testing the whole collection.

The difference between the dataset in [119] and the dataset used here can be seen in the average study sizes, shown in Table 4.3 and Table 4.1, on page 69. These show that the study sizes in the dataset in [119] tend to be smaller in size and contain fewer images than the studies in the dataset employed in this work. In fact, the studies used in [119] are closer in size to a series in the dataset used here. Therefore, when comparing the transfer rates between the two pieces of work, it would be fairer to compare the rates reported in [119] with those obtained using series-level data in this thesis.

Series Sizes

The histograms of series sizes in the PROSTATE-DIAGNOSIS collection and the randomly selected sample of series used to perform the series-wise tests are shown in Figure 4.6.

Table 4.4 shows the range and mean number of images per series in the PROSTATE-DIAGNOSIS collection and the randomly selected sample, as well as average size of the images and the series.

Table 4.4: Series size statistics for PROSTATE-DIAGNOSIS collection and randomly selected sample

	Average Series	**Images per Series**			**Image Size (MB)**		
	Size (MB)	**Min.**	**Max.**	**Mean**	**Min.**	**Max.**	**Mean**
Collection	16.25	10	360	88.42	0.12	0.56	0.18
Sample	16.40	20	336	94.00	0.12	0.32	0.17

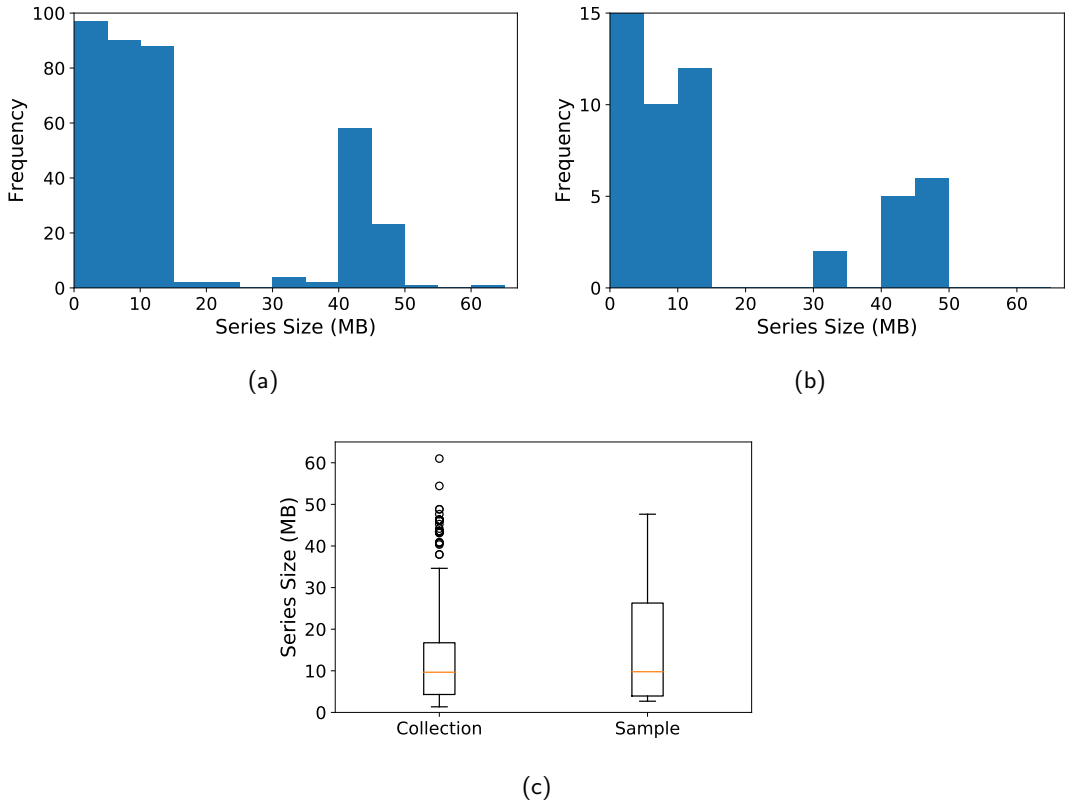


Figure 4.6: Series size histograms for (a) the PROSTATE-DIAGNOSIS collection, (b) the randomly selected sample, and (c) the PROSTATE-DIAGNOSIS collection and randomly selected sample as boxplots

The similarity in shape of the series size histograms for the PROSTATE-DIAGNOSIS data and the randomly selected sample shown in Figure 4.6(a) and (b) is not as clear as for the study sizes. The dissimilarities between the two distributions are even more obvious in the boxplots shown in Figure 4.6(c). From these it can be seen that the two distributions have similar first quartile and median values, however the third quartile value for the sample is around 10MB larger than that of the collection. This indicates that the randomly selected sample contains a greater proportion of larger series than the collection as a whole. This may affect how generalisable the results obtained using the sampled data were, if the results depended significantly on the size of the series being transferred.

4.3.2 Measured Transfer Rates

Inserting Studies in the PACS

The mean values for the transfer rate of each platform tested for inserting study-level data in the PACS model are shown in Table 4.5. These are the mean values of the transfer rates computed for each study that was transferred, based on the timestamps recorded by the C++ applications. The standard deviation of the measured transfer rates are also shown.

Table 4.5: Transfer rates when inserting studies in the PACS model

Platform	Mean Transfer Rate (MB/s)	Standard Deviation (MB/s)
ZedBoard	3.7128	0.6321
ZCU102	4.4207	0.8651
Desktop with HDD	4.5094	0.9352
Desktop with SD Card	4.3548	0.8085

A statistically significant ($p < .05$) difference in transfer rate was found between the ZedBoard and each of the other platforms, with the ZedBoard tending to have a lower rate, as can be seen in Table 4.5. No other statistically significant differences in transfer rate were found.

The real-terms effects of the measured transfer rates are illustrated in Table 4.6, where the time taken to transfer an image is tabulated. Mean and worst case times for transferring a study are also shown.

Mean image transfer rates were calculated using the mean transfer rates shown in Table 4.5 and the mean image size from the PROSTATE-DIAGNOSIS collection of 0.18MB.

The worst case image transfer rates were computed using the mean transfer rates and the largest image size from the PROSTATE-DIAGNOSIS collection of 0.56MB. A similar approach was used in calculating the mean and worst case times to transfer a study.

Table 4.6: Image transfer rates and time taken to transfer studies when inserting studies in the PACS model

	Images per Second		Study Transfer Time (s)	
	Mean	**Worst Case**	**Mean**	**Worst Case**
ZedBoard	20.63	6.63	17.48	23.16
ZCU102	25.56	7.89	14.68	19.45
Desktop with HDD	25.05	8.05	14.39	19.07
Desktop with SD Card	24.19	7.78	14.91	19.75

Table 4.6 shows that the difference found between the transfer rate for the ZedBoard and the other platforms would translate to the ZedBoard taking around 3–4 seconds longer to insert a study in the PACS model, compared to the other platforms. This was a 20–23% increase in the amount of time taken compared to the desktop with HDD, which represented the rate expected for current radiotherapy equipment.

Although this increase in data transfer time puts any hardware accelerator based on the ZedBoard platform at a disadvantage, it may still be deemed acceptable, if the ZedBoard were shown to sufficiently reduce the time taken to perform the image processing tasks compared to the desktop.

Inserting Series in the PACS

The mean values for the transfer rate of each platform tested for inserting series-level data in the PACS model are shown in Table 4.7.

Table 4.7: Transfer rates when inserting series in the PACS model

Platform	**Mean Transfer Rate (MB/s)**	**Standard Deviation (MB/s)**
ZedBoard	4.6887	1.3279
ZCU102	5.2863	1.5452
Desktop with HDD	5.9710	2.1953
Desktop with SD Card	5.5823	1.4864

Statistically significant differences ($p < .05$) were found between the transfer rates for the ZedBoard and the two desktop-based platforms. The difference in transfer rate measured between the ZedBoard and ZCU102 was also close to being statistically significant.

The measured transfer rate on each platform was slightly higher than those measured for inserting studies into the PACS model, shown in Table 4.5.

The real-terms effects of the transfer rates in Table 4.7 are illustrated in Table 4.8.

Table 4.8: Image transfer rates and time taken to transfer series when inserting series in the PACS model

	Images per Second		**Series Transfer Time (s)**	
	Mean	**Worst Case**	**Mean**	**Worst Case**
ZedBoard	26.05	8.37	3.47	13.01
ZCU102	29.37	9.44	3.07	11.54
Desktop with HDD	33.17	10.66	2.72	10.22
Desktop with SD Card	31.01	9.97	2.91	10.93

Table 4.8 shows that the time taken for the ZedBoard to insert a series in the PACS model was less than 3 seconds longer than the desktop with HDD platform. This difference is shorter in real-time than that found between the two platforms for inserting a study, but is longer in relative terms, representing a 24–30% increase in the time taken to insert a series.

Retrieving Studies from the PACS

Table 4.9 shows the mean transfer rates for each of the platforms when retrieving study-level data from the PACS.

The differences in transfer rate between each of the platforms were found to be statistically significant ($p < .05$).

It can be seen from Table 4.9 that the desktop platform using the HDD had the highest transfer rate followed by the desktop using the SD card, the ZCU102 and then the ZedBoard.

Table 4.9: Transfer rates when retrieving studies from the PACS model

Platform	Mean Transfer Rate (MB/s)	Standard Deviation (MB/s)
ZedBoard	3.2345	0.3237
ZCU102	5.0946	0.6645
Desktop with HDD	8.5447	0.8308
Desktop with SD Card	6.0782	1.2691

Table 4.10 illustrates the effects of the measured transfer rates in real-terms. It shows an increase of around 36–45% in the amount of time taken to retrieve a study from the PACS model between the two desktop platforms. The ZCU102 platform would be expected to take around 5–7 seconds longer to transfer a study from the PACS model than the desktop with HDD, a difference of almost 70%. The difference in transfer rates found between the desktop with HDD and ZedBoard platforms would be expected to result in the ZedBoard taking between 12 and 17 seconds longer than the desktop, or around 1.6 times as long, to retrieve a study from the PACS model.

Table 4.10: Image transfer rates and time taken to transfer studies when retrieving studies from the PACS model

	Images per Second		Study Transfer Time (s)	
	Mean	Worst Case	Mean	Worst Case
ZedBoard	17.97	5.78	20.07	26.59
ZCU102	28.30	9.10	12.74	16.88
Desktop with HDD	47.47	15.26	7.60	10.06
Desktop with SD Card	33.77	10.85	10.68	14.15

Retrieving Series from the PACS

Table 4.11 shows the mean transfer rates for each of the platforms when retrieving series-level data from the PACS.

Table 4.11: Transfer rates when retrieving series from the PACS model

Platform	Mean Transfer Rate (MB/s)	Standard Deviation (MB/s)
ZedBoard	3.7844	1.1671
ZCU102	4.8788	1.2186
Desktop with HDD	8.1307	2.1942
Desktop with SD Card	7.1428	2.6985

Significant ($p < .05$) differences in transfer rate between each pair of platforms were found, with the exception of between the two desktop-based platforms. The difference in transfer rate between the two desktop-based platforms was nearly statistically significant. The transfer rates measured for each platform were similar to those measured for retrieving studies, as shown in Table 4.9, although the rate for the desktop with SD card was slightly higher.

Table 4.12 illustrates the effects of the measured transfer rates in real-terms.

Table 4.12: Image transfer rates and time taken to transfer series when retrieving series from the PACS model

	Images per Second		Series Transfer Time (s)	
	Mean	Worst Case	Mean	Worst Case
ZedBoard	21.02	6.76	4.29	16.12
ZCU102	27.10	8.71	3.33	12.50
Desktop with HDD	45.17	14.52	2.00	7.50
Desktop with SD Card	39.68	12.76	2.28	8.54

The relative difference in the time taken to retrieve a series from the PACS model between the ZCU102 and the desktop with HDD was similar to that observed for retrieving a study, at around 65%. This equated to a difference of between 1 and 5 seconds, approximately. The time taken to retrieve a series from the PACS model on

the ZedBoard was around 1.1 times as long as on the desktop with HDD, lower than the 1.6 times observed when retrieving studies. This equated to a difference of between 2 and 9.5 seconds between the two platforms.

4.3.3 Correlation between Transfer Rate and Size

The transfer rates measured here were tested for linear correlation with the size of the studies or series being transferred. The results of this showed that, with the exception of receiving series-level data from the PACS model, no statistically significant ($p < .05$) linear correlation was found.

In the cases where no statistically significant correlation was found, it is reasonable to assume that a similar transfer rate would be obtained in that case regardless of the size of the study or series being transferred. In the case of receiving series-level data from the PACS model, where a fairly weak, negative correlation was found, some consideration would need to be given to the size of the series being transferred when proposing a typical value for the transfer rate.

As was mentioned above, the randomly selected sample of series used to obtain the tranfer rate measurements contains a greater proportion of larger series than the collection as a whole. This, along with the negative correlation between series size and transfer rate, would suggest that the mean transfer rate measured for the sample may be lower than the mean transfer rate that would be expected for the whole collection. However, the strength of the correlation was also fairly weak, suggesting that the effect would not be great.

4.3.4 System Activity

CPU

The CPU utilisation data recorded by the *sysstat* software is presented for each of the platforms tested in each of the test cases in Figure 4.7. The figure shows the amount of CPU processing time spent with:

- the CPU idle and no outstanding I/O requests to be serviced by the DMAC;

- the CPU executing user level instructions, which was assumed to be the test applications;
- the CPU idle with outstanding I/O requests to be serviced by the DMAC.

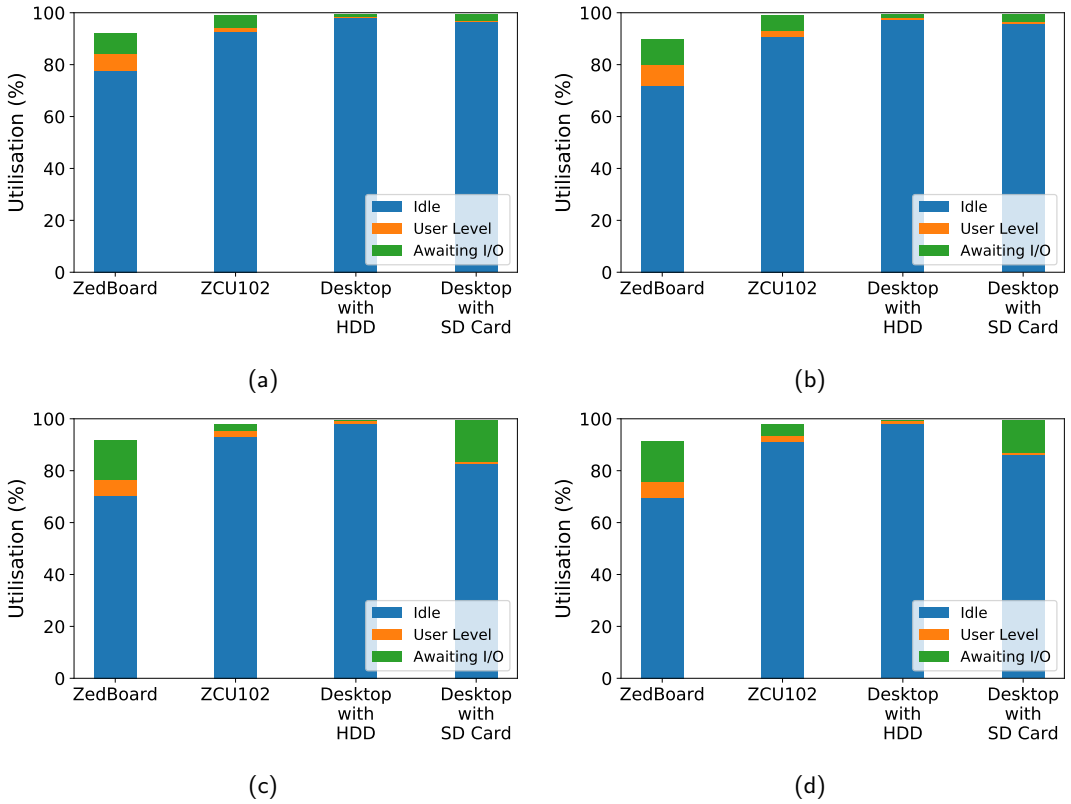


Figure 4.7: CPU utilisation for (a) inserting studies in the PACS model, (b) inserting series in the PACS model, (c) retrieving studies from the PACS model, and (d) retrieving series from the PACS model

It can be seen from Figure 4.7 that the vast majority of CPU time on each of the platforms, in each test was spent idle and only a small proportion was spent executing the test application. This suggests that there would be plenty of processing capacity available on each of these platforms to perform other tasks in conjunction with transmitting and receiving DICOM data.

In all of the tests, the ZedBoard spent a far greater proportion of time executing the application code than any other platform. The ZCU102 also spent a substantially greater proportion of time executing the application code than either of the desktop-based platforms.

The differences in application execution time between the SoC-based and desktop-based platforms are likely to be due to the relatively low performance of the CPUs on the SoCs compared to that used in the desktop. In addition to running at lower clock frequencies than the CPU used in the desktop, the CPUs in the SoCs are specifically designed for low power consumption rather than high processing performance. These factors are likely to have contributed to the lower transfer rates achieved with the SoC-based platforms, particularly the ZedBoard, which had the least powerful CPU of the platforms tested here.

Network

Graphs showing the proportion of the available network bandwidth that was used during the course of each test are given in Figure 4.8. A full-duplex connection was used in each case, and the graphs depict the utilistion on the channel with the greatest usage.

It can be seen from Figure 4.8 that network bandwidth utilisation is very low for each of the platforms tested across all of the tests. It is apparent from this that a lack of network bandwidth is not a limiting factor in determining the transfer rates reported here. It also shows that there is potential to achieve much higher transfer rates before there would be a need to increase network bandwidth.

On a real clinical LAN there is likely to be much more traffic due to the increased number of network nodes compared to the network model used here, which only had two. However, the network utilisation measured here suggests that the transfer rates reported would be robust against quite large amounts of additional network traffic.

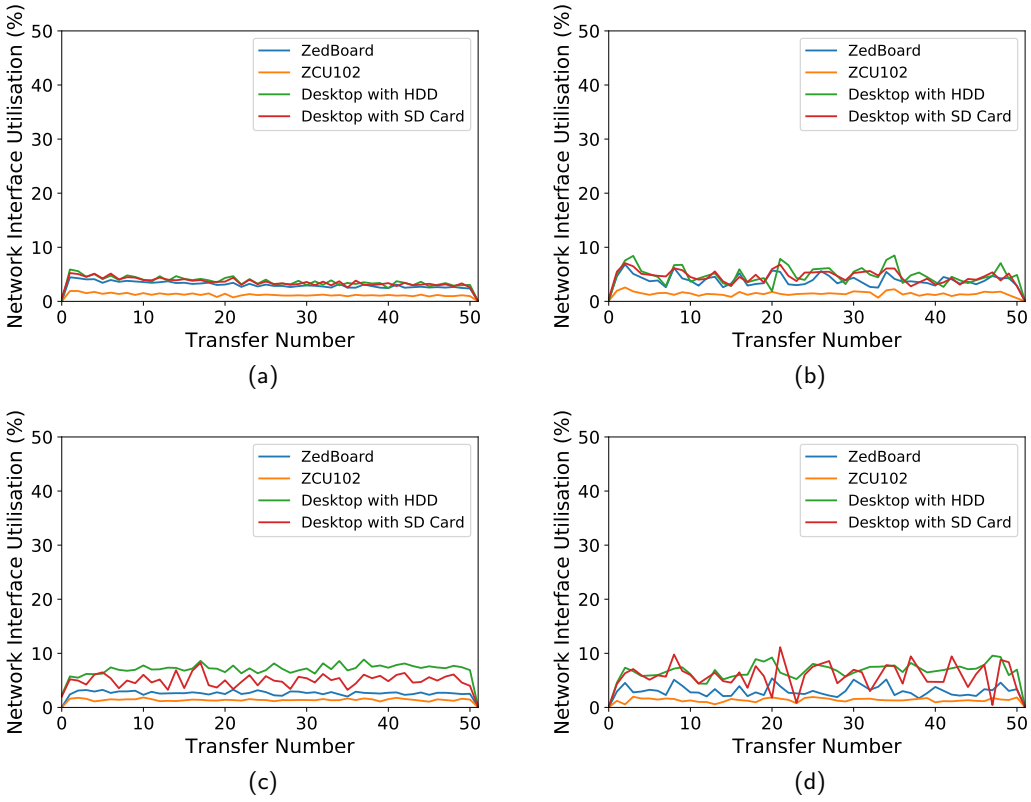


Figure 4.8: Network interface usage while (a) inserting studies in the PACS model, (b) inserting series in the PACS model, (c) retrieving studies from the PACS model, and (d) retrieving series from the PACS model

Non-volatile Storage

Mean values for non-volatile storage activity recorded by the *sysstat* software for each of the test cases are presented in Table 4.13.

Table 4.13 shows that the ZedBoard had the lowest average read speed from non-volatile storage in the case of inserting studies in the PACS. This is likely to have contributed to the ZedBoard spending a much greater proportion of time waiting for I/O requests to be serviced than any of the other platforms, as shown in Figure 4.7(a) on page 91. In turn, it is probable that this was a factor in the ZedBoard having a significantly slower DICOM transfer rate than the other platforms.

It can be seen from Table 4.13 that in the case of inserting series in the PACS, the mean read speed from non-volatile storage increased for each of the platforms compared to that when inserting studies in the PACS. It can also be seen from Tables 4.5 and 4.7,

Table 4.13: Mean values for non-volatile storage activity

	Zed-Board	**ZCU-102**	**Desktop with HDD**	**Desktop with SD Card**
Read speed inserting studies in the PACS (MB/s)	3.5685	4.2410	4.4798	4.3724
Read speed inserting series in the PACS (MB/s)	4.3379	4.8570	5.0116	5.0475
Write speed retrieving studies from the PACS (MB/s)	3.2280	4.7867	23.3767	6.5874
Write speed retrieving series from the PACS (MB/s)	4.3773	6.2695	23.9717	6.3669

on pages 85 and 86 respectively, that the increases in non-volatile storage read speed closely correlate with increases in the DICOM transfer rate. This suggests that the non-volatile storage read speeds were a significant factor in determining the DICOM transfer rates measured here. Furthermore, it suggests that the device drivers used by the operating system to read from the non-volatile storage were better suited to the data access pattern when inserting series, rather than studies, in the PACS. It may, therefore, be possible to gain improvements in transfer speed between non-volatile storage and memory by developing bespoke device drivers optimised for the data transfer patterns specific to a given application.

With the exception of the desktop with HDD platform, the mean write speeds to non-volatile storage, shown in Table 4.13, are similar to the mean transfer rates for retrieving data from the PACS, presented in Table 4.9 on page 88 and Table 4.11 on page 89. This clearly suggests that the factor limiting the transfer rate performance of each platform, except the desktop with HDD, was the write speed to non-volatile storage.

Table 4.13 shows that the desktop using the HDD clearly had the largest bandwidth for writing to non-volatile storage of the platforms tested. The mean write speed was an order of magnitude higher even than the same platform using the SD card. In the case of retrieving studies from the PACS, this resulted in a significantly lower DICOM

transfer rate for the desktop using the SD card. Thus, further demonstrating that the limiting factor on the transfer rate of the desktop using the SD card was the write speed to non-volatile storage.

4.3.5 Experimental Set-up

PACS Model

The PACS was modelled using a moderate performance laptop computer. In the clinical setting, server-grade equipment is more likely to be used for the PACS, such as that used in [119]. Compared to the server used in [119], the computer used as the PACS model here would not be expected to perform as well. The PACS model used here had two processing cores compared to the eight processing cores on the server, which also operated at a higher clock frequency. Moreover, the CPU on the PACS model used here was optimised for low-power performance, meaning it was designed to sacrifice some processing performance to gain energy efficiency compared to standard CPUs. The server used in [119] also had twelve times the amount of memory available to the PACS model used here and the nominal throughput to non-volatile storage was more than double.

Although the PACS model used here was likely to exhibit lower performance than the server-grade equipment likely to be found in the clinical setting, this must be offset against the fact that it was also used with an atypically small workload. Three orders of magnitude less data than was contained in the full dataset, and an order of magnitude less than the MR and CT component, used in [119].

Network Model

The experimental setup used in the test scenario from [119] that has been discussed here, connected the client and server machines by five parallel Gigabit Ethernet connections, giving five times the available network bandwidth compared to the experimental setup used here. It is possible that this was due to the purpose of the work reported in [119] being to promote the author's database application by demonstrating its capabilities rather than to accurately model the realities of the networking infrastructure found

in most medical institutions. It would not have served this purpose as well had the reported performance of the database application been unduly restrained by limitations in the hardware used for the demonstration. The single Gigabit Ethernet connection used in the experimental setup presented here was more representative of what would currently be used in a typical radiotherapy institution [116].

In other respects, the network model used here was idealised; there were only two network nodes resulting in less network traffic than might be expected on a clinical network and the two nodes were essentially in the same geographical location, connected by a short cable. In these respects, this setup appears to have been similar to that used in [119], although the geographical separation of their network nodes was not reported.

The transfer rates published in [118], however, were based on transferring data between network nodes separated by tens or hundreds of kilometres. These distances were based on the typical geographical separations that might be found in the grid computing use-case they were investigating. Most medical institutions tend to use their PACS on a LAN, making the test setup employed here more appropriate than the approach used in [118].

In the tests presented here, image data was transmitted uncompressed and unencrypted. Encryption substantially increases the amount of data that needs to be transmitted per image [118] and would seem an unnecessary overhead when data is being transmitted over a secure LAN, as is typically the case. The times reported here for transferring image, series and study data would need to be increased if encryption was used. The compression rates that can be achieved using lossless compression tend to be modest [117] and the additional processing overhead for compressing and uncompressing the data can outweigh the time saved from transferring less data. Lossy compression on the other hand, can achieve much higher compression rates, but the use of lossy compression is controversial in the medical imaging field [117].

4.3.6 Comparison with Literature

All of the platforms in each of the cases tested here comfortably exceeded the minimum image transfer rate suggested in [116] of two full resolution images in less than two seconds. The discussion in [116] concerned medical imaging in general and there are some medical imaging modalities that produce much higher resolution images than the MR images used here. The minimum standard proposed in [116] should therefore not be regarded as a high standard to meet for MR images.

The peak bandwidth calculated in [117] of 0.5MB/s was also comfortably exceeded by all of the platforms in each of the test cases here. However, the work reported in [117] was conducted quite some time ago. There have been improvements in technology in the interim, as well as an increase in the use of medical imaging data. The relevance of the rate reported in [117] in the context of modern radiotherapy is therefore questionable.

Most of the platforms in the majority of test cases recorded transfer rates that met or exceeded those reported in [118] for transferring MR and CT image data using the plain DICOM protocol. Unlike in [118], there was no geographical separation between the AEs in the tests carried out here. This is likely to have contributed to the increased transfer rates achieved in some cases. It is also possible that some of the improvement in transfer performance was due to using higher performance hardware. However, this is difficult to be certain of as few details of the hardware used in [118] were reported.

The only platform that exhibited transfer rates lower than those reported in [118] was the ZedBoard, and this only in the case of retrieving studies from the PACS model. In this case, the mean transfer rate measured for the ZedBoard was 0.4MB/s lower. The difference in performance may be attributable to the use of a lower performance CPU optimised for energy efficiency on the ZedBoard, although, again, it is difficult to be confident of this as few details of the hardware used in [118] were reported.

The transfer rates reported in [119] and [120] greatly exceed the rates recorded for any of the platforms tested here, by two orders of magnitude. The specification of the hardware used in [119] and [120] was much higher than that used here, although a much greater image data workload was also used.

Some of the differences in performance may be attributable to the difference in hardware specification. In some respects, such as the network bandwidth, the hardware used in [119] and [120] may not be representative of the equipment found in a typical radiotherapy institution. The work in [119] and [120] can also be criticised, as can the work presented here, that only two network nodes and no other network traffic or competition for access to the PACS were modelled.

Parallelism was exploited in [119] and [120]. Multiple images were transferred in parallel and multiple client applications ran concurrently. By contrast, images were transferred serially and only one application instance ran at a time in the work presented here. This is likely to have contributed to some of the differences in transfer rates observed.

Two levels of parallelism were used in [119]: having multiple instances of the data accessing application running to retrieve or insert multiple studies in parallel; and retrieving or inserting multiple images from the same study in parallel.

The benefit of the former approach in the context of adaptive radiotherapy depends on how the image processing platform is used. If the image processing platform were serving only one treatment machine, this approach is unlikely to be of much practical benefit since it is probable that only two datasets would need accessed per patient: the patient's previous image data from the PACS and the most recent image data from the treatment machine. There would be ample opportunity to download the patient's previous image data during patient setup and the acquisition of the image data for the current treatment session. There is therefore little advantage to be gained from being able to access multiple datasets in parallel. If, however, the image processing platform were serving multiple treatment machines, it is feasible that the ability to download multiple studies in parallel would be beneficial to the average data transfer rate. This could be achieved by running multiple instances of the application software concurrently on the image processing platform.

The latter approach would likely be advantageous in achieving higher data transfer rates, if applied here. It would require the algorithm in the application created to send image data to the PACS, Algorithm 1, to be altered to create separate threads for the

number of images to be sent in parallel when sending C-STORE requests to the PACS and a mechanism to re-synchronise the threads as responses were received from the PACS. In the case of receiving data from the PACS, the Orthanc source code would need similar alterations to its C-STORE SCU functionality to enable it to send multiple images in parallel.

The CPU utilisation data collected here indicates that there is sufficient processing capacity on all of the platforms tested to support transferring multiple images in parallel. Likewise, there was found to be plenty of spare capacity in the network bandwidth to allow more data to be transferred in parallel. However, in a real clinical setting, where there will almost certainly be multiple nodes competing for access to the PACS and network, the capacity for transferring data faster may be more limited.

4.3.7 Results in the Context of ART

The key measurements, in terms of meeting the clinical timing requirements of ART, are the times taken to retrieve study- and series-level data from the PACS. These are recapitulated in Table 4.14.

Table 4.14: Mean study and series transfer times when retrieving data from the PACS model

	Study Transfer Time (s)	**Series Transfer Time (s)**
ZedBoard	20.07	4.29
ZCU102	12.74	3.33
Desktop with HDD	7.60	2.00
Desktop with SD Card	10.68	2.28

The time taken to retrieve study-level data is representative of the time needed to transfer the original treatment plan and pre-treatment imaging data to the hardware accelerator at the beginning of the fraction. The time taken to retrieve series-level data is representative of the time needed to transfer the images acquired during the fraction to the hardware accelerator. In both of these metrics, the SoC platforms were found to be significantly slower than the desktop computer.

For retrieving study-level data, Table 4.14 shows that the ZedBoard platform would take more than double the amount of time to transfer the data as the desktop computer. However, the transfer would still take less than 30s, and could easily be accommodated during the several minutes required for patient set up, as illustrated in Figure 2.13 on page 36. Thus, even with the reduced performance of the SoC platforms, this transfer is unlikely to impinge on the critical time period between the acquisition of the treatment time images and the start of treatment delivery.

The time taken to transfer the image series acquired during the treatment fraction will, however, affect this critical time period, as highlighted in Figure 2.13. Table 4.14 shows that the ZedBoard platform would, again, take more than twice as long as the desktop computer to transfer the data. Yet, in absolute terms, Table 4.12 on page 89 also exhibits that the transfer to the ZedBoard would likely take no more than around 16s to complete. In the context of the clinical requirement for the time between image acquisition and treatment delivery being no more than ten minutes [3, 4], this still represents a very small proportion of the available timeframe, and would not preclude the use of an SoC platform for accelerating ART algorithms.

4.4 Conclusion

This chapter has presented work demonstrating the ability to interface FPGA-based SoC platforms with existing radiotherapy PACS using the DICOM standard. The rates at which image data could be transferred between the FPGA-based SoC platforms and the PACS were measured. These rates were compared against those reported in the literature and those measured using a desktop computer with a similar specification to a typical workstation found in a modern radiotherapy institution. System activity on each of the platforms tested was recorded to identify the critical factors affecting the data transfer performance of the FPGA-based SoC platforms.

The transfer rates of the SoC platforms were found to meet or exceed those reported in the literature, except in the case of the ZedBoard retrieving study-level data from the PACS.

Compared to the desktop computer, the ZedBoard was found to transfer at a statistically significantly lower rate in each of the cases tested. The ZCU102 platform, on the other hand, was only found to transfer at a statistically significantly lower rate than the desktop computer when retrieving data from the PACS.

The ZedBoard was found to take up to 17s longer than the desktop to transfer a study and up to 8.5s longer for a series. The ZCU102 was found to take up to 7s longer than the desktop to transfer a study and up to 5s longer to transfer a series.

These differences in transfer times in the order of, at most, tens of seconds are unlikely to have a significant impact on the time-critical portion of an ART fraction lasting up to ten minutes. However, they do place SoC-based designs at a substantial disadvantage in terms of accelerating ART algorithms. The improvement in performance achieved with an SoC-based design must be sufficient to overcome the transfer deficit in order to justify its usage.

The key factor identified as limiting the transfer rates on the SoC platforms was the bandwidth to non-volatile storage. The performance of the SoC platforms would be significantly improved by removing the need to read and write image data from non-volatile storage. Any SoC platform intended for use in ART should, therefore, be designed or chosen with enough memory to cache all of the image data required for an ART fraction.

In the event that non-volatile storage were still necessary, the non-volatile memory technology and interface should be chosen to maximise the bandwidth to system memory. Device drivers for accessing non-volatile storage should also be optimised for the data access patterns prevalent in ART.

Another factor identified as contributing to the slower transfer rates observed on the SoC platforms was the processing capabilities of the CPUs on the SoC devices. They were substantially less than those of the dedicated CPU used in the desktop, particularly in the case of the ZedBoard. Ideally, the CPU on the SoC devices being applied to ART would be optimised for computational performance, including a much faster clock frequency than either of the SoC devices used here.

Chapter 5

Segmentation of Bony Anatomy from CT Scans of Bladder Cancer Patients

ART depends on establishing a correspondence between a patient's anatomy at treatment time and at the time the planning scan was acquired. For patients receiving external beam radiotherapy to the pelvis, such as bladder cancer patients, this can be especially challenging due to the variability in the size, position and shape of organs within the pelvic cavity, such as the bladder and bowel. These variations are in addition to the variations in patient set-up where the entire anatomy of the patient is rotated and translated relative to the planning scan.

The rigidity of most bony anatomy makes it well suited for establishing the rigid correspondence between the planning and treatment images. The first stage in establishing the correspondence between the bony anatomy in each image is to segment the bony anatomy. This chapter presents the results of applying hardware acceleration to such an algorithm. It is envisaged that this would form the first stage in an ART image processing pipeline. Subsequent stages would include a rigid registration based on the bony anatomy segmented in this first stage and non-rigid registration to establish a corresponence between the soft tissues in the planning and treatment images.

In this chapter, adaptations to the segmentation algorithm are proposed to make it more suitable for implementation in hardware. The segmentation quality and execution time of the hardware implementation is compared to the original algorithm in software. Strategies for improving the performance of the hardware implementation in terms of execution time are proposed and the results are discussed in the context of ART.

5.1 Segmentation Algorithm

The algorithm selected for further investigation is one described by Haas et al [5]. Specifically, it was aimed at speeding up the segmentation of pelvic CT scans in radiotherapy planning with a view to enabling dynamic ART, making it extremely relevant to the work presented here. In addition, the algorithm is fully automated and requires no *a priori* information, making it well suited for ART. The algorithm is, however, reliant on high quality images and was developed for use on CT volumes, rather than the typically lower quality, CBCT volumes. Furthermore, the authors subjected the algorithm to significant clinical validation against CT scans of 69 pelvic studies, and it was rated to be time-saving in around 69% of these cases compared to manual segmentation [5]. Although, in the majority of these cases the automatically generated segmentation required major corrections to make it clinically acceptable.

The entire algorithm presented by [5] was designed to identify the patient's gender and orientation and multiple anatomic structures within each CT scan, however it starts by identifying the patient body and then decomposing that into soft tissue, bone and air or lung equivalent tissues. It is these initial parts of the algorithm, referred to by the authors of [5] as the *pre-segmentation*, that are investigated for hardware acceleration here.

In addition to being used commercially and clinically validated, the pre-segmentation algorithm proposed in [5] was chosen due to its use of predominantly local pixel operations. These are operations where the value of a pixel in the output image is dependent only on a small number of pixel values in close proximity in the input image. A series of local pixel operations can be implemented in a processing architecture where the pixel

data is streamed through the processing stages with small amounts of the data being buffered in memory local to each processing stage. Such architectures are well-suited for implementation on FPGAs.

In the first instance, the pre-segmentation algorithm proposed in [5], hereafter referred to as the *Haas* algorithm, creates a mask of the patient's body, eliminating any patient supports or extraneous material. This mask is applied to the original image and the bony structures are sought within the masked image. A rough segmentation into the three tissue classes:

(i) soft tissue;

(ii) bone tissue;

(iii) air or lung equivalent tissue;

can be achieved using simple range thresholds, as the pixel intensity values of a CT scan are related to the Hounsfield Units (HU) value of the material imaged, and these ranges are well characterised for human tissues. The algorithm operates on each 2D slice of a 3D CT volume in turn. A schematic representation of the algorithm is shown in Figure 5.1.

A mask is a binary image, where pixels belong to either the foreground or background sets. When a mask is applied to an image, the intensities of the image pixels that correspond with foreground pixels in the mask are retained in the output image. Those that correspond with background pixels are given an arbitrary value, usually 0, in the output image.

To create the body mask, a threshold range is applied to the CT slice. That is, if $I(x, y)$ is the intensity value of the pixel at position (x, y) of the input image, then the binary image output, $B(x, y)$ is given by Equation 5.1, where $\mathcal{F}$ is the set of foreground pixels, $\mathcal{B}$ is the set of background pixels and t_1 and t_2 are the lower and upper limits of the threshold range, respectively. For creating the body mask, a threshold range of between -175 and 1250HU is used. Figure 5.2(b) illustrates the result of applying these thresholds to an example CT slice.

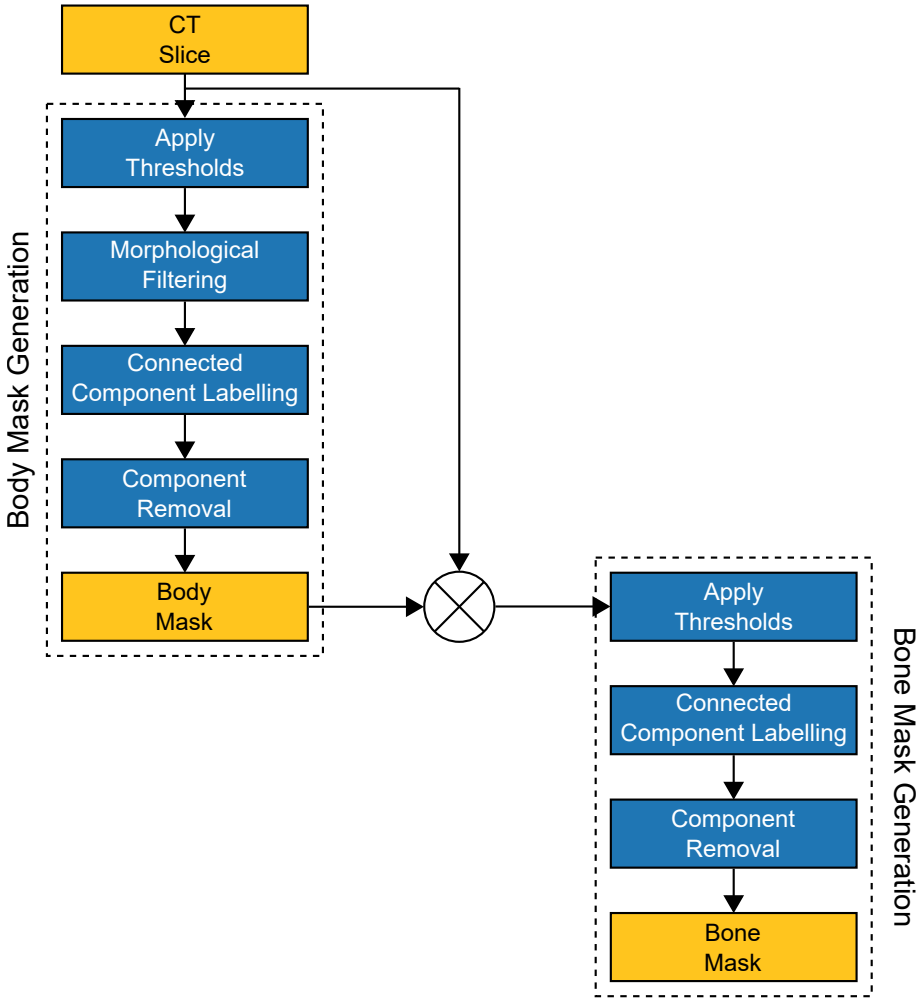


Figure 5.1: Pre-segmentation algorithm proposed by Haas et al. [5]

$$B(x,y) \in \begin{cases} \mathcal{F} & \text{for} \quad t_1 < I(x,y) < t_2 \\ \mathcal{B} & \text{otherwise} \end{cases} \tag{5.1}$$

Binary morphological operations are used to eliminate any patient supports from the body mask. Morphological operators assign the value of an output pixel based on the value of the corresponding pixel and a selected number of its neighbours in the input image. The local neighbourhood around a pixel used to determine the value of the

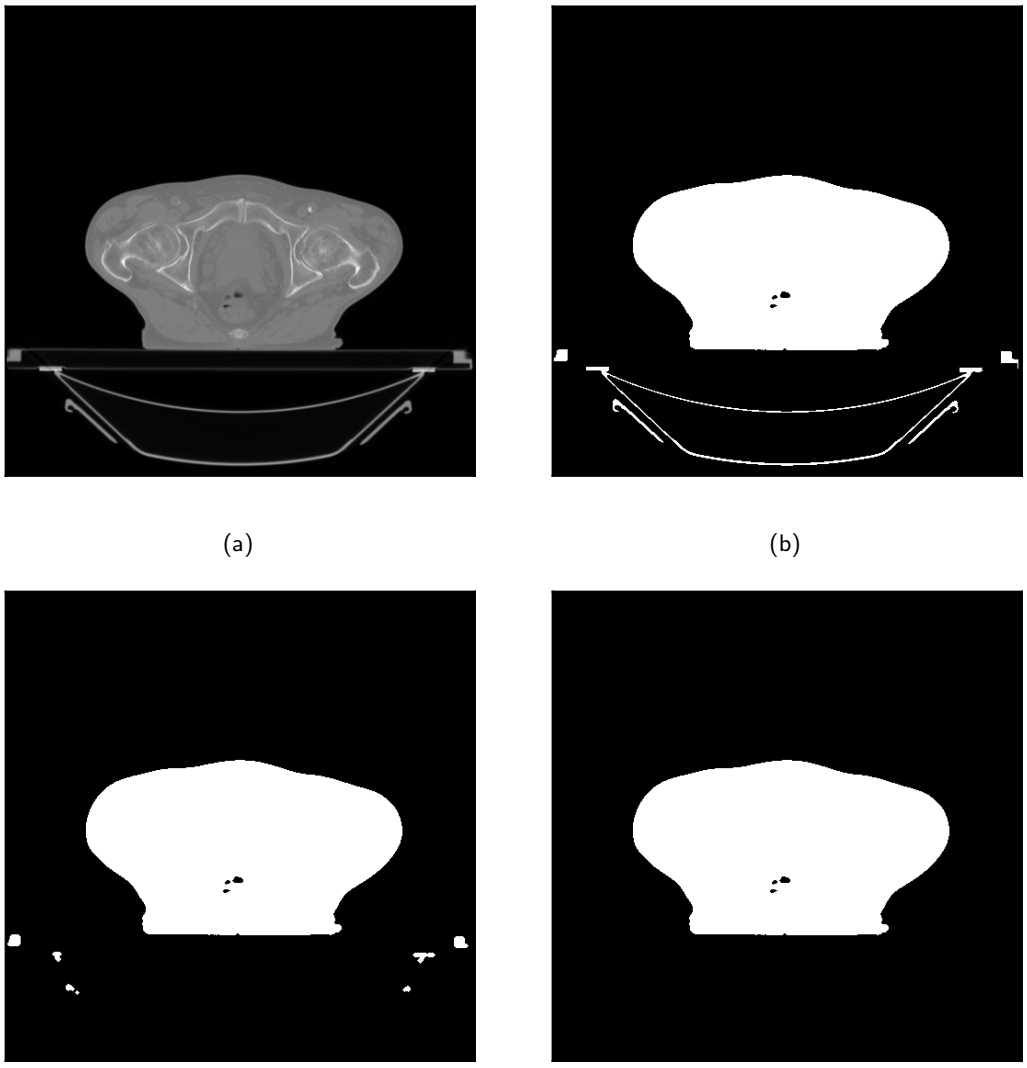


Figure 5.2: Illustration of the stages of the Haas algorithm to generate the body mask. (a) shows the CT slice used as input; (b) shows the output from the thresholding stage; (c) shows the output from the morphological filtering stage; and (d) shows the final body mask.

pixel in the output image is termed the structuring element. In the Haas algorithm, a cross-shaped structuring element, as shown in Figure 5.3, is used in each morphological operation.

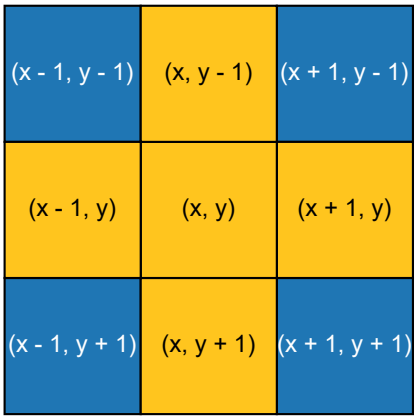


Figure 5.3: Pixels belonging to the 8-connected neighbourhood of the pixel (x, y) with their coordinates shown. The yellow pixels show the 4-connected neighbourhood of the pixel (x, y) and the structuring element used in the morphological filtering operations of the Haas algorithm.

Two iterations of a morphological opening are performed using this structuring element. The two iterations together consist of two morphological erosion operations followed by two morphological dilation operations. The output of an erosion operation, $E(x, y)$, on a pixel of an input image, $I(x, y)$, using the structuring element shown in Figure 5.3 is given by Equation 5.2. The output of a dilation operation, $D(x, y)$, on a pixel of an input image, $I(x, y)$, using the structuring element shown in Figure 5.3 is given by Equation 5.3. An example of the output produced from this stage is also shown in Figure 5.2(c).

$$E(x,y) \in \begin{cases} \mathcal{F} & \text{for} \quad \{I(x,y), I(x-1,y), I(x+1,y), \\ & \qquad I(x,y-1), I(x,y+1)\} \subseteq \mathcal{F} \\ \mathcal{B} & \text{otherwise} \end{cases} \tag{5.2}$$

$$D(x,y) \in \begin{cases} \mathcal{F} & \text{for} \quad \{I(x,y), I(x-1,y), I(x+1,y), \\ & \qquad I(x,y-1), I(x,y+1)\} \cap \mathcal{F} \neq \emptyset \\ \mathcal{B} & \text{otherwise} \end{cases} \tag{5.3}$$

Connected foreground pixels are collected into segments. The authors of [5] do not specify whether 4- or 8-connectivity is used.

In 4-connectivity, the foreground pixel (x, y) is connected to any other foreground pixels in the set $\{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\}$, as shown in Figure 5.3. In 8-connectivity, the foreground pixel (x, y) is additionally connected to foreground pixels in the set $\{(x-1, y-1), (x-1, y+1), (x+1, y-1), (x+1, y+1)\}$, also shown in Figure 5.3. Two foreground pixels, a_0 and a_n, belong to the same segment if there exists a set of pixels $\{a_1, \ldots, a_{n-1}\}$ such that a_i and a_{i-1} are connected for all $i \in [1, n]$.

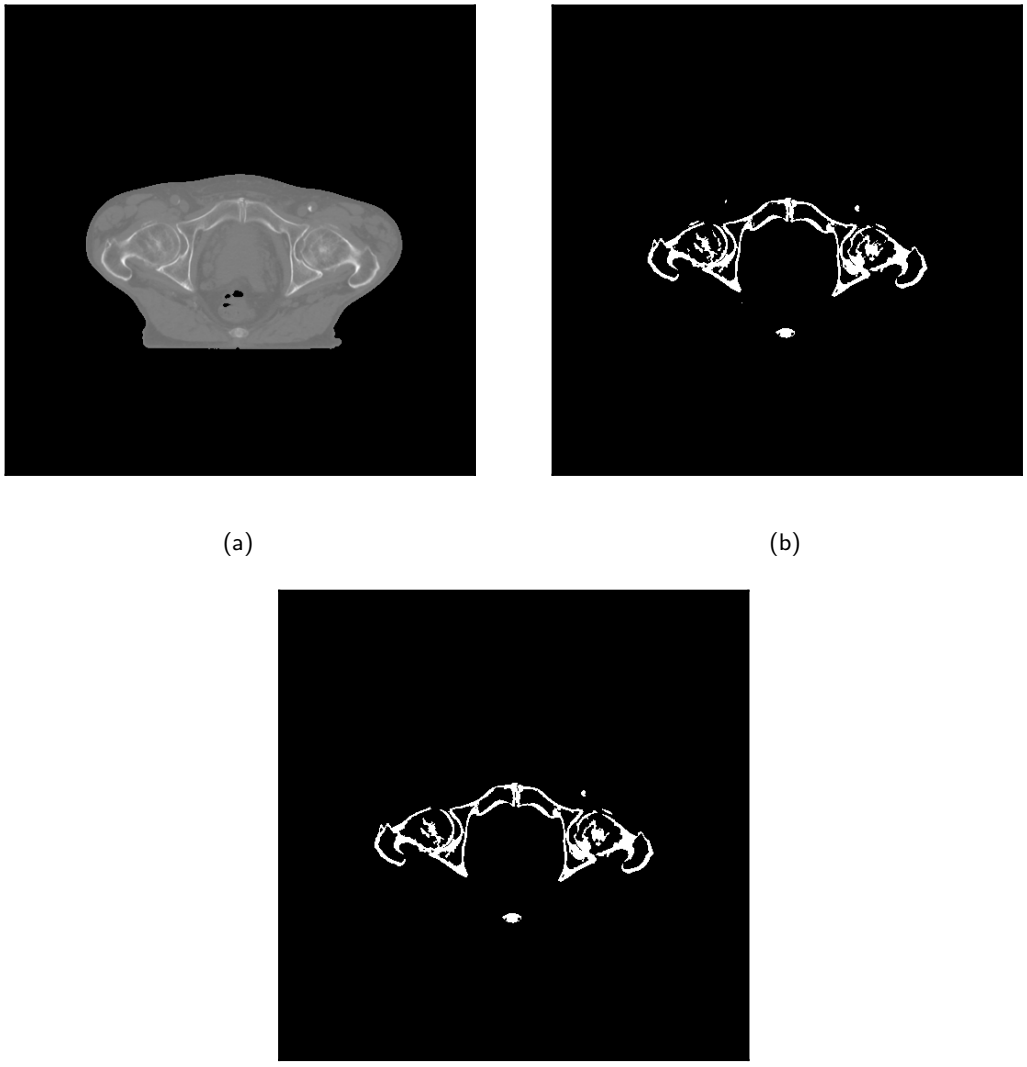
Any segments where the area of the pixels contained in them is less than 800mm^2 are discarded from the body mask. That is to say, if a segment, $\mathcal{S}$, has fewer than the requisite number of pixels to constitute an area of at least 800mm^2, any pixel in $\mathcal{S}$ is removed from the set of foreground pixels and becomes a member of the set of background pixels. This is also the case for segments that do not overlap with a 340×170mm rectangle centred in the image. An example of the body mask produced by this stage is shown in Figure 5.2(d).

The body mask, M, is applied to the original CT image slice, I, to form the masked image, N, as described in Equation 5.4. Figure 5.4(a) illustrates the result of applying the body mask shown in Figure 5.2(d) to the CT slice shown in Figure 5.2(a).

$$N(x, y) = \begin{cases} I(x, y) & \text{for} \quad M(x, y) \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Bone pixels are segmented from the masked image by applying a range threshold, as described in Equation 5.1, of between 145 and 1500HU. An example of the output produced from this thresholding operation is shown in Figure 5.4(b).

In addition to bone pixels, this also segments pixels belonging to calcifications, metal and image artifacts. To segment only those pixels that are most likely to genuinely represent bony antomy, segments with an area less than 25mm^2, or an aspect ratio less than $\frac{1}{6}$ or greater than 6, are removed from the segmentation. An example of the output produced from this operation is shown in Figure 5.4(c).



(a) (b)

(c)

Figure 5.4: Illustration of the stages of the Haas algorithm to generate the bone mask from the body segmentation. (a) shows the body segmentation from the original CT slice used as input; (b) shows the output from the thresholding stage; and (c) shows the output from removing segments that were unlikely to represent bone due to their size or aspect ratio.

Similarly, segments that are too close to the outline of the body are also removed, although the authors do not provide definitive criteria for what constitutes 'too close' [5]. An example of the complete segmentation produced by the Haas algorithm is shown in Figure 5.5. The boundary of the body mask produced by the Haas algorithm is indicated in the image by the outer-most green contour. The boundaries of the bone mask produced by the Haas algorithm are given by the yellow contours. Figure 5.5 also shows segmentations that are produced by stages of the algorithm proposed in [5] subsequent to the pre-segmentation stage that is of interest in this thesis. The additional segmentations denoted are the bladder, prostate gland, rectum and femoral heads.

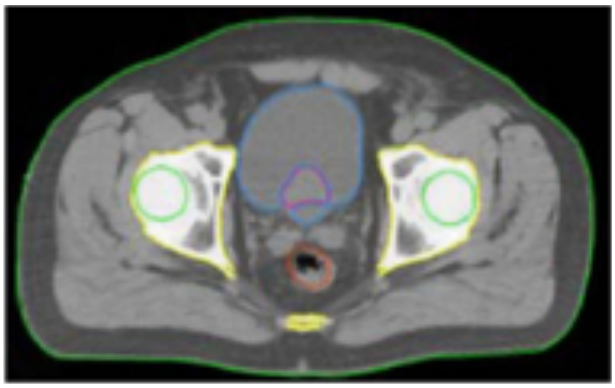


Figure 5.5: An example segmentation produced by the full algorithm proposed in [5]. The boundaries of the body and bone masks produced by the pre-segmentation portion of the algorithm can be seen in green and yellow, respectively. Segmentations of the bladder, prostate, rectum and femoral heads are also shown by the blue, pink, red and inner-most green contours, respectively. [5]

5.2 Haas Algorithm Implementations

This section provides details of the software and hardware implementations of the Haas algorithm employed in this thesis. The metrics used to assess the performance of the hardware implementation are described, as is the image data used to generate them.

5.2.1 Software Implementation

The Haas algorithm was implemented in software using the C++ language, based on the description given in [5], to act as a benchmark for both segmentation quality and execution time. The source code for this implementation can be found in Appendix B. The OpenCV software library [128] was used to implement the application.

The algorithm was implemented as described in Section 5.1 with the following caveats. 4-connectivity was assumed when establishing segments for both body and bone segmentation. The upper threshold for segmenting bone was changed to 1500HU, as using the thresholds proposed by Haas et al. excluded some areas of high density bone from the segmentation in the image data used here. The step of removing bony anatomy segments located too close to the body outline was omitted as the criteria for determing which segments were too close was unclear from [5].

Modifications for Hardware Implementation

The Haas algorithm relies upon the establishment of segments of connected foreground pixels when excluding segments both from the body segmentation and the bone segmentation. The establishment of these segments requires connected component labelling, where each foreground pixel is assigned exactly one label identifying the segment to which it belongs, and all foreground pixels with the same label are connected.

The connected component labelling algorithm is not well suited for acceleration on an FPGA since it is an inherently global algorithm. That is, the value of any pixel in the labelled image cannot be definitively determined only from the values of the pixels in its immediate neighbourhood, but depends on the values of pixels in the entire image. Therefore, the entire image must be buffered in memory in order to assign the correct label to each pixel.

The image could be buffered in the memory elements present in the FPGA fabric, although there may not be enough memory capacity on more modest FPGA devices, depending on the size of the image. Alternatively, image data could be buffered in external memory, but this would incur a performance penalty for transferring the image data to and from external memory.

FPGA implementations of connected component analysis that do not require the entire image to be buffered have been proposed. Instead, they record features of each segment found and dynamically update these features, including combining them when two segments are found to be connected, as each pixel of the image is processed in raster-scan order. Since they do not apply the segment labels to the image there is no need to buffer or rescan the image data. This technique is not suitable for this application since, in this case, it is intended to remove the segments that do not meet the segmentation criteria. In such an event, it is necessary to be able to identify all of the pixels belonging to each segment. Therefore, a fully labelled image is required.

To better suit implementation on hardware, the Haas algorithm was adapted to avoid the need for connected component labelling. Instead, a region of interest (ROI) was selected of a rectangular area measuring approximately 340×170mm in the centre of each CT slice to coincide with the region that each segment in the body segmentation must overlap with in the Haas algorithm.

As a further consequence of not performing connected component labelling, the elimination of segments based on area, aspect ratio and proximity to the edge of the body contour, as described in [5], were omitted from the algorithm implemented in hardware.

5.2.2 Hardware Implementation

The algorithm implemented in hardware, hereafter referred to as the *Simplified Haas* algorithm, is shown in Figure 5.6 and consisted of two main stages. First, the ROI from the CT slice was segmented to create a mask representing the body of the patient. Secondly, the body mask was applied to the original ROI and a segmentation was performed on the masked image to identify pixels classified as bone.

The body mask was created by thresholding the ROI from the CT slice, as described in Equation 5.1, between -175 and 1250HU. The thresholded image was eroded twice using the cross-shaped structuring element shown in Figure 5.3 and as described in Equation 5.2. The result was then dilated twice using the same structuring element, as described in Equation 5.3, to remove parts of the couch from the body mask.

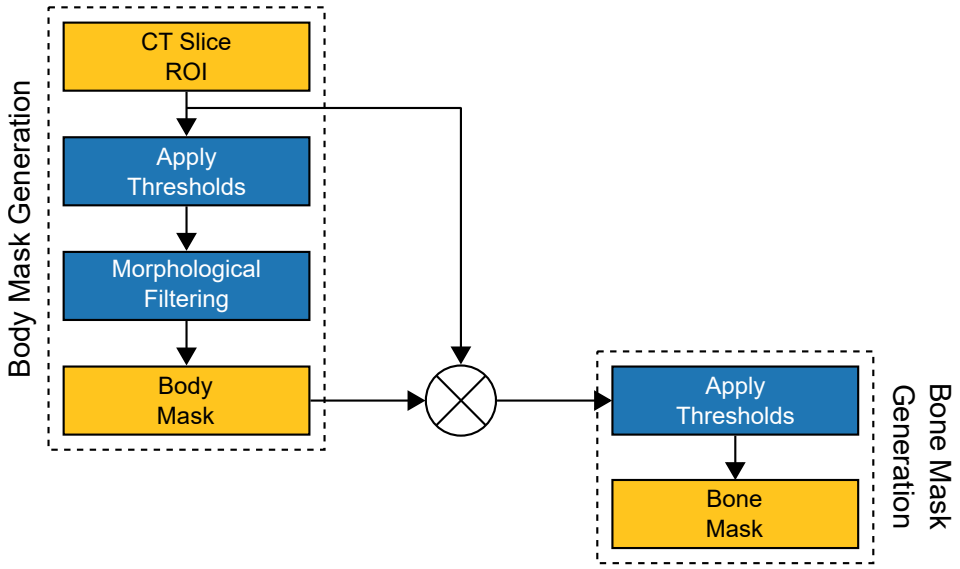


Figure 5.6: Segmentation algorithm implemented in hardware

The generated body mask was applied to the ROI from the CT slice and the resulting masked image was thresholded, as described in Equation 5.1, between 145 and 1500HU to create the bone segmentation.

Custom IP

IP cores were developed using the Vivado HLS tool to implement the Simplified Haas algorithm in the programmable logic of a Zynq Z7020 device. The IP cores were generated from C++ implementations of the IP core functionality with the aid of directives to help optimise the synthesised design. Directives supply additional information or instructions to the synthesis tool to assist it in generating the hardware implementation.

IP cores with four different functionalities were developed: thresholding, erosion, dilation and masking. The thresholding and morphological operator IP cores were based on version 2017.2 of the *xfopencv* library—a library of computer vision functions from which hardware can be readily generated using the SDx software [129]. The synthesis directives were changed to suit implementation using Vivado HLS rather than SDx, and the code was simplified to produce more application-specific and efficient designs.

Each of the IP cores were synthesised to be capable of processing images of up to 512×512 pixels in size; the typical size of a full CT slice. The actual size of the image to be processed was configured using an AXI4-Lite [130] interface prior to the core processing any data. Likewise, the thresholds to be used by the threshold IP core were configured via an AXI4-Lite interface.

All of the IP cores used AXI4-Stream [130] interfaces for transferring image data into and out of the IP core. The CT image pixel intensity values were represented as HU values offset by 1024, so the range of intensity values started from 24, representing air. AXI4-Stream interfaces require the width of the data bus to be byte-aligned, so, although many of the CT scans used here only used 12-bits to represent pixel intensities, 16-bit wide data buses were used on interfaces for transferring CT pixel intensity data.

On interfaces transferring binary image data, 8-bit wide data buses were used: the minimum permissible with the AXI4-Stream protocol. Pixels belonging to the set of background pixels were represented with the value 0, while pixels belonging to the set of foreground pixels were represented with the value 255.

Processing Pipeline

The IP cores were connected to form the processing pipeline shown in Figure 5.7 using Xilinx's Vivado tool.

Two AXI Direct Memory Access (DMA) engines were used to transfer image data between memory and the Zynq device's programmable logic. One of these engines was configured to use two channels: one to transfer the CT image ROI data to the start of the processing pipeline and the other to transfer the segmented bone mask from the output of the pipeline. The other AXI DMA engine was used to transfer the CT image ROI data to the masking stage of the processing pipeline to apply the body mask.

Each DMA engine used a dedicated AFI port to transfer the image data between the programmable logic and processing system. This arrangement was chosen to reduce competition between the DMA channels for access to memory and to maximise the bandwidth between memory and programmable logic.

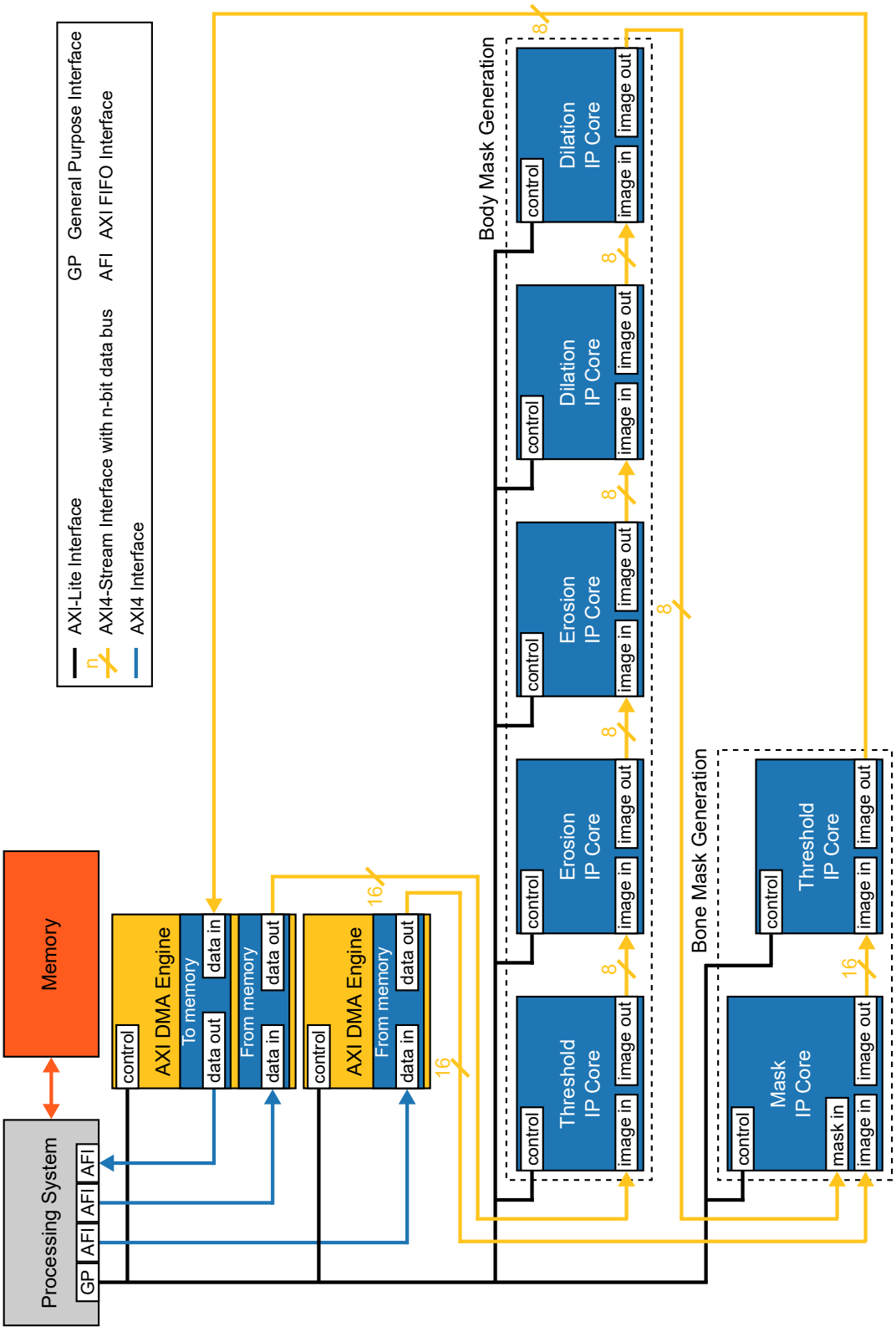


Figure 5.7: Hardware system implementing the Simplified Haas algorithm

A general purpose AXI port was shared between the custom IP cores and the DMA engines for configuration and control of the processing pipeline. This arrangement was selected as the configuration and control data had lower bandwidth requirements than the image data transfer, and sharing a port enabled fewer programmable logic resources to be used.

The transfer of each CT slice ROI to the processing pipeline was started as soon as the transfer of the previous slice's ROI had completed. In this fashion, parallelism could be achieved through processing multiple ROIs concurrently, each being at a different stage of the processing pipeline.

The processing pipeline was targeted for execution at a clock frequency of 100MHz. This frequency was chosen as it was believed to be a realistically attainable operating frequency for the circuit complexity and SoC device being targeted.

5.2.3 Implementation Platforms

Two platforms were chosen to compare the execution performance of the segmentation algorithms between software and hardware. The software was implemented on an Intel Core-i5 3230M CPU with 4GB of memory. An Avnet ZedBoard development board [131] with a Xilinx Zynq Z7020 device and 512MB of memory was used for the hardware implementation.

These platforms represent approximately contemporary technologies in terms of CPU and SoC design. Both are relatively modest within their respective product ranges with a focus on low-cost rather than performance. They therefore present a reasonable comparison between CPU and SoC execution performance.

The Simplified Haas algorithm was implemented on the ZedBoard. Both the Haas and Simplified Haas algorithms were implemented in software. In the software implementation, each CT slice was processed sequentially. The source code for the software implementation is given in Appendix B.

A performance comparison could also have been made against implementations on other processing architectures, such as GPUs or digital signal processors. However, the expertise required to produce non-naive implementations of the algorithm on these

architectures were not accessible during the course of the work presented here. The ubiquity of the CPU architecture does, however, provide an easily accessible common benchmark against which the performance of other processing architectures can be assessed.

5.2.4 Performance Metrics

In comparing the algorithms executed in software and hardware, two aspects were considered: the quality of the segmentation produced by the Simplified Haas algorithm relative to that produced by the Haas algorithm and the execution times of the algorithms.

Segmentation Quality

The results produced by the Simplified Haas algorithm were compared to those produced by the Haas algorithm to assess the quality of the segmentation.

A quantitative comparison was made between corresponding segmentations produced by the two algorithms using the Dice Similarity Coefficient (DSC) [132]. The DSC is a metric widely used in the assessment of medical image segmentation. The DSC between two segmentations, S, is calculated as shown in Equation 5.5, where A and B are the sets of foreground pixels from their respective segmented images. A DSC of greater than 0.7 is regarded as indicating good agreement between the two segmentations.

$$S = 2 \cdot \frac{|A \cap B|}{|A| + |B|} \qquad S \in [0, 1] \tag{5.5}$$

Time of Execution

To compare the execution performance of the algorithms, the times taken to process each CT volume and each CT slice were measured.

In the case of the software implementation, timestamps were recorded immediately before the function call to process each slice and immediately upon its return. The difference between these two timestamps was taken as an estimate of the time taken

to process the slice. The timestamps were recorded once per slice. Since each slice was processed sequentially, the time to process the volume was calculated as the sum of the times to process each slice in the volume.

In the case of the hardware implementation, a free-running timer on the Zynq device's processing sytem was read whenever the transfer of image data to the start of the processing pipeline was set up, and then again whenever the transfer of image data from the output of the processing pipeline completed. The difference between the two corresponding readings was taken as an estimate of the time taken to process the slice. This estimate was made once per slice.

The time to process the volume was calculated as the difference between the time when setting up the transfer of the first slice of the volume to the pipeline, and the time when the transfer of the final output slice of the volume from the pipeline completed.

The times calculated for the hardware platform include the overhead of transferring the image data between system memory and programmable logic.

5.2.5 Image Data

The image data used were CT and CBCT scans of bladder cancer patients (n=15) who recieved external beam radiotherapy at the Edinburgh Cancer Centre.

Every patient had either one or two CT scans that were obtained prior to the commencement of radiotherapy for the purpose of treatment planning. In addition, each patient had between five and ten CBCT scans that were obtained during the course of treatment as part of the routine workflow at the ECC.

In total, 18 CT scans and 114 CBCT scans were used. CT slices were 512×512 pixels in size and volumes ranged between 105 and 189 slices. CBCT slices were either 384×384 (n=66) or 512×512 (n=48) pixels in size with volumes ranging between 64 and 93 slices.

5.3 Results and Discussion

The results of the work are presented and discussed in this section. A comparison is made of the performance of the implemented algorithms, firstly in terms of segmentation quality and then in terms of execution time. The discussion of the execution time performance initially deals with the time taken to process an image volume, before examining the performance in more detail, by considering the time taken to process the slices the image volumes are composed of. The results of implementing the Simplified Haas algorithm in hardware are presented and discussed, as are strategies for improving the performance of the algorithms. Finally, the findings are reviewed as they apply to ART.

5.3.1 Segmentation Quality

The mean DSC comparing the segmentations obtained with the Haas and Simplified Haas algorithms for CBCT volumes was 0.97 and for CT volumes was 0.99. The distributions of DSC are shown in Figure 5.8. The differences observed between the DSC for CBCT and CT volumes were found to be statistically significant ($p < .05$).

The results of the comparison of the segmentation produced using the Haas and Simplified Haas algorithms shown in Figure 5.8 show good agreement between the segmentations produced by the two algorithms. The agreement between the segmentations on CT image volumes was stronger than for CBCT image volumes and this difference was statistically significant.

These results demonstrate that the Simplified Haas algorithm is a reasonable alternative to the Haas algorithm for the image data used here with minimum DSC values of 0.9786 and 0.8796 for CT and CBCT image volumes respectively.

The stronger agreement between segmentations for CT volumes than for CBCT volumes indicates that the Simplified Haas algorithm may be more susceptible to noise than the Haas algorithm.

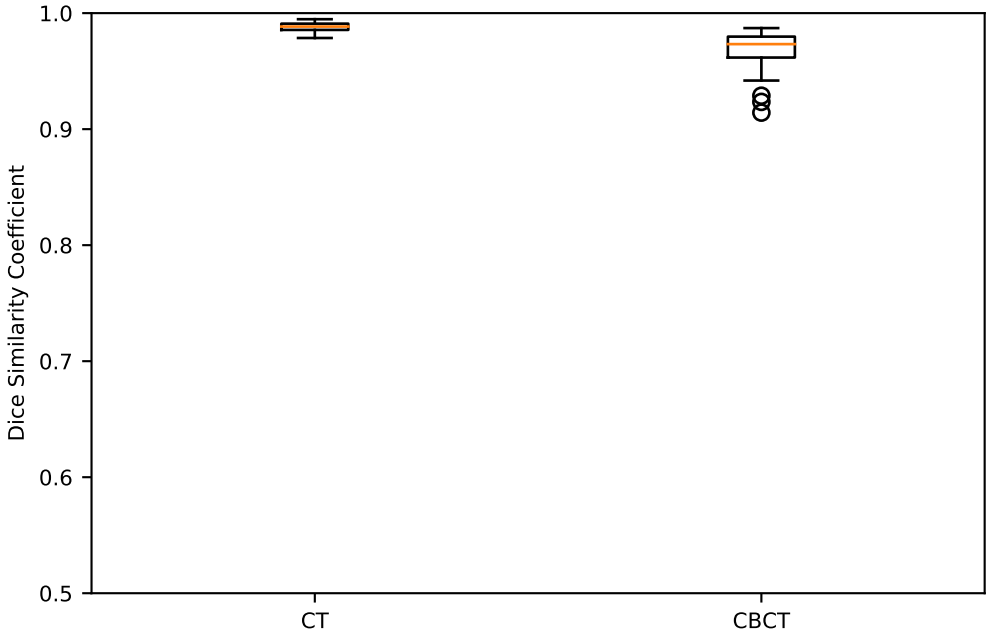
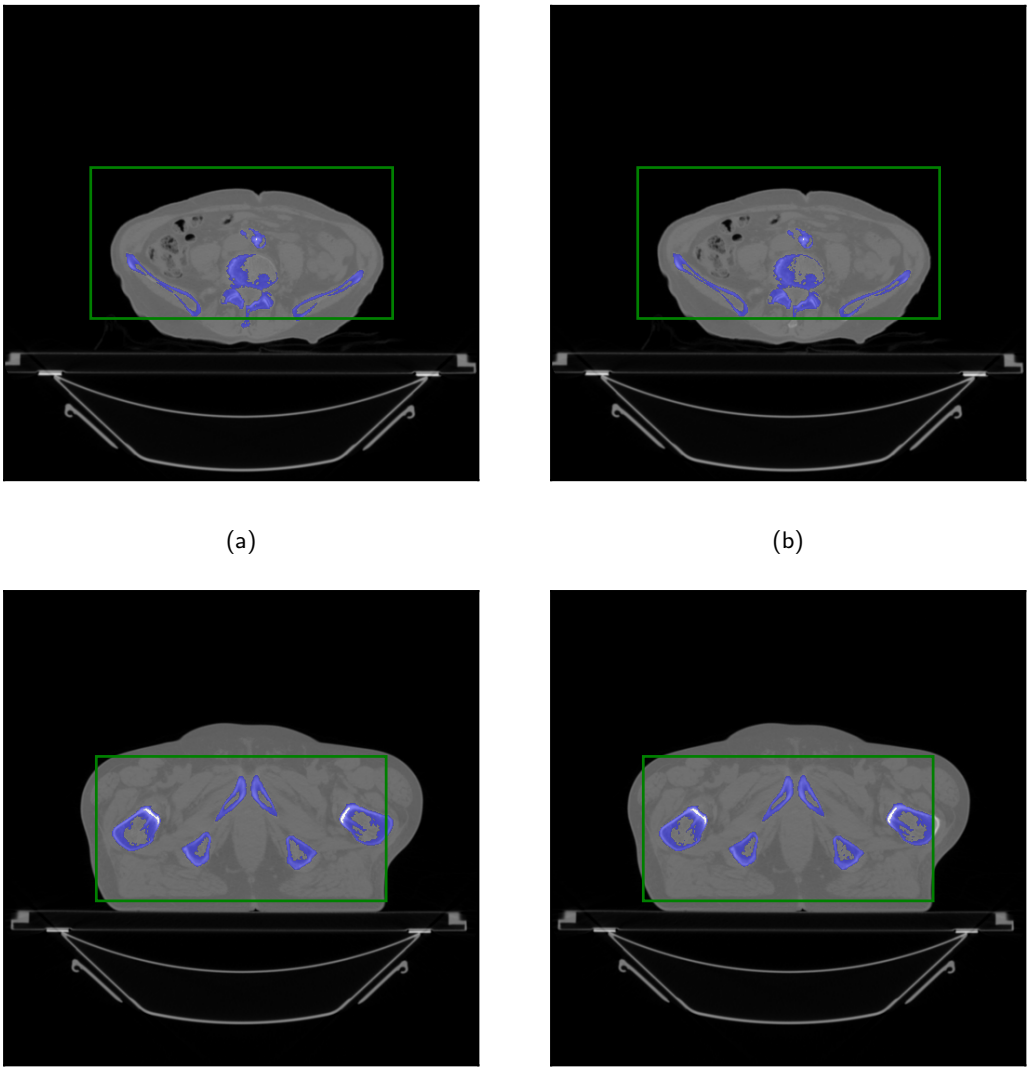


Figure 5.8: Dice Similarity Coefficients comparing segmentations produced by the Haas and Simplified Haas algorithms for CBCT and CT image volumes

Examples of the segmentations obtained using the Haas and Simplified Haas algorithms are shown in Figures 5.9 and 5.10. Figure 5.9 shows segmentations obtained from the CT image volumes with the highest and lowest DSC values. Figure 5.10 shows the equivalent segmentations obtained for the CBCT image volumes chosen using the same criteria.

Figures 5.9 and 5.10 show that, within the ROI used for the Simplified Haas algorithm, there was strong agreement between the Haas and Simplified Haas algorithms and this is bourne out in the high values of DSC obtained, as shown in Figure 5.8.

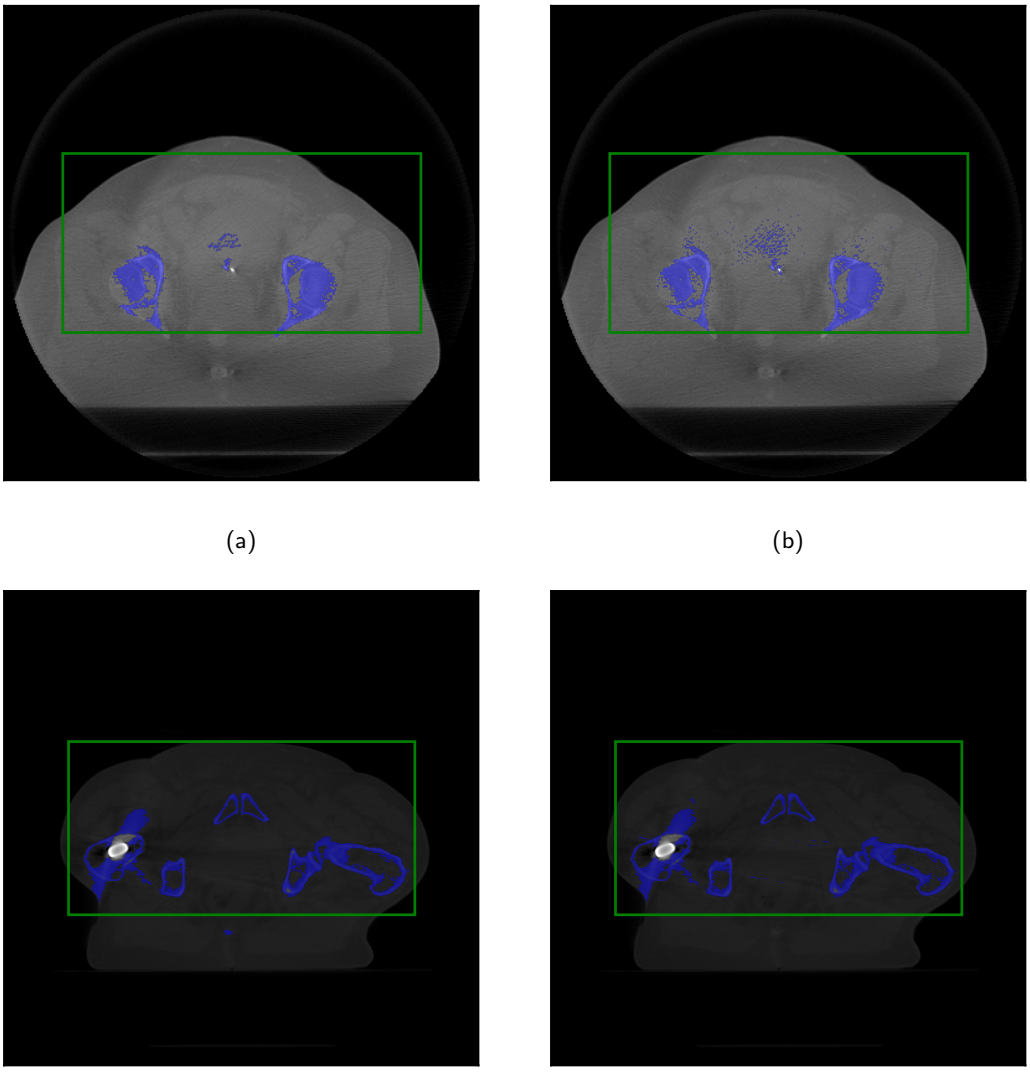
The CBCT scan shown in Figure 5.10(a) and (b) shows some discrepancies in the segmentations obtained using the two algorithms around the segmented regions. In these areas, the Simplified Haas algorithm appears to have been less robust in the presence of noise than the Haas algorithm and has segmented additional material.



(a) (b)

(c) (d)

Figure 5.9: Example segmentations obtained from CT volumes with segmented areas shown in blue. (a) and (b) are segmentations obtained using the Haas and Simplified Haas algorithms respectively from the CT volume with the lowest DSC. (c) and (d) are segmentations obtained using the Haas and Simplified Haas algorithms respectively from the CT volume with the highest DSC. The green box indicates the ROI processed by the Simplified Haas algorithm.



(a) (b)

(c) (d)

Figure 5.10: Example segmentations obtained from CBCT volumes with segmented areas shown in blue. (a) and (b) are segmentations obtained using the Haas and Simplified Haas algorithms respectively from the CBCT volume with the lowest DSC. (c) and (d) are segmentations obtained using the Haas and Simplified Haas algorithms respectively from the CBCT volume with the highest DSC. The green box indicates the ROI processed by the Simplified Haas algorithm.

Despite the increase in the upper threshold for bone segmentation applied in the Simplified Haas algorithm compared to the Haas algorithm, it is evident from Figure 5.9(c) and (d) and Figure 5.10(c) and (d) that both algorithms fail to segment areas of high density bone and hip prostheses. Furthermore, Figure 5.10(c) and (d) shows that artifacts in CBCT imaged caused by hip prostheses are spuriously segmented by both algorithms.

Anomalous Segmentations

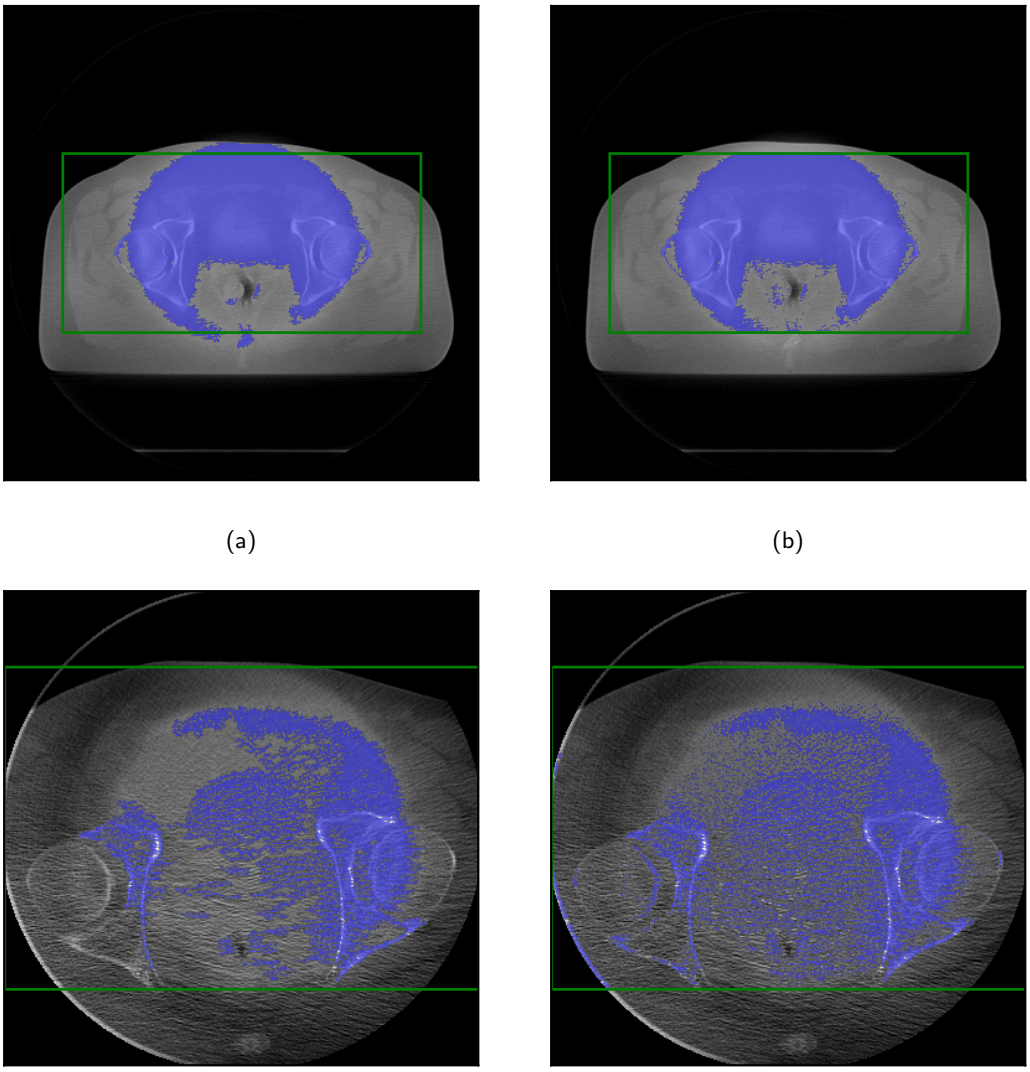
In two CBCT cases, shown in Figure 5.11, the segmentations produced by both algorithms were found to be grossly inaccurate in identifying the bony anatomy. However, the DSC obtained from comparing the segmentations produced by the two algorithms indicated good agreement, with DSC values of 0.88 and 0.99. This, again, suggests that the Simplified Haas algorithm did not perform significantly worse than the Haas algorithm.

The poor segmentation appears to have been as a result of the images having a high noise content. This exposes the limitations of both the Haas and Simplified Haas algorithms in handling noisy image data. However, this problem was only found in 2 out of the 114 CBCT volumes tested here and may be avoidable by selecting lower noise imaging protocols.

Excluding the two anomalous CBCT volumes shown in Figure 5.11, the DSC values for comparing the Haas and Simplified Haas algorithms ranged between 0.91 and 0.99, well above the 0.7 level proposed as indicating strong agreement. This suggests that the proposed modifications to the Haas algorithm produce similar segmentation results.

5.3.2 Execution Time per Volume

The results of measuring the time taken to process each image volume are summarised in the boxplot of Figure 5.12. The mean time taken taken to process CBCT and CT image volumes are shown in Table 5.1.



(a) (b)

(c) (d)

Figure 5.11: Example segmentations obtained from the CBCT volumes where the segmentations were found to be grossly inaccurate. An example segmentation from the first case is shown, obtained using (a) the Haas algorithm and (b) the Simplified Haas algorithm. An example segmentation from the second case is also shown, obtained using (c) the Haas algorithm and (d) the Simplified Haas algorithm. Segmented areas are shown in blue. The green box indicates the ROI processed by the Simplified Haas algorithm.

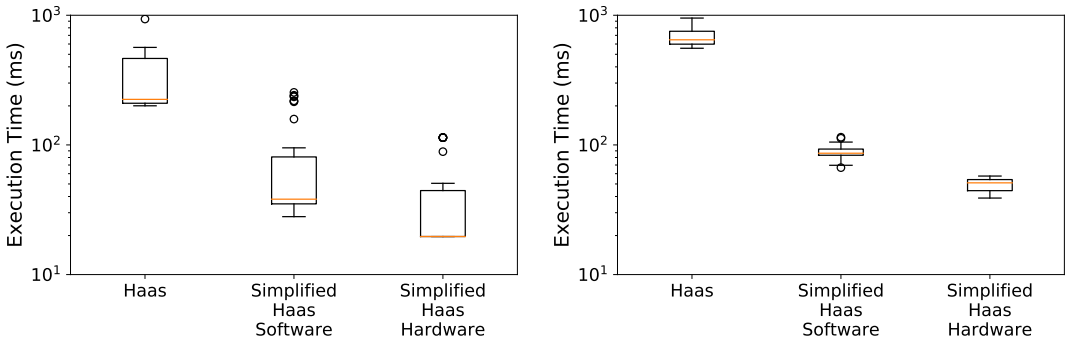


Figure 5.12: Measured times to process image volumes using the three algorithm implementations for (a) CBCT volumes and (b) CT volumes

Table 5.1: Mean measured times to process image volumes using the three algorithm implementations

Algorithm Implementation	**CBCT Volume (ms)**	**CT Volume (ms)**
Haas Software	331.3505	680.5748
Simplified Haas Software	64.8180	88.0092
Simplified Haas Hardware	34.2446	49.2951

It was found that the difference in time to process an image volume between the hardware implementation and the Haas algorithm in software was statistically significant ($p < .05$) for both CT and CBCT volumes. Likewise, the difference in time taken to process an image volume between the Haas and Simplified Haas algorithms in software was statistically significant ($p < .05$). However, the difference in time taken to process an image volume between the hardware and software implementations of the Simplified Haas algorithm was found to be statistically significant ($p < .05$) only in the case of CBCT volumes. The difference for CT volumes was not statistically significant.

Compared to the Haas algorithm executing in software, the hardware implementation of the Simplified Haas algorithm was considerably quicker. For processing CT scans, the hardware implementation was found to be 13.81 times faster on average than the software implementation, as shown in Table 5.1. Likewise, the hardware implementation was 9.68 times faster on average for CBCT image volumes, also shown in Table 5.1.

This increase in performance was achieved in spite of the CPU executing the software algorithm operating at a clock frequency 2.6 times greater than that used by the hardware implementation. These results also include the data transfer overhead inherent to SoC-based designs for moving data between system memory and the FPGA fabric.

Much of the performance improvement achieved with the hardware implementation compared to the Haas algorithm executing in software is likely to be attributable to the simplification of the segmentation algorithm. This can be seen from the substantial improvement obtained using the Simplified Haas algorithm compared to the Haas algorithm when both were executed in software, as shown in Table 5.1.

However, Table 5.1 shows that there was also a performance improvement for the hardware implementation compared to the software implementation of the Simplified Haas algorithm. For CBCT volumes, the hardware implementation was around 1.89 times faster on average than the software implementation and this difference was statistically significant. For CT volumes, on the other hand, the difference in average performance of 1.79 times was found not to be statistically significant. The lack of statistical significance for these results is likely due to the relatively small sample size of CT images (n=18) since the performance of the hardware implementation was faster than that of the Simplified Haas algorithm in software for each of the CT volumes tested, as can be seen from Figure 5.12.

5.3.3 Execution Time per Slice

The results of measuring the time taken to process each image slice are summarised in the boxplot of Figure 5.13. The mean time taken taken to process CBCT and CT image slices are shown in Table 5.2.

The differences in slice processing time between each of the implementations for both CBCT and CT image slices were found to be statistically significant ($p < .05$).

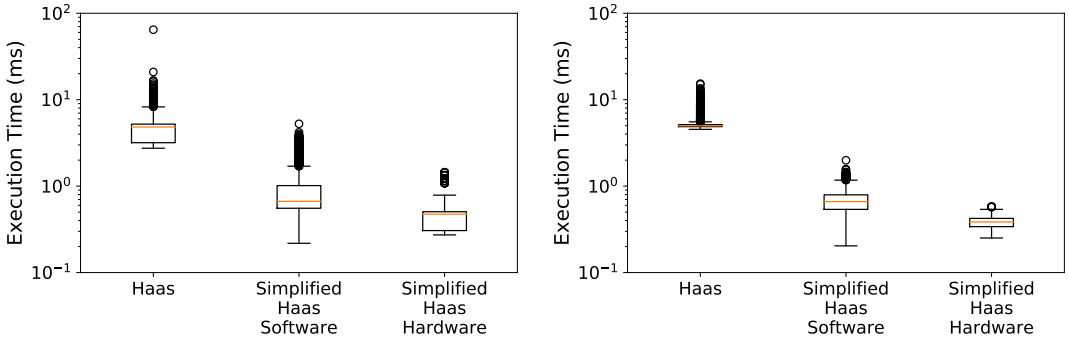


Figure 5.13: Measured times to process image slices using the three algorithm implementations for (a) CBCT volumes and (b) CT volumes

Table 5.2: Mean measured times to process image slices using the three algorithm implementations

Algorithm Implementation	**CBCT Volume (ms)**	**CT Volume (ms)**
Haas Software	4.4134	5.2374
Simplified Haas Software	0.8633	0.6773
Simplified Haas Hardware	0.4590	0.3822

Effect of Pipelining the Processing of Consecutive Image Slices

A comparison was made between the time measured to process each image volume in hardware and the sum of the times measured to process each slice of the corresponding image volumes. In all cases the sum of the times to process the individual image slices was greater than the time taken to process the volume as a whole. The mean difference across all tested image volumes was 0.2362ms.

The improvement in performance between the software implementation of the Simplified Haas algorithm and the hardware implementation is attributable to the increased parallelism of data processing operations achieved with the processing pipeline implemented in hardware. In the hardware implementation, the next stage of the pipeline is able to start processing the first pixels output from the previous stage of the pipeline while the previous stage is still processing the subsequent pixels. This is in contrast to the operation of the software, where the next processing stage only begins once all pixels have been processed by the previous stage.

Furthermore, once the first stage of the processing pipeline has finished processing the pixels for one slice, it becomes available to start processing the next slice in parallel with the previous slice being processed by subsequent stages of the pipeline. This is, again, in contrast to the software implementation, which only begins processing the next slice once the previous slice has been processed by each of the stages. This concept is illustrated in Figure 3.3 on page 51.

The increase in performance in hardware due to image slice pipelining is evident from comparing the difference in performance between the implementations on both a slice-by-slice and volume-by-volume basis. The results of comparing the performance of each of the implementations tested here on a slice-by-slice basis were broadly similar to those on a volume-by-volume basis, as can be seen from a comparison of Figures 5.12 and 5.13. However, the measured increase in performance of the hardware implementation compared to the Haas and Simplified Haas algorithms in software was not as large as on a volume-by-volume basis. Table 5.2 shows that the hardware implementation was 13.70 times faster for CT image volumes and 9.62 times faster for CBCT image volumes on a slice-by-slice basis than the Haas algorithm on average. This compares with average performance increases of 13.81 times for CT image volumes and 9.68 times for CBCT image volumes measured on a volume-by-volume basis, as shown in Table 5.1.

Similarly when comparing the hardware implementation with the Simplified Haas algorithm implemented in software, Table 5.2 shows that the hardware implementation was 1.77 times faster for CT image volumes and 1.88 times faster for CBCT images volumes on a slice-by-slice basis on average. This compares with average performance increases of 1.79 times for CT image volumes and 1.89 times for CBCT image volumes measured on a volume-by-volume basis, as shown in Table 5.1.

Although the improvement attributable to image slice pipelining is small relative to the time taken to process the image volume, it will be proportional to the number of slices in the volume, suggesting greater improvement in images of larger volumes or those imaged with higher resolution between slices. This is likely to explain the

difference in performance improvement between CT and CBCT image volumes, as shown in Table 5.1, as the CT volumes used here had a greater number of slices than the CBCT volumes.

5.3.4 Hardware Implementation

Resource Utilisation

Table 5.3 summarises the resource utilisation of the hardware design implemented on the SoC. The resource utilisation is given in absolute terms and as a percentage of the resources available on the Zynq Z7020 SoC used as the target platform for the work. Table 5.4 similarly shows the utilisation of interfaces between the processing system and programmable logic.

Table 5.3: Resource utilisation of hardware implementation

Resource	Absolute Utilisation	% Utilisation
LUT	6329	11.90
FF	8057	7.57
BRAM	7	5.00

Table 5.4: Processing system and programmable logic interconnection utilisation of hardware implementation

Interconnection	Absolute Utilisation	% Utilisation
General purpose	1	25
ACP	0	0
AFI	3	75

Increasing Parallelism

Despite the use of a modest SoC device, and the corresonding limitations of the FPGA fabric resources, it can be seen from Table 5.3 that the hardware design used very few of the available FPGA resources. There is scope, therefore, to improve the performance of the SoC implementation relative to the CPU by trading FPGA resources for greater parallelism.

As touched upon in Section 5.3.2, multi-threaded parallelism could be achieved in the hardware implementation by replicating the processing pipeline shown in Figure 5.7 in the FPGA fabric.

Given the resource utilisation shown in Table 5.3, the processing pipeline could be duplicated up to 8 times, allowing up to 8 slices of an image volume to be processed in parallel. However, the performance achieved using multiple pipelines is likely to be limited by the bandwidth between memory and FPGA fabric, as there are only 4 AFIs available on the device used here, and a single instance of the processing pipeline makes use of 3 of them, as shown in Table 5.4.

The AFIs were configured to use 32-bit wide data buses and a 100MHz clock rate to obtain the results presented here. In this configuration, and assuming perfect access to memory and no protocol overhead, the 4 AFIs combined provide a maximum throughput to memory of 1600MB/s on both the read and write channels [89]. Using the largest image size processed here of 332×512 pixels as the worst-case example, the performance of multiple pipelines would be limited by the bandwidth provided by the 4 AFIs once there were more than 2 pipeline instances.

The bandwidth of the AFIs could be doubled by making full use of the 64-bit wide data buses available on these interfaces [88, 89]. This would allow four 16-bit pixel values to be packed onto the bus per transfer and the throughput of the pipeline could be increased by processing multiple pixels per clock cycle. Additional FPGA resouces would be required to enable the pipeline to process multiple pixels per clock cycle but, as Table 5.3 shows, there would be ample resources available to achieve this. In

such a configuration, it would not be the bandwidth of the AFIs that would limit the number of parallel processing pipelines that could be supported, but the bandwidth of the memory controller.

The ZedBoard uses DDR3 memory with a clock rate of 533MHz and a 32-bit data bus [131]. Again, using the largest image size processed here as the worst-case example and assuming an 87% efficiency for the memory controller [89], the available bandwidth to memory would only allow 5 parallel pipelines to be fully serviced.

The bandwidth to memory could be increased by using memory with a higher clock rate or a wider data bus. The effect on the bandwidth between memory and the FPGA fabric of an increase in the memory controller bandwidth would be limited if the bandwidth of the AFIs was not also commensurately increased. This could be achieved by increasing the number of available AFIs or the width of the data buses.

In the design presented here, the input image is transferred from memory to the processing pipeline twice: first at the start of the pipeline and then when the body mask is applied to it, as shown in Figure 5.6. The memory bandwidth requirements could be greatly reduced by transferring the image from memory once and caching it in the FPGA fabric until it is required for the body mask application. This would half the bandwidth requirements of the read channels of the AFIs and would reduce the bandwidth requirements of the memory controller by 40%.

It would also greatly increase the number of memory resources used on the SoC device. Using the largest image size processed here as the worst-case example, caching a single slice would consume a further 83 BRAMs in addition to the 7 already used in the processing pipeline. The 140 BRAMs available on the SoC device used here would prevent the caching of more than 1 input image slice, therefore limiting the reduction in memory bandwidth usage and possible increase in the number of parallel processing pipelines that could be serviced. This could be remedied by using an SoC device with more FPGA memory resources, although that would also increase the cost of the system.

The relatively low usage of the available FPGA resources also provides the capacity to increase the level of parallel processing in another way. If the complexity of the implemented algorithm were increased by extending the current pipeline with additional processing stages, the amount of data being processed concurrently by the pipeline would also increase. The latency of the pipeline would increase by adding post-processing stages to it, however, if these additional stages were able to at least match the throughput of the current pipeline, the overall throughput of the design would not be reduced.

Reducing Resource Utilisation

In the design implemented here, AXI4-Stream interfaces were used between each of the stages of the processing pipeline, as shown in Figure 5.7. While the AXI4-Stream interfaces were required to connect to the AXI DMA engines, they were not required between the intermediate stages and their use restricted the data bus widths to being byte-aligned.

In reality, most of the interfaces between the intermediate stages transfer binary image pixel values, requiring data bus widths of 1-bit per pixel rather than the minimum 8-bit data bus width required by the AXI4-Stream protocol. Implementing an alternative interface protocol between these stages could therefore reduce the amount of FPGA resources used and provide further opportunities to trade off FPGA resources for faster processing.

Timing Constraints

The timing constraints for the system were met targeting a 100MHz clock with the worst case timing slacks shown in Table 5.5.

Increasing the Operating Frequency

The SoC hardware processing pipeline was synthesised and implemented using the default strategies in Xilinx Vivado v2017.2 with a target clock frequency of 100MHz. The worst case negative slack resulting from this was 0.021ns, as shown in Table 5.5.

Table 5.5: Worst case negative timing slacks of hardware implementation

Slack Type	Worst Negative Slack (ns)
Setup	1.297
Hold	0.021
Pulse Width	3.750

This implies that the performance of the hardware implementation could have been marginally improved, by around 0.21%, just by increasing the frequency of the clock used to drive the design.

It may have been possible to improve the worst case negative slack and achieve a design capable of operating at an even higher clock frequency by adopting a more aggressive implementation strategy. It is not anticipated, however, that any such improvement would be substantial compared to the results presented here.

5.3.5 Strategies to Improve Performance

A multi-threaded programming model for the software implementation, rather than the single-threaded approach used here, would be likely to improve the performance of the software implementation. To achieve the best performance would require partitioning the algorithm into separate threads capable of being executed largely independently of one another.

Assuming perfect partitioning of the algorithm, the performance of the software implementation would theoretically increase by a factor equal to the number of threads the CPU is capable of processing, in this case 4. Even with such an increase in the performance of the software implementation of the Haas algorithm, it would still fail to match that of the hardware implementation.

However, in practice, there are a number of factors that can reduce the performance of a multi-threaded implementation compared to the idealised maximum. The partitioning of an algorithm often incurs a processing overhead for managing and synchronising the multiple threads. In addition, some duplication of processing effort can occur where it is not possible to make the separate threads completely independent. For

example, if a multi-threaded approach were employed for the morphological operators used here by having each thread process a sub-region of the image, some of the pixels would need to be processed by more than one thread. This is because the output from a morphological operator depends on the values of a small neighbourhood of input pixel values. Therefore, pixels at the boundaries of the image sub-regions would need to be processed by the threads processing the sub-regions on either side of the boundary.

A multi-threaded programming model can also create competition for shared resources, such as memory, between threads. This can decrease the actual performance increase achieved compared to the theoretical maximum as one thread stalls waiting for access to a resource being used by another thread.

Moreover, multi-threaded parallelism could also be pursued for the hardware implementation through duplication of the processing pipeline in the FPGA fabric. Each pipeline could then process independent slices of the image volume in parallel, as discussed in Section 5.3.4.

5.3.6 Results in the Context of ART

The relative increase in performance of the SoC implementation over the Haas algorithm in software found here is substantial. However, in absolute terms, it reduced the execution time of the algorithm from hundreds of milliseconds to tens of milliseconds. In the context of an ART fraction lasting many minutes, the difference in performance of this particular algorithm is likely to be inconsequential as the unaccelerated algorithm took such little time to execute as a proportion of the fraction time. Nonetheless, any reduction in the amount of time to process an image captured at treatment time increases the validity of the image of the patient anatomy for adjusting the treatment, thus increasing the accuracy of the treatment.

In ART, the most time sensitive image processing is likely to be required for those images captured during a treatment fraction, which are most likely to be CBCT images with current technologies. The increase in performance for CBCT images was not as good as that measured for CT, but was still substantial.

Although the absolute improvement in performance for this particular algorithm is relatively small in the context of an ART treatment fraction, the algorithm itself is a small portion of the full image processing pipeline that is likely to be required for real-time ART. In fact, the algorithm accelerated here is only the pre-segmentation stage of a larger segmentation algorithm [5]. In comparing the planning image volume and the treatment fraction image volume there is also likely to be a registration stage in the full ART pipeline to establish a correspondence between the treatment plan and the patient anatomy at the time of treatment. If similar relative increases in performance could be achieved for each stage of the ART processing pipeline as have been found here, the overall reduction in time is likely to be consequential for real-time ART.

For example, if the improvement in execution time performance attained here for processing CBCT images were replicated for the ART algorithm reported in [33], the time to adapt the treatment plan would reduce from 26 minutes to under 3 minutes, and would then satisfy the ART timing requirements proposed in [3] and [4]. Considering the findings presented here in this context clearly demonstrates their significance, and highlights the potential for hardware acceleration to enable ART in the clinical setting.

Moreover, as was discussed in Section 5.3.4, increasing the depth of the processing pipeline by increasing the complexity of the algorithm can improve the comparative performance of the algorithm as it allows more processing to be done in parallel.

DICOM Feasibility

A single instance of the hardware processing pipeline for the Simplified Haas algorithm was shown in Table 5.2 to be capable of processing in excess of 2000 image slices per second. The results, presented in Chapter 4, page 85, from assessing the performance of the DICOM protocol on the same ZedBoard platform as was used here, illustrate that the rate of transfer of image data using the DICOM protocol over an Ethernet connection was two orders of magnitude less than that required to support one instance of the hardware processing pipeline. Such a low rate of image data transfer would

severely limit the performance of the system. Furthermore, even the extrordinarily high DICOM transfer rates reported in [119] and [120] would not be sufficient to allow the hardware processing pipeline to execute at its full rate.

These findings suggest that the DICOM protocol over an Ethernet connection is likely to limit the performance of an ART system and an alternative, faster, method of transferring image data should be used.

Segmentation Quality

The quality of the segmentation produced by the SoC implementation was assessed against the segmentations produced by the Haas algorithm. The authors of the Haas algorithm clinically validated the segmentations produced by their algorithm [5] and, given that the algorithm implemented on the SoC was a slight modification of their algorithm, it was reasonable to judge the quality of the segmentations produced by the work here against this standard. However, in the context of a full ART algorithm, the quality of the results produced by an initial segmentation algorithm will necessarily impact the effectiveness of subsequent processing stages and the ultimate result of the algorithm. Therefore, a better test of the quality of the segmentations produced would be to use them in a full implementation of an ART algorithm and to clinically assess the quality of the output from this. The difficulty in achieving this is that there is currently no consensus on a clinically acceptable ART algorithm.

5.4 Conclusions

This chapter has considered the implementation in hardware of a commercially developed and clinically validated segmentation algorithm, the Haas algorithm, for ART based on pelvic CT scans. Alterations to the Haas algorithm were proposed to make it more suitable for realisation in hardware, with this new algorithm being termed the Simplified Haas algorithm.

The segmentations produced by the Simplified Haas algorithm were assessed quantitatively against those produced by the Haas algorithm. The execution time of the Haas algorithm in software was compared with those of the Simplified Haas algorithm in software and hardware across a dataset of CT and CBCT image volumes of bladder cancer patients. Strategies to further improve the performance of the hardware implementation were discussed, as were the results obtained in the context of ART.

The Simplified Haas algorithm in both software and hardware was found to be much faster than the Haas algorithm while producing segmentations that strongly agreed with those produced using the Haas algorithm. This implies that the proposed modifications to the Haas algorithm greatly improve the segmentation performance in terms of execution time without greatly affecting the quality of the segmentation as assessed here.

The hardware implementation of the Simplified Haas algorithm outperformed the software implementation in terms of execution time. This was due to greater parallelism in the hardware implementation.

Although the improvement in the execution time of the segmentation algorithm was found to be relatively small compared to the time of an ART fraction time, it was demonstrated that, if similar improvements were made for each stage of a full ART algorithm, the resulting reduction in execution time was significant and the algorithm would consequently meet ART timing requirements.

The performance of the hardware implementation could be improved further by increasing the amount of concurrent processing. One approach would be to replicate the processing pipeline to process multiple images in parallel. This approach would be limited by the bandwidth available between system memory and the FPGA fabric and the memory resources available to cache image data in FPGA fabric on the SoC. Another approach would be to extend the processing pipeline with additional functionality, such as registration. The reduction in execution time achieved by the hardware implementation is also likely to be improved when using image datasets with a greater number of slices in the image volumes, as these would take greater advantage of the parallelism already realised.

The rate of transferring image data using the DICOM protocol over an Ethernet connection was found to be substantially slower than that needed to service a single instance of the Simplified Haas hardware processing pipeline. An alternative method of transferring image data to the pipeline would therefore be required to realise its full performance.

The results presented in this chapter demonstrate that FPGAs are good candidates to improve the execution time performance of ART algorithms that can be implemented as a data streaming pipeline. However, the bandwidth between system memory and the FPGA fabric is likely to limit the performance of FPGA implementations of ART algorithms. Some of this limitation could be mitigated by selecting FPGA devices with large amounts of memory resources in the fabric. This would enable more data to be cached in on-chip memory and reduce the number of accesses required to system memory. FPGA devices with faster or wider data buses to system memory would also help to alleviate this limitation. Selecting FPGA devices with large amounts of resources also increases the scope for implementing deeper processing pipelines or more processing pipeline instances. This increased parallelism would improve the execution time performance of the FPGA implementation for algorithms that can support it.

Another critical consideration when selecting an FPGA-based device for hardware acceleration in ART is the data rate that can be achieved when interfacing it to existing radiotherapy equipment. The processing pipeline will be starved of data if it cannot be transferred to the FPGA fast enough, and this will limit the performance of the hardware accelerator.

Although the Haas and Simplified Haas algorithms were shown to produce similar quality segmentations on the considered image data, both algorithms were also found to be limited in their ability to produce accurate segmentations in the presence of noise. This was found to be particularly the case for the Simplified Haas algorithm. Some radiotherapy techniques particularly well-suited to ART utilise image data that are more prone to noise than the image data used here, such as four-dimensional CT (4DCT). Consideration should therefore be given to the segmentation performance of the Simplified Haas algorithm on such data.

Chapter 6

Hardware Accelerated Segmentation of 4DCT Images

4DCT scans are intended to better visualise tissues regularly displaced by a periodic perturbation, such as breathing. During scanning, the capture of image data is correlated to the position of a surrogate marker, whose movement is, in turn, correlated to the periodic perturbation. By dividing the period of motion into a number of phases, the captured image data can be assigned to the phase of motion during which it was captured. Image volumes pertaining to each phase of motion can then be reconstructed using the image data captured only during the respective phase of the periodic motion.

Modern radiotherapy techniques such as SBRT are predicated on being able to deliver a precise and accurate treatment. For lesions in the thoracic and, to a lesser extent, pelvic cavities this requires consideration of the anatomical movement induced by breathing [1, 2]. 4DCT scans are therefore well-suited to the planning and quality assurance of modern treatment of lesions in these areas.

Compared to conventional CT scans, 4DCT scans produce much more image data [1, 133, 134]. With the equipment used here, 14 times as many image volumes were produced using 4DCT scanning techniques as would have been using conventional techniques. With such a copious amount of data to be processed, the rate of image processing is even more important in order to meet the timing requirements of ART.

In addition, the motion of the subject being scanned tends to increase the amount of noise present in 4DCT image volumes compared to conventional CT [1, 2, 135]. This further increases the challenge of processing the image volumes automatically.

This chapter focuses on accelerating the segmentation of 4DCT image data as the preliminary stage of a full ART algorithm for abdominal malignancies whose treatment is affected by respiratory motion. Acceleration of the segmentation algorithms was sought by implementing them in dedicated hardware. The image data investigated was of a quality assurance phantom designed to model the anatomical movement in patients caused by respiration.

Two approaches to the segmentation problem are investigated. Firstly, a simplistic approach based on generating optimal thresholds using Otsu's method [6]. Secondly, a continuation of the work presented in Chapter 5 using the algorithm based on that proposed by Haas *et al.* [5].

For both approaches, software and hardware implementations were developed and their performance compared in terms of execution time. In addition, the quality of the segmentation produced was assessed both quantitatively and qualitatively. Consideration is given to the extension of these algorithms for application to clinical 4DCT image data and the limitations of the developed implementations are also discussed.

6.1 4DCT Data

The image data used here was of a Modus Medical QUASAR Programmable Respiratory Motion Phantom (Modus Medical Devices Inc., London, ON) [136]. The phantom is designed to mimic breathing motion in patients and is shown in Figure 6.1. There are three salient parts to the phantom: the main phantom body, the phantom insert and the marker platform, as illustrated in Figure 6.1.

The main phantom body mimics a patient's body for the purposes of imaging and surrounds the phantom insert.

The phantom insert is a cylindrical piece that moves periodically, mimicking the movement caused by respiration of a patient's internal anatomy in the superior-inferior direction. The phantom insert used here was the QUASAR Imaging Insert [137], which

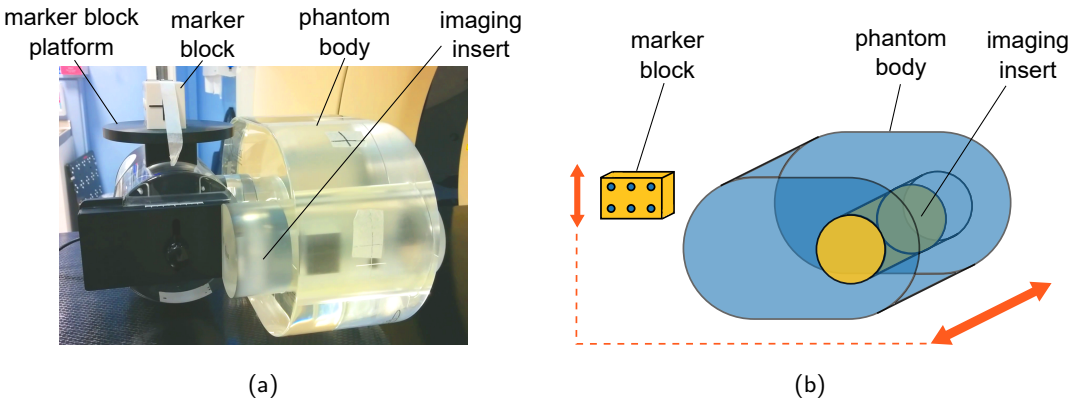


Figure 6.1: QUASAR phantom shown (a) photographically and (b) schematically

contains a set of target objects, as shown in Figure 6.2. The target objects consist of a Delrin cube of side 30mm, two Delrin spheres with 10mm and 20mm diameters respectively, and a 1mm diameter steel ball.

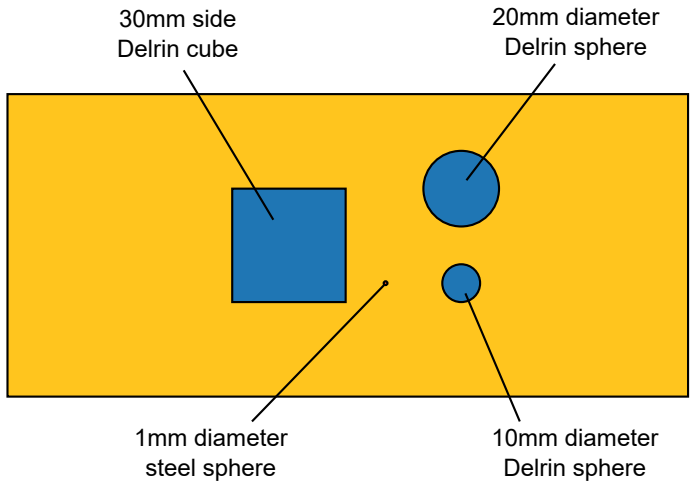


Figure 6.2: Section through the QUASAR Imaging Insert

Movement of the phantom insert was synchronised with the movement of the marker platform. The marker platform motion replicates the movement of the anterior surface of a supine patient's chest during breathing and allows a marker block to be placed on it. The position of the marker block was tracked using a video camera to act as a surrogate for the motion of the phantom insert and to assign the captured image data to its appropriate motion phase.

Each period of motion was divided into ten phases with the first phase beginning at the peak inhale point, when the marker block was at its highest position. The other phases represented equal time durations between peak inhale points, as illustrated in Figure 6.3.

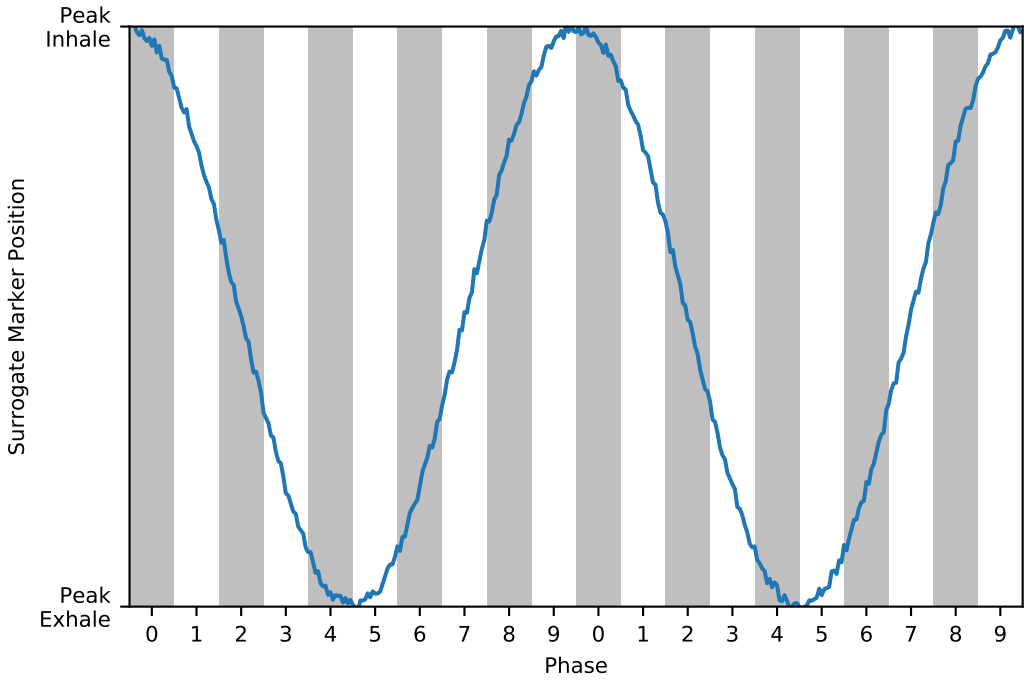


Figure 6.3: Illustration of marker block position against time showing the division of the motion period into phases

Each 4DCT dataset therefore contained ten image volumes, each representing a different phase of the motion. Additionally, an image volume was produced using all of the captured image data in the dataset and three further volumes were produced from a combination of the phase volumes:

- the maximum intensity projection — represents each pixel by the maximum intensity value across all phases;
- the minimum intensity projection — represents each pixel by the minimum intensity value across all phases;

- the average projection — represents each pixel by the average intensity value across all phases.

A conventional CT image volume was also captured for each of the eight 4DCT datasets used here. In total, the eight datasets used here comprised 120 image volumes.

The phantom allows the range of motion of the phantom insert to be configured. Two ranges of motion were used for the image data here: four datasets with a 30mm range of motion and four datasets with a 15mm range of motion.

6.2 Segmentation based on Otsu's Method

Otsu's method is a widely used method for determining optimal thresholds to segment a greyscale image into different classes based on pixel intensity [138–140]. The object of the algorithm as applied here was to find the optimal set of thresholds to segment the 4DCT image data into three classes corresponding to air, phantom body and the target objects within the imaging insert.

The image data of a phantom provided a more simplistic scenario than is likely in the clinical setting, where the imaged volume largely contained only three, quite distinct materials. An approach based on Otsu's method was therefore selected as it was thought this may be effective given this scenario.

6.2.1 Otsu's Method

Otsu's method works by iterating over all sets of possible thresholds to find the set that minimises the variance of pixel intensities within the segmented classes. It was shown by Otsu that minimising the within-class variance is equivalent to maximising the between-class variance, σ_B^2 [6].

In the case of segmenting a greyscale image into K classes, σ_B^2 is given by Equation 6.1, where P_k is the probabilty that a pixel belongs to the kth class, μ_k is the mean intensity of the pixels in the kth class and μ_T is the mean pixel intensity in the image.

$$\sigma_B^2 = \sum_{k=0}^{K-1} P_k \mu_k^2 - \mu_T^2 \tag{6.1}$$

For a set of thresholds, $\{T_1, \ldots, T_{K-1}\}$, P_k is given by Equation 6.2, where N is the number of pixels in the image and n_i is the number of pixels in the image with intensity i. The lower limit for the first class is given by $T_0 = 0$, while the upper limit for the Kth class is given by $T_K = L$, where the pixel intensities in the image are on the interval $[0, L-1]$.

$$P_k = \frac{1}{N} \sum_{i=T_k}^{T_{k+1}-1} n_i \tag{6.2}$$

All possible sets of threshold values need to be searched to find the set of thresholds, $\{T_1^*, \ldots, T_{K-1}^*\}$, that maximises the value of σ_B^2. In the case where there is more than one set of optimal thresholds, the average of each of these sets is taken [6].

Practical Computation

The search space for the optimal set of thresholds is constrained by the condition: $0 < T_1 < \ldots < T_{K-1} < L - 1$. Furthermore, the μ_T term in Equation 6.1 is the same for each set of thresholds and, therefore, the optimal set of thresholds are simply those that maximse the first term in Equation 6.1, termed the modified between-class variance, ${\sigma_B^2}'$ and given in Equation 6.3 [6].

$${\sigma_B^2}' = \sum_{k=0}^{K-1} P_k \mu_k^2 \tag{6.3}$$

Although this reduces the computational burden, ${\sigma_B^2}'$ must still be calculated $(L - K + 1)^{K-1}$ times. It is, therefore, desirable to reduce the computational effort required for each calculation. This can be achieved by framing the computation of ${\sigma_B^2}'$ in terms of the results of recursive operations on the image histogram [141].

The computation of P_k lends itself well to this approach. The probability of a pixel having an intensity between 1 and v is given by the relationship in Equation 6.4, with the special case for the initial condition, $P(1, 0) = 0$. P_k can then be calculated for an arbitrary T_k and T_{k+1} as shown in Equation 6.5.

$$P(1,v) = P(1,v-1) + \frac{n_v}{N} \tag{6.4}$$

$$P_k = P(1,T_{k+1}) - P(1,T_k) \tag{6.5}$$

The computation of μ_k, on the other hand, does not lend itself so well to a simple recursive calculation. Instead, it can be substituted in the calculation of ${\sigma_B^2}'$ for the cumulative mean of the class, m_k. Equations 6.6 and 6.7 show the relationship between μ_k and m_k, and express ${\sigma_B^2}'$ in terms of m_k, respectively.

$$\mu_k = \frac{m_k}{P_k} \tag{6.6}$$

$${\sigma_B^2}' = \sum_{k=0}^{K-1} \frac{m_k^2}{P_k} \tag{6.7}$$

The relationship between m_k and the image histogram is shown in Equation 6.8. From this, it can be seen that the cumulative mean for pixels with intensities between 1 and v is given by Equation 6.9, with the special case for the initial condition, $m(1,0) = 0$.

$$m_k = \frac{1}{N} \sum_{i=T_k}^{T_{k+1}-1} in_i \tag{6.8}$$

$$m(1,v) = m(1,v-1) + v \cdot \frac{n_v}{N} \tag{6.9}$$

m_k can then be calculated for an arbitrary T_k and T_{k+1} as shown in Equation 6.10.

$$m_k = m(1,T_{k+1}) - m(1,T_k) \tag{6.10}$$

This approach allows $P(1,v)$ and $m(1,v)$ to be computed recursively for $v \in [1, L-1]$ and written to look-up tables. The computation of either P_k or m_k for any class with arbitrary thresholds is then reduced to two table look-ups and a subtraction, thus reducing the computational burden during the search for the optimal thresholds.

6.2.2 Applying Otsu's Method to 4DCT Phantom Image Data

Hardware and software implementations of Otsu's method were developed with the aim of finding optimal thresholds for segmenting the 4DCT phantom image data into air, phantom body and target objects within the imaging insert.

Limitations

The global nature of the optimal threshold generation process used in Otsu's method was found to be problematic when applying it to the 4DCT image data used here.

The most obvious way to apply Otsu's method to the 4DCT image data would be to apply it to each volume in turn. However, the thresholds arrived at using Otsu's method depend upon the distribution of the pixel intensities in the entire image volume. Therefore, classes with a low proportion of pixels in the image, such as those in the target objects within the imaging insert, can be lost in the natural variations of pixel intensities within classes forming higher proportions of the image. This is illustrated in the pixel intensity distribution for one of the image volumes shown in Figure 6.4. It can be seen that the number of pixels that should have been classified as air, at values around -1000HU, dominate the distribution, while pixels that should have been classified as belonging to target objects, at values around 300HU, are not visible in the histogram. This resulted in the segmentation failing to segment the target objects in the imaging insert from the rest of the phantom. An example of the segmentation produced is also shown in Figure 6.4.

The application of Otsu's method to the image data on a slice-by-slice basis was precluded by the fact that some slices in each image volume would certainly not contain any pixels belonging to the target objects in the imaging insert. The thresholds produced for these image slices would, therefore, be spurious.

Instead, a $32 \times 64 \times 128$ pixel subvolume was selected from within the image volumes. The same subvolume was selected from each volume. The subvolume was selected to contain a more equal proportion of pixels from each of the target classes than in the

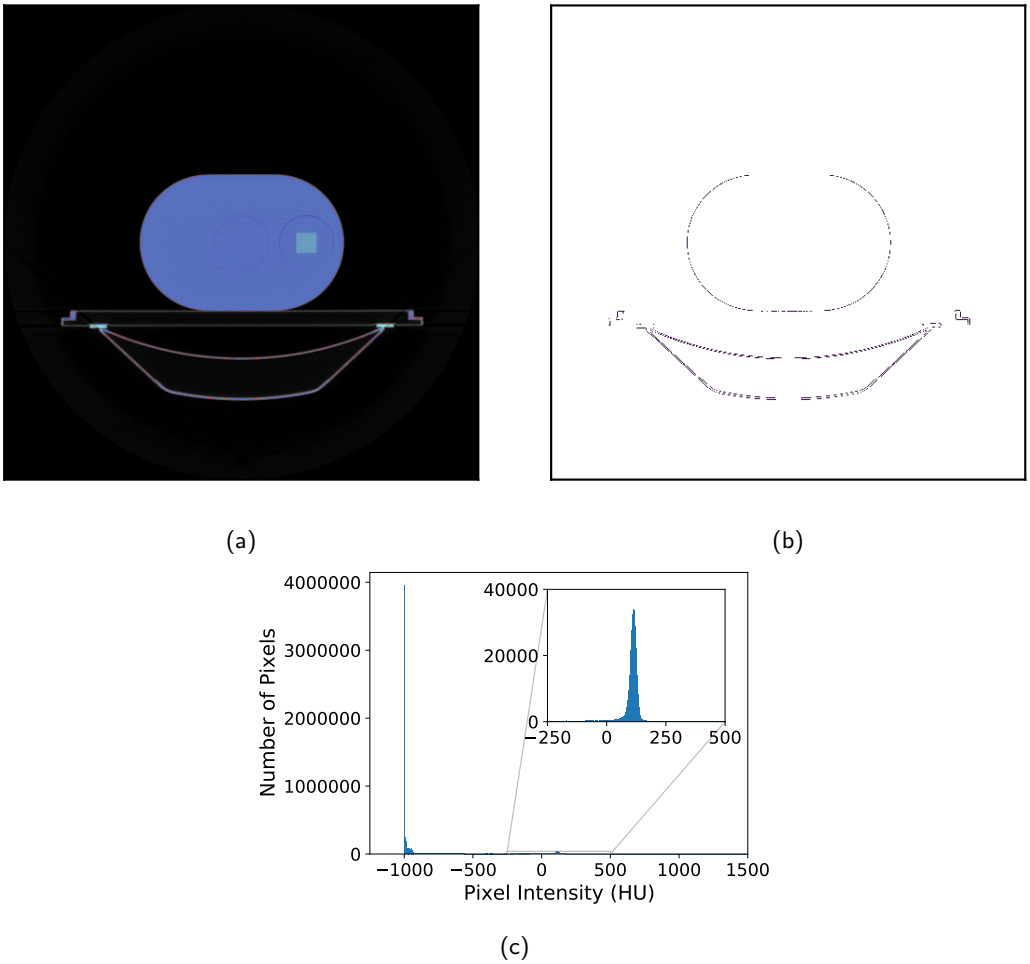


Figure 6.4: Application of Otsu's method-based segmentation to a full 4DCT image volume showing (a) the segmentation produced, with the pixels classified as phantom body shown in red and those classified as target object shown in blue, (b) the pixels classified as phantom body shown in isolation for clarity, and (c) the histogram of pixel intensities.

complete volumes and to encompass the full range of motion of the target objects in the imaging insert. The selection of the subvolume depended on *a priori* knowledge of the location of the target objects within the image volumes and their range of motion.

Noise Reduction

To reduce the effect of noise on the segmentation, a mean filter was applied to the selected subvolumes. The value of an output pixel from the mean filter was calculated as the mean of the corresponding input pixel and its 26 nearest neighbours, as shown in Figure 6.5. For input pixels at the boundaries of the subvolume, a replication scheme [142] was used to assign values to the neighbouring pixels that did not correspond to pixels within the subvolume. The thresholds obtained from applying Otsu's method to the original volume were used to segment the filtered subvolume. This approach was termed the *4DCT Optimised Threshold* algorithm.

An alternative averaging filter type, such as a median filter, could have been used in place of the mean filter. A median filter would tend to be more robust against isolated spurious pixel values caused by noise, since its output is not affected by outlying values in the input. However, the mean filter uses a simpler computation than the median filter and, therefore, has superior execution time performance.

Implementation

Figure 6.6 shows the 4DCT Optimised Threshold algorithm applied to the 4DCT image data used here to segment it into three classes. After the image histogram was constructed for the subvolume, look-up tables of $P(1, v)$ and $m(1, v)$ were created recursively for each bin in the histogram, as described in Equations 6.4 and 6.9 respectively. These look-up tables were used to perform an exhaustive search of each pair of thresholds in the search space, calculating ${\sigma_B^2}'$ for each pair. The first pair of thresholds that were found to maximise ${\sigma_B^2}'$ were returned as the optimal set of thresholds and were used to segment the filtered 4DCT subvolume.

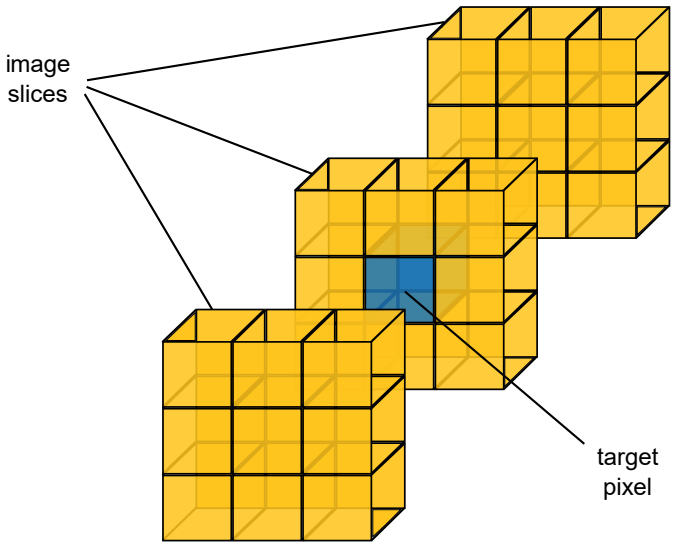


Figure 6.5: Mean filter neighbourhood: the target pixel is shown in blue with neighbouring pixels used in the calculation shown in yellow

Some adaptations were made to the generic Otsu's method described in Section 6.2.1 to improve the performance of the implementation in terms of execution time. These adaptations will now be outlined.

As can be seen from Equations 6.2, 6.7 and 6.8, each term in the computation of ${\sigma_B^2}'$ is divided by the number of pixels in the subvolume, N. Typically this would be accounted for by computing the normalised histogram for the image, where the ith bin of the histogram is expressed as $\frac{n_i}{N}$. However, in determining the optimal thresholds, computing the precise value of ${\sigma_B^2}'$ is not necessary. It is only necessary to find the set of thresholds that maximise ${\sigma_B^2}'$ and so a comparative value is sufficient. The unnormalised histogram was therefore used to compute ${\sigma_B^2}'$ scaled by a factor of N. This still allowed the optimal thresholds to be found while avoiding the computational and hardware expense of normalising the histogram or dividing ${\sigma_B^2}'$ by N.

The range of pixel intensities searched for the optimal thresholds was also restricted to improve the execution time performance of the algorithm. The range was restricted based on application-specific *a priori* knowledge of the expected range in which the thresholds between classes were expected to be found. Having this *a priori* knowledge

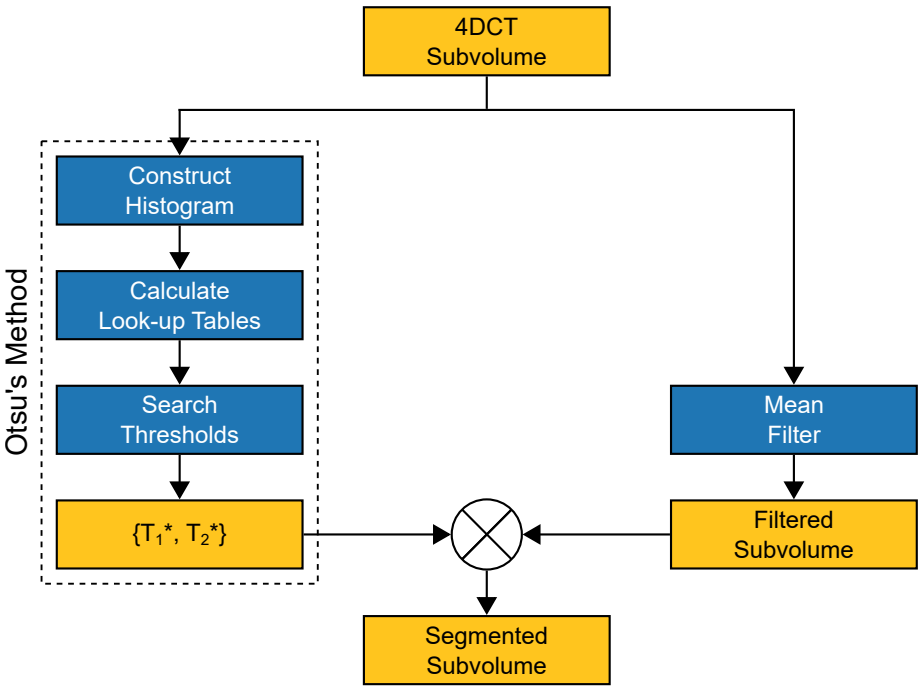


Figure 6.6: The 4DCT Optimised Threshold algorithm

is realistic in the clinical setting for segmenting CT scans into air, soft tissue and bone classes, since the approximate ranges of CT numbers corresponding to these classes are relatively well defined [5].

The search range was restricted to pixel intensities between -600 and 400HU. When constructing the histogram of pixel intensities in the image, any pixels with an intensity less than -600HU were added to the -600HU bin, and any pixels with an intensity greater than 400HU were added to the 400HU bin.

Finally, the first pair of thresholds that were found to maximise ${\sigma_B^2}'$ were returned as the optimal set of threshold rather than returning the average of every set of thresholds that maximised ${\sigma_B^2}'$. This approach was chosen as it still provided an optimal solution to the problem, while also avoiding the computational and hardware expense of recording and post-processing every set of thresholds that maximised ${\sigma_B^2}'$.

Software

The software implementation was written in C++ and executed on an Intel Core-i5 3230M CPU. The generation of optimal thresholds and filtering of the subvolume were performed sequentially. An example of the software code can be found in Appendix B.

Hardware

The hardware implementation used a Xilinx Zynq 7020 SoC device on an AvNet ZedBoard development board. The system was implemented from a software description of the algorithm using the Xilinx SDx v2017.2 software.

The generation of optimal thresholds and filtering of the subvolume were performed concurrently in the programmable logic of the SoC, with the input pixel values streamed to the two implementations simultaneously. The output pixels from the mean filter were streamed back to system memory as they were generated.

Generation of the optimal thresholds took much longer than the mean filtering of the subvolume and caching of the filtered subvolume in programmable logic would have been prohibitively expensive in terms of memory resource utilisation. Therefore, the thresholding of the filtered subvolume using the optimal threholds was performed using the CPU in the processing system to save the additional data transfer overhead from caching the filtered subvolume in system memory and thresholding in the programmable logic.

The use of binary logarithms was considered to further simplify the computation of ${\sigma_B^2}'$ in hardware. This approach has been shown to improve the speed of hardware implementations of Otsu's method in the case of segmenting images into two classes [139, 140].

By approximating the logarithms of m_k and P_k in Equation 6.7, the squaring of m_k becomes a multiplication by two, which is trivial in binary logic, and the division of m_k^2 by P_k becomes a subtraction of $\log_2 P_k$ from $2\log_2 m_k$ [139, 140]. It was found that the time saved from the elimination of a division and a non-trivial multiplication was greater than the time taken to approximate the logarithms of m_k and P_k. Therefore the approach improved the execution time of the algorithm overall [139, 140].

However, in the case of segmentation into three or more classes, the logarithm of the $\frac{m_k^2}{P_k}$ expression must be converted back, by approximating the anti-logarithm, before summing the values for each class to ensure that the true optimal thresholds are found.

It was found that, with the addition of the approximation of the anti-logarithm, the performance advantage of using logarithms over the standard approach was lost, and the standard approach was faster to compute the optimal thresholds while requiring fewer hardware resources. Therefore the standard approach was used for the hardware implementation presented here.

Assessing Performance

A number of facets to the performance of the algorithm implementations presented here were considered.

Firstly, the quality of the segmentations produced by the implementations was assessed. This task was simplified by the segmentations from the hardware and software implementations being identical.

The quality of the segmentations produced was assessed quantitatively by detecting the range of motion of the segmented target objects in the imaging insert and comparing this with the nominal range of motion programmed on the QUASAR phantom for the respective image dataset. To assess the range of motion, the goal was to detect the leading edge of the cube target object in each of the motion phase subvolumes in the dataset. By comparing the minimum and maximum slice that the leading edge was detected across all phases of the motion, and knowing the slice thickness, the range of motion was approximated.

To find slices containing the cube target object, a morphological opening was performed on each slice using a square structuring element of side approximately 15mm. The first slice found with target object pixels after this opening was assumed to be the leading edge of the cube target object. The first five slices of the subvolume (equating to 15mm) were neglected in this operation, as these were likely to contain extraneous parts of the phantom that could be spuriously classified as belonging to the class of target object pixels.

Secondly, the execution time for the hardware implementation was compared to that of the software implementation.

Finally, the FPGA resources required for the hardware implementation were considered.

6.2.3 Results and Discussion

The results of running the software and hardware implementations of the 4DCT Optimal Threshold algorithm on the 4DCT phantom image data are presented and discussed below. The results are divided into those concerning segmentation quality, execution time and the resource utilisation of the hardware implementation.

Segmentation Quality

The detected range of motion of the target objects in the imaging insert are shown in Table 6.1. In each case the detected range of motion matched the nominal range to within 1.5mm; the extent to which measurements could be taken, given the spatial resolution of the image volumes considered.

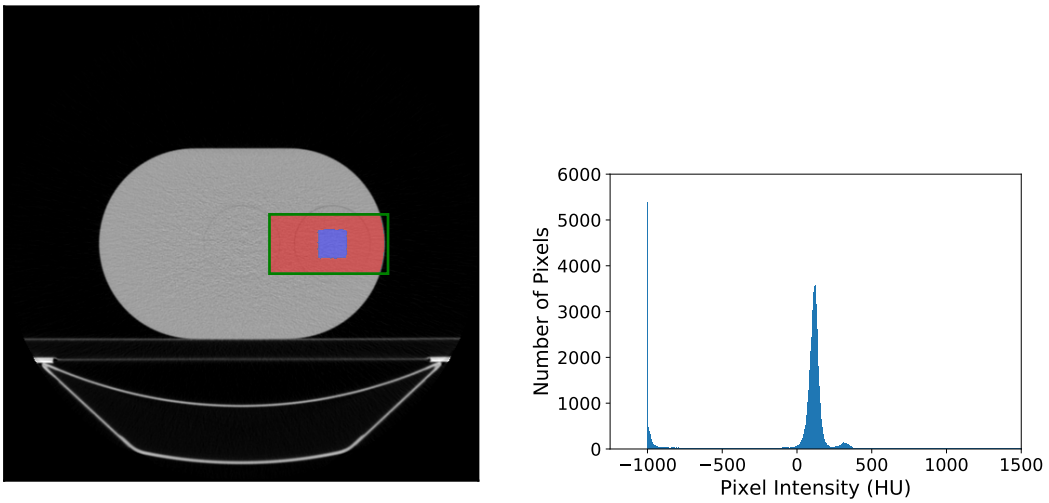
Table 6.1: Ranges of motion for the imaging insert in the 4DCT image data

Dataset	**Nominal Range (mm)**	**Detected Range (mm $\pm$1.5mm)**
1	30	30
2	30	30
3	30	30
4	30	30
5	15	15
6	15	15
7	15	15
8	15	15

The agreement between the nominal and detected ranges of motion tends to indicate that the segmentation produced using the 4DCT Optimal Threshold algorithm was of good quality. However, the accuracy of the range of motion detection algorithm can be

ascribed in large part to the well-known and well-defined shape of the target object. In clinical practice, where these assumptions are not valid, the task of distinguishing between the true target object and noise in the segmentation would be much more challenging. It is unlikely, therefore, that such an accurate measurement of the range of motion could be made in clinical image data. Nevertheless, this approach served the purpose here of providing a quantitative assessment of the segmentation quality.

An example of a typical segmentation is shown in Figure 6.7 along with the distribution of of pixel intensities in the subvolume. It can be seen that the trimodal nature of the pixel intensity distribution is much clearer for the selected subvolume than it was for the entire 4DCT image volume, shown in Figure 6.4 on page 147. This resulted in a greatly improved segmentation of the target objects, as shown in Figure 6.7.



(a) (b)

Figure 6.7: Application of 4DCT Optimal Threshold algorithm showing (a) the segmentation produced — with the pixels classified as phantom body shown in red, those classified as target object shown in blue and the extent of the subvolume shown in green — and (b) the histogram of pixel intensities.

Execution Time

The average execution times for the hardware and software implementations to segment a 4DCT subvolume are shown in Table 6.2.

Table 6.2: Average measured times to process 4DCT subvolume using the two algorithm implementations

Algorithm Implementation	Execution Time (ms)
Software	46.6741
Hardware	23.9212

It can be seen from Table 6.2 that the segmentation in hardware is 1.95 times faster than in software. This difference was found to be statistically significant ($p < .05$).

The performance advantage of the hardware implementation over the software implementation was not nearly so marked as was found for the Haas algorithm applied to the bladder cancer patient data in Chapter 5. Much of the performance advantage that was observed here is likely to be attributable to the hardware implementation executing the mean filtering and Otsu's method concurrently, wheras the software implementation performed these operations sequentially. If a multi-threaded approach were adopted for the software implementation to perform these two operations in parallel, the performance advantage of the hardware implementation would be greatly reduced.

Resource Utilisation

The resource utilisation for the hardware implementation is shown in Table 6.3. This shows relatively low resource utilisation, particularly considering the modest amount of FPGA resources available on the SoC device used here.

Table 6.3: Resource utilisation for hardware implementation

Resource	Utilisation (%)
LUT	19252 (36.19)
FF	23378 (21.97)
BRAM	21.5 (15.36)
DSP	14 (6.36)

Each histogram bin was represented by a 19-bit unsigned integer to be able to denote values up to and including 262144, the number of pixels in the subvolume. To represent all 1599 bins used in the histogram, three 18Kbit BRAMs were required. A further 14 BRAMs were required for the computation of $\sigma_B^{2\prime}$.

The image subvolume was oriented to minimise the amount of memory resources required to buffer an image slice, at the expense of a slower execution time due to there being more slices in the subvolume. The image subvolume of dimensions $32 \times 64 \times 128$ was processed as 128 slices of 32 rows and 64 columns, with each slice containing 2048 pixels.

The mean filter required two slice buffers, each buffering 30 rows of a slice, with the other two rows being contained in row buffers. A slice buffer therefore required enough memory to contain 1920 16-bit values. Each 18Kbit BRAM has capacity to hold 1000 16-bit values, meaning that each slice buffer required two BRAMs and the mean filter required four BRAMs overall. The six row buffers required by the mean filter were implemented using LUT resources rather than BRAMs.

The resource utilisation reported here should be viewed in the context that this implementation used a small subvolume extracted from the main image volume using *a priori* information about the position of the target objects within the image volume. This is a legitimate approach when using a well-defined phantom, the setup of which can be readily reproduced. However, it would be less appropriate in the case of clinical image data, since the position of any target anatomy is unlikely to be so well localised prior to imaging and the reproducibilty of patient setup is more challenging [143, 144]. Therefore, it is anticipated that this implementation would need to be extended to be able to process larger image volumes if it were to be used in ART.

6.2.4 Extension to Clinical Image Data

The 4DCT Optimal Threshold algorithm approach was premised on using specifically selected subvolumes from each of the full image volumes. The subvolumes were chosen to contain significant proportions of pixels belonging to each of the three classes the images were to be segmented into. These selections were based on *a priori* information

about the phantom and its setup. In practice, with clinical image data, such information is unlikely to be available. A more robust approach would be to apply the algorithm to the entire image volume.

Hardware Resource Requirements

Increasing the size of the volume to be processed would increase the execution time of the algorithm commensurate with the number of additional pixels to be processed. For the mean filter, more pixels would need to be filtered and, for Otsu's method, the construction of the histogram would take longer because of the additional pixels. The search for the optimal set of thresholds is unlikely to increase in time, however, since the number of bins in the histogram would not change.

Although the number of bins in the histogram would not increase, the amount of memory required to represent the histogram may do. This is dependent on whether a greater number of bits would be required for each histogram bin, to be able to represent values up to the number of pixels in the volume. The amount of memory required to represent the look-up tables used in the computation of $\sigma_B^{2\prime}$ would also increase with any increase in the bit-width of the histogram values.

Increasing the size of the volume to be processed would also increase the resources required to implement the mean filter in hardware. For the 4DCT image data used here, a full image volume was of dimension $512 \times 512 \times 47$ pixels. Assuming the volume orientation was selected to minimise the size of the image, each slice buffer would need to consist of 45 rows of 512 columns of 16-bit values in order to process a full volume. Each slice buffer would require 24 BRAMs, meaning the mean filter would require 48 BRAMs in total. While this is a substantial increase in the number of BRAMs required compared to the design implemented here, it is still well within the available limits for the SoC device used here of 280 BRAMs.

Faster processing could be achieved, at the expense of memory resources, by using the slice orientation producing the fewest number of slices. For the full 4DCT image volume, this would require processing slices with 512 rows and 512 columns and slice buffers capable of containing 261120 16-bit pixel values. Each slice buffer would need

262 BRAMs, and 524 BRAMs would be necessary for the entire mean filter. The requirements would preclude the use of the SoC device used here and a more expensive device with greater FPGA resources would be needed.

Algorithm Suitability

Although it would be technically feasible to extend the hardware implementation of the algorithm used here to process the full 4DCT volumes, the segmentation shown in Figure 6.4 on page 147 illustrates the poor segmentation results that would be achieved. These findings demonstrate a fundamental weakness in Otsu's method for this application, in the sense that it depends greatly upon there being a reasonable proportion of pixels belonging to each class in the image to accurately identify the optimal thresholds. If one of the classes is poorly represented in the image then the algorithm may identify a subdivision of a more dominant class as the optimal segmentation.

Otsu's method is, therefore, unsuitable for the segmentation of 4DCT data into air, soft tissue and bony anatomy without additonal processing or *a priori* knowledge that the proportion of pixels in the image belonging to each class is relatively balanced. It may, however, be suitable as part of a greater segmentation algorithm for ART, where these criteria can be more readily met, such as the automatic abdominal segmentation algorithm proposed in [145]. Therefore, the results presented here should not be entirely discounted.

Notwithstanding that the hardware implementation of the 4DCT Optimal Threshold algorithm presented here showed a significant speed-up over its software counterpart, the structure of Otsu's method is not well suited to the FPGA architecture. Otsu's method is a global algorithm that relies on all of the pixel values in an image to produce the output. This is in contrast to the Haas algorithm or the mean filter, which can be implemented as a sequence of simple local operations where the output of a stage at any point in time depends only on a few input values that have relatively recently been input to the stage. The computationally expensive processing required for Otsu's method cannot begin until all of the pixels have been recorded in the histogram. This restricts the creation of deep processing pipelines, where the pixel data can be streamed

through a sequence of concurrently executing processing stages. An algorithm that can be decomposed into a series of local operations, such as the Haas algorithm, would be better suited as a segmentation stage in an ART algorithm implemented on an FPGA architecture.

6.3 Segmentation based on Haas' Algorithm

The other approach that was considered for segmenting the target objects from the imaging insert in the 4DCT image data was the same algorithm that was investigated for segmenting bony anatomy in Chapter 5. A brief description of the investigation that was carried out follows in the remainder of this section, along with a presentation and discussion of the results obtained.

Based on these results, an extension to the previously developed implementations was proposed to improve the quality of the segmentation in the presence of noise. The results from implementing this extended algorithm in software and hardware are also presented and discussed here.

6.3.1 Application of the Haas and Simplified Haas Algorithms

The same algorithms that were used for bone segmentation of CT scans of bladder cancer patients in Chapter 5 were applied to the 4DCT image data. These were, namely, the *Haas* algorithm implemented in software, and the *Simplified Haas* algorithm implemented in both software and hardware.

The same platforms that were used for the software and hardware implementations in Chapter 5 and for the Otsu-based approach were also used here.

The algorithm implementations were applied to each of the 120 4DCT image volumes available in the datasets. For the Simplified Haas algorithm, a subvolume of interest corresponding to an approximately 340×170mm region in the centre of each slice was extracted for processing.

Two methods were used to assess the quality of the segmentations produced. Firstly, the segmentations produced by the Haas and Simplified Haas algorithms were assessed by detecting the range of motion of the segmented cube target object in the eight datasets employed in this work. The detected range of motion was then compared to the nominal range of motion for each of the datasets. A more detailed description of the method used is provided in Section 6.2.2.

Secondly, the segmentations produced for each image volume using the Simplified Haas algorithm were compared to those produced using the Haas algorithm. The DSC was calculated to provide a quantitative assessment of the segmentation quality.

The execution times of the implementations were measured and compared.

The resource utilisation for the hardware implementation of the Simplified Haas algorithm is the same as that reported and discussed in Chapter 5. Therefore, it is not reported or discussed any further here.

6.3.2 Results and Discussion

The results obtained are presented in this section. Firstly, the results from measuring the execution time of the implementations are presented and discussed. These are followed by the results obtained from assessing the quality of the segmentations produced.

Segmentation Quality

The quality of the segmentations produced using the Haas and Simplified Haas algorithms was assessed by detecting the range of motion of the segmented cube target object in the phantom's imaging insert. Table 6.4 shows the nominal and detected ranges of motion.

It can be seen from the results in Table 6.4, that in each of the datasets tested here, the segmentations produced by both the Haas and Simplified Haas algorithms accurately identified the range of motion of the target objects from the image data.

Table 6.4: Segmented target object range of motion

Dataset	Nominal Range (mm)	Detected Range (mm ±1.5mm)	
		Haas	Simplified Haas
1	30	30	30
2	30	30	30
3	30	30	30
4	30	30	30
5	15	15	15
6	15	15	15
7	15	15	15
8	15	15	15

On the face of it, these results suggest that the segmentation produced by the Simplified Haas algorithm was as good, in terms of detecting the range of motion of the target objects, as that produced using the Haas algorithm and that, therefore, the two segmentations are of similar quality. However, it must be borne in mind that the algorithm used to detect the range of motion was based on having *a priori* knowledge of the size, shape and orientation of the target objects, making the algorithm robust against noisy segmentations. While this premise is perfectly valid when phantom data is being used, it is extremely unlikely to be equally valid in clinical practice. In clinical practice, an algorithm for detecting the range of motion from a segmentation is likely to be far less robust against noise in the segmentation and is unlikely to produce as accurate results as have been found here.

A further assessment of the quality of the segmentation produced using the Simplified Haas algorithm was made by comparing it directly to the segmentation produced using the Haas method on a volume-by-volume basis. The DSC measure [132] was used to quantify the similarity between the segmentations, and the distribution of the results is shown in Figure 6.8.

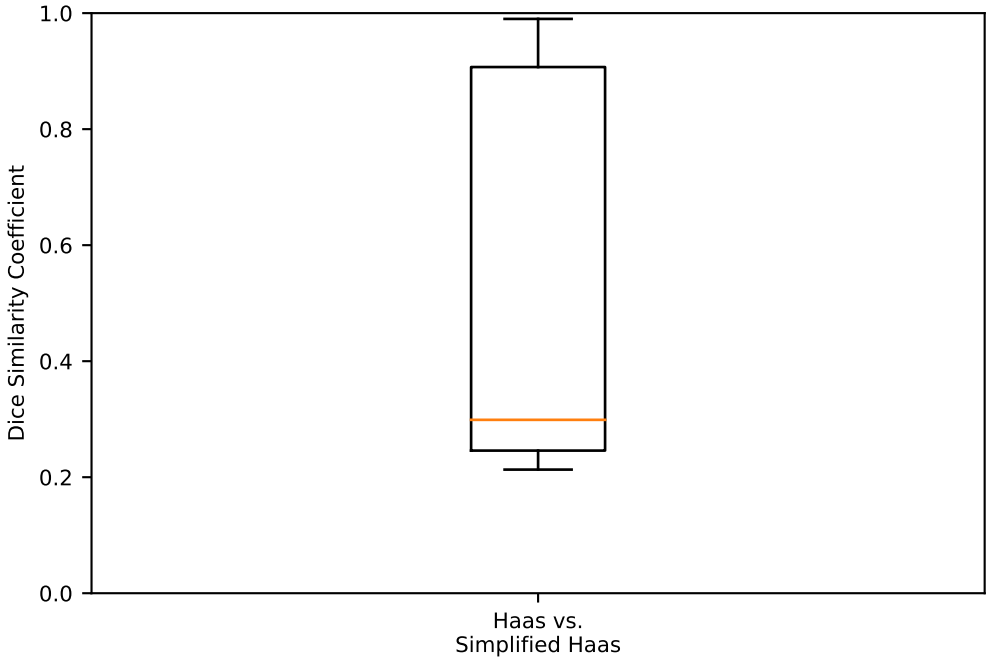
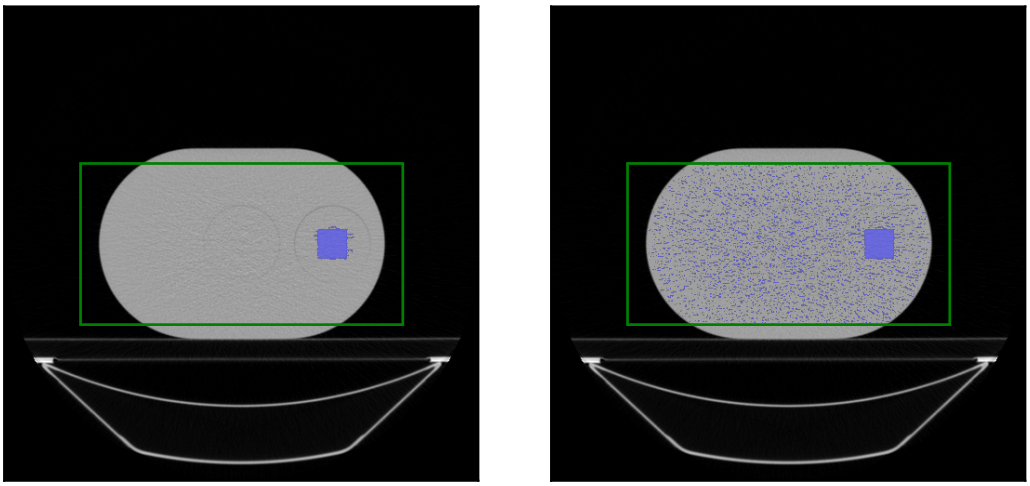


Figure 6.8: DSC resulting from a comparison of the segmentations produced by the Simplified Haas and Haas Algorithms

It can be seen from Figure 6.8 that in the majority of the 120 image volumes tested, the segmentation produced using the Simplified Haas algorithm was not in good agreement ($DSC < 0.7$) with that produced by the Haas algorithm.

A visual comparison of the segmentations produced using the two algorithms revealed that the segmentation produced using the Simplified Haas algorithm contained a good deal more noise than that produced using the Haas algorithm. Figure 6.9 shows a typical comparison from the two algorithms. This finding corresponds with the findings presented in Chapter 5, where the quality of the segmentation produced using the Simplified Haas algorithm is significantly reduced when noise is present in the image. Given the inherently noisy nature of 4DCT image data, the Simplified Haas algorithm would need to be adapted to improve its robustness to noise to make it suitable for use with 4DCT data.



(a) (b)

Figure 6.9: Typical segmentations produced by the (a) Haas algorithm and (b) Simplified Haas algorithm. The region of interest processed by the Simplified Haas algorithm is shown in green.

Execution Time

The average time taken to process an image volume using the Haas and the Simplified Haas algorithms are shown in Table 6.5. The average time taken to process an image slice is also shown.

Table 6.5: Average time to segment 4DCT image data

Implementation	**Time to Segment Volume (ms)**	**Time to Segment Slice (ms)**
Haas Algorithm in Software	300.0594	6.3342
Simplified Haas in Software	17.5056	0.3725
Simplified Haas in Hardware	20.0414	0.4293

The results in Table 6.5 show that the execution of the Simplified Haas algorithm, in hardware and software, was much faster than the Haas algorithm in software. By 14.97 and 17.14 times, respectively, for an image volume. These results were found to be statistically significant ($p < .05$).

The software implementation of the Simplified Haas algorithm was found to be slightly faster than the hardware implementation, around 14.5% faster for an image volume, although this result was not statistically significant. This finding is the converse of that found in Chapter 5, where the hardware implementation was found to perform slightly better than the software implementation.

Although the time taken to process an image slice using the hardware implementation is broadly similar to those observed when processing the bladder image data, as shown in Table 5.2 on page 5.2, the time taken to process a slice of the 4DCT image data using the software implementation was much less than the times recorded for the bladder image data. This difference in processing time between the two sets of image data is difficult to reconcile. It cannot be explained by the difference in the number of pixels per slice between the two sets of image data, as the average number is similar between the two sets and the 4DCT set, in fact, had slightly more on average.

However, the time taken to process a slice was not measured while processing the slices individually, but in the context of processing entire volumes. The volumes in the 4DCT dataset contain far fewer slices than the volumes in the bladder dataset. There may, therefore, have been some memory caching efficiency that was possible because the amount of data in a 4DCT volume was so much lower than that in a bladder volume. This may explain the faster processing of the 4DCT data than the bladder data using the software implementation.

When comparing the relative speed-up of the Simplified Haas algorithm implementations against the implementation of the Haas algorithm on a volume- and slice-wise basis, the effects of slice pipelining in the hardware implementation can be observed. The relative improvement in execution time for the software implementation of the Simplified Haas algorithm over the Haas algorithm was 17.14 times for both the average time to process a slice and a volume. For the hardware implementation, however, the improvement over the Haas algorithm was slightly greater when considering the time to process a volume compared to the time to process a slice, at 14.97 and 14.87 times, respectively.

This pattern was also, obviously, observed in the performance deficit of the hardware implementation relative to the software implementation of the Simplified Haas algorithm. The software implementation was 15.25% faster on a slice-wise basis, but only 14.97% faster on a volume-wise basis.

In the case of the 4DCT data used here, the advantage imparted to the hardware implementation by slice pipelining was limited by the small number of slices in each volume, namely 47. With image data composed of a greater number of slices, the time to process an image volume using the hardware implementation would improve relative to the software implementation of both the Haas and Simplified Haas algorithms. There is likely to be a number of slices above which the volume-wise performance of the hardware implementation is superior to the software implementation of the Simplified Haas algorithm.

The number of slices in the 4DCT image volumes used here was relatively low because the size of the phantom was small compared to a patient, and the extremities of the volume to be imaged were well-defined when setting the extent of the CT scans. In clinical practice, volumes composed of a greater number of slices are likely to be more prevalent [119, 123] given the greater variability in patient setup and poorer definition of the volume of interest prior to CT scanning. The image volumes used in Chapter 5 are likely to be more representative of the typical sizes of clinical image volumes for abdominal therapies.

With image volumes composed of more slices, the execution time performance of the hardware implementation is anticipated to improve relative to the software implementations, due to the increased effect of slice pipelining. For clinical 4DCT scans, the performance of the hardware implementation relative to the software implementations reported here is likely to be pessimistic.

6.3.3 Improving Segmentation Quality

The Simplified Haas algorithm was adapted to improve the quality of the segmentation produced from noisy image data to make it more suitable for use with 4DCT image data.

It can be seen from Figure 6.9 that the noise in the segmentation produced by the Simplified Haas algorithm tended to be individual pixels isolated from other segmented pixels. These types of artifacts can be removed using a morphological opening operation on the segmentation [142]. Moreover, the Simplified Haas algorithm already utilised a morphological opening operation in the creation of the body mask. The Simplified Haas algorithm could therefore be simply augmented with the duplication of this functionality in order to improve its performance on noisy image data. Figure 6.10 shows the adapted algorithm schematically and the new algorithm was termed the *Noise-reduced* algorithm.

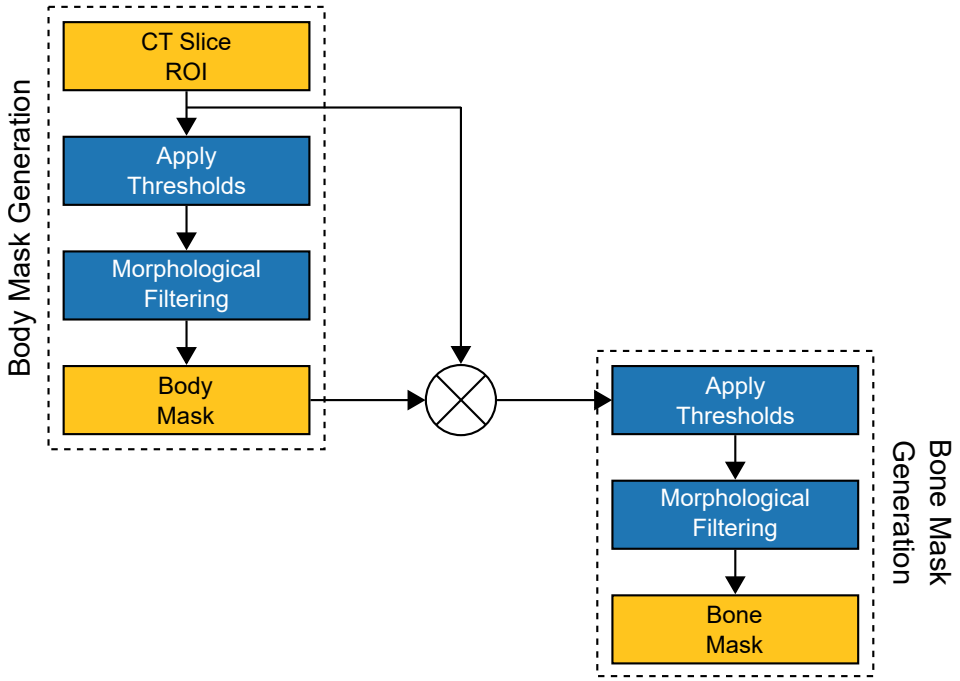


Figure 6.10: Noise-reduced algorithm

The Noise-reduced algorithm was implemented in both software and hardware using the same platforms as were used previously. An example of the code used for the software implementation can be found in Appendix B. The same 4DCT image data were also used.

The quality of the segmentation produced by the Noise-reduced algorithm was, again, assessed by detecting the range of motion of the segmented target objects, as well as comparing it to the segmentation produced using the Haas algorithm. This enabled the effects of the proposed modifications to the Simplified Haas algorithm to be evaluated.

The execution times of the Noise-reduced algorithm were measured for both the hardware and software implementations to compare with each other and those recorded for the Haas and the Simplified Haas algorithms. The resource utilisation of the hardware implementation was also recorded for comparison with that of the hardware implementation of the Simplified Haas algorithm.

6.3.4 Results and Discussion

Segmentation Quality

The main purpose in proposing the Noise-reduced algorithm was to improve the quality of the segmentation produced using the Simplified Haas algorithm. The quality of the segmentation was assessed by comparing it with the segmentation produced using the Haas algorithm and by finding the range of motion of the segmented cube target object.

Figure 6.11 shows the distribution of the DSC values obtained from comparing the segmentations produced using the Noise-reduced and Haas algorithms. The distribution of DSC values obtained from comparing the segmentations produced using the Simplified Haas and Haas algorithms are also shown in Figure 6.11.

It can be clearly seen from Figure 6.11 that, in the majority of cases tested here, the Noise-reduced algorithm produced a segmentation more similar to that produced by the Haas algorithm than the Simplified Haas algorithm. Greater than 75% of the segmentations produced by the Noise-reduced algorithm showed good agreement ($DSC \geq 0.7$) with those produced by the Haas algorithm, compared with less than 50% for the Simplified Haas algorithm. The mean DSC for the Noise-reduced algorithm was 0.83, whereas the mean DSC for the Simplified Haas algorithm was 0.49 and this difference was found to be statistically significant ($p < .05$).

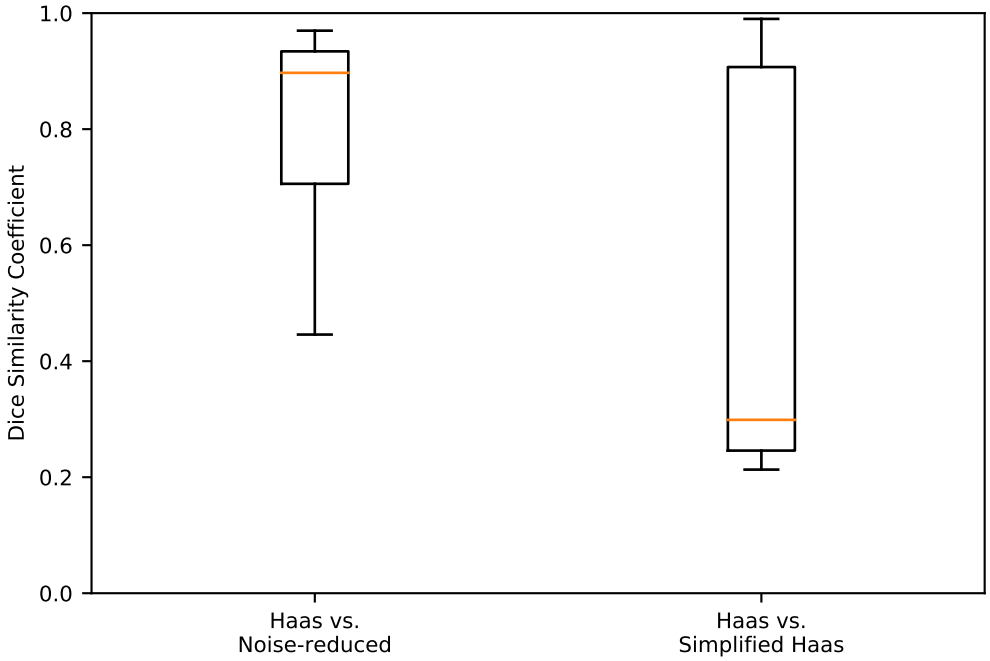


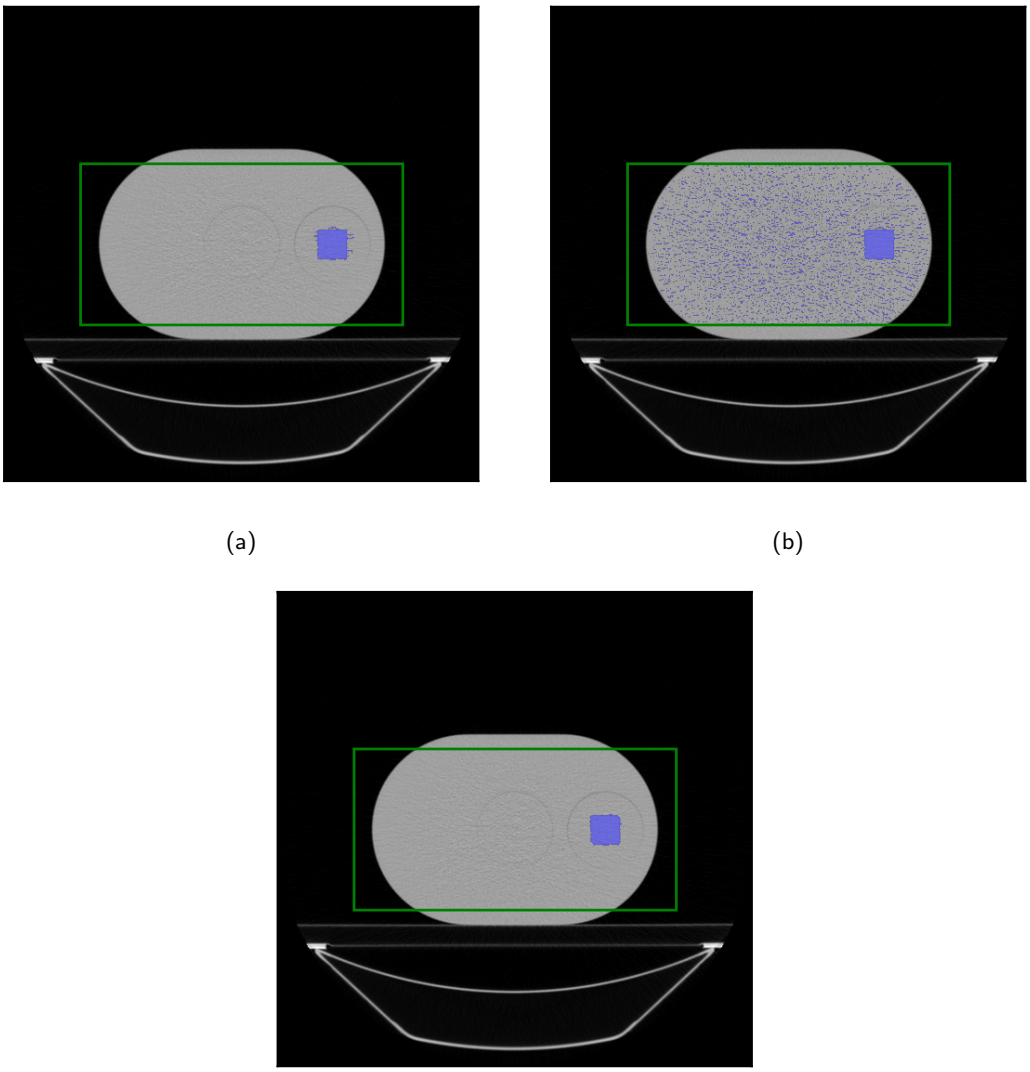
Figure 6.11: DSC resulting from comparing segmentations produced by the Noise-reduced and Simplified Haas algorithms with those produced by the Haas algorithm

These results demonstrate that the proposed additional morphological filtering stages were effective at reducing the susceptibility of the Simplified Haas algorithm to noise. This point was reinforced by a visual comparison between the segmentations of a typical image slice produced using the Haas, Simplified Haas and Noise-reduced algorithms, as shown in Figure 6.12.

The detected range of motion of the segmented cube target object was, again, found to match the nominal range of motion for each of the datasets tested here. This result is unsurprising given that this was also the outcome for the segmentations produced using both the Haas and Simplified Haas algorithms.

Execution Time

The average time taken to process an image volume and slice using the hardware and software implementations of the Noise-reduced algorithm are shown in Table 6.6.



(a) (b)

(c)

Figure 6.12: Typical segmentations produced by the (a) Haas algorithm, (b) Simplified Haas algorithm and (c) Noise-reduced algorithm. The region of interest processed by the Simplified Haas and Noise-reduced algorithms is shown in green.

Table 6.6: Average time to segment 4DCT image data using the Noise-reduced algorithm

Implementation	Time to Segment Volume (ms)	Time to Segment Slice (ms)
Software	22.0274	0.4687
Hardware	20.0533	0.4295

In comparison to the software implementation of the Haas algorithm, the execution time for both the hardware and software implementations of the Noise-reduced algorithm were much reduced, by factors of 14.96 and 13.62, respectively, on average. These differences were found to be statistically significant ($p < .05$). The proposed Noise-reduced algorithm, therefore, still represents a significant improvement over the Haas algorithm in terms of execution time.

Compared to the corresponding implementations of the Simplified Haas algorithm, the execution time of the Noise-reduced algorithm was greater. However, the increase in the mean execution time between the hardware implementations was small, around 0.05%, and was not found to be statistically significant. While the increase in the execution time between the software implementations, on the other hand, was found to be greater, around 25% on average, and statistically significant ($p < .05$).

An increased execution time for the Noise-reduced algorithm was unsurprising given that the Noise-reduced algorithm extends the Simplified Haas algorithm by adding a morphological filter post-processing stage to remove noise from the segmentation.

In the case of the software implementation, the additional processing stages increased the execution time of the algorithm considerably. As each of the additional stages is processed sequentially in the software implementation, the time to process a slice is increased by the sum of the execution times of the added stages. In turn, since each slice is processed sequentially, the time to process a volume is lengthened by the increased amount of time to process a slice multiplied by the number of slices in a volume.

In the case of the hardware implementation, on the other hand, there was no statistically significant increase in the execution time of the Noise-reduced algorithm compared to the Simplified Haas algorithm. The additional processing of the Noise-reduced algorithm increased the depth of the processing pipeline, resulting in a greater capacity for processing pixels concurrently. The increase in processing time on a slice-by-slice basis was, therefore, much less compared to the software implementation.

Furthermore, the results imply an increased slice pipelining effect in the hardware implementation due to the deeper pipeline. This is suggested by the hardware implementation of the Noise-reduced algorithm being 9.13% faster than the software implementation on a slice-by-slice basis, but 9.84% faster on a volume-by-volume basis. Neither of these results were statistically significant.

The results in Table 6.6 show the hardware implementation of the Noise-reduced algorithm was faster than the software implementation with the 4DCT image data used here, although, the difference was not found to be statistically significant.

The performance advantage of the hardware implementation is likely to be further increased when applied to clinical image data. As discussed previously, the image volumes used here were composed of very few slices compared to typical clinical image data. In image volumes with a greater number of slices, the slice pipelining effect would be more pronounced and provide a greater performance advantage for the hardware implementation.

Further improvements in performance of the hardware implementation relative to the software implementation are likely to be exhibited if the depth of the processing pipeline can be increased further. Extending the pipeline with the additional processing for a full ART algorithm may present opportunities to achieve this.

Resource Utilisation

The resource utilisation for the hardware implementation of the Noise-reduced algorithm is shown in Table 6.7.

It can be seen for Table 6.7 that the Noise-reduced algorithm still uses relatively few of the resources available, even on the fairly modest SoC device used here.

Table 6.7: Resource utilisation for the hardware implementation of the Noise-reduced algorithm in absolute terms and as a percentage of the available resources

Resource	**Utilisation (%)**
LUT	8977 (16.87)
FF	10784 (10.14)
BRAM	11 (7.86)

For each of the resources reported in Table 6.7, there is increased utilisation compared to the hardware implementation of the Simplified Haas algorithm shown in Table 5.3 on page 5.3. An increased use of resources relative to the Simplified Haas algorithm was expected, since the Noise-reduced algorithm is the Simplified Haas algorithm augmented with additional morphological filtering stages. The increases in resource usage are shown in Table 6.8, where they are also expressed as a relative increase in terms of the available resources on the SoC device.

Table 6.8: Increase in resource utilisation for the hardware implementation of the Noise-reduced algorithm compared to the Simplified Haas algorithm

Resource	**Utilisation Increase (%)**
LUT	2648 (4.93)
FF	2727 (2.58)
BRAM	4 (2.86)

Table 6.8 shows a small increase in the resource utilisation for the Noise-reduced algorithm compared to the Simplified Haas algorithm. However, given the significant improvement in segmentation quality exhibited by the Noise-reduced algorithm, this small increase in resource utilisation would seem a prudent trade-off.

The small increase in the number of resources used was not sufficient to materially affect the analysis of the factors limiting the implementation of multiple parallel processing pipeline instances provided in Chapter 5, page 5.3.4. Specifically, the finding that the implementation of multiple parallel processing pipelines would be constrained by the bandwidth between the FPGA fabric and system memory on the platform used here remains valid.

Likewise, the changes to the design to reduce the number of resources used proposed in Chapter 5, page 5.3.4 are equally applicable to the design for the Noise-reduced algorithm proposed here.

Similarly to the hardware implementation of the Simplified Haas algorithm discussed in Chapter 5, the Noise-reduced processing pipeline was synthesised and implemented using the default strategies in Xilinx Vivado v2017.2 with a target clock frequency of 100MHz. The worst case negative slack resulting from this was 0.023ns. This, again, implies that the performance of the hardware implementation could have been marginally improved, by around 0.23%, just by increasing the clock frequency used to drive the design. Some improvement may also have been possible to the worst case negative slack to produce a design capable of operating at higher frequencies by adopting a more aggressive implementation strategy.

6.4 Conclusions

Two algorithms were investigated for accelerating the segmentation of 4DCT image data by implementing them in hardware. One algorithm was based on Otsu's method for optimally segmenting images into multiple classes, while the other was based on the work presented in Chapter 5 using the Haas algorithm. Eight sets of 4DCT image data of a quality assurance phantom were used for the investigation.

4DCT Optimal Threshold Algorithm

A hardware implementation of Otsu's method to generate optimal thresholds to segment an image into three classes was developed. Adaptations to the original algorithm were proposed to suit the specific application of segmenting 4DCT data and for its realisation in hardware. Subvolumes, selected from each of the 4DCT image volumes, were used.

A hardware implementation of a three-dimensional mean filter was also developed to reduce the amount of noise in the 4DCT image data. The optimal thresholds generated using Otsu's method were used to segment the image volume after it had been filtered using the mean filter. This algorithm was termed the 4DCT Optimal Threshold algorithm.

The execution time of the algorithm was compared between the software and hardware implementations. The quality of the segmentations produced were also assessed quantitatively by detecting the range of motion of a segmented target object in each of the 4DCT datasets tested. In addition, the feasibility and suitability of extending the algorithm presented here to clinical 4DCT image data was considered.

The algorithm with the adaptations proposed here was shown to be effective at segmenting the subvolumes selected from the 4DCT image volumes. In each dataset tested, the range of motion of the segmented target object was accurately detected.

The hardware implementation was found to be faster than the single-threaded software implementation. The improved performance of the hardware implementation was attributable to more parallel processing, although a multi-threaded software implementation would significantly reduce this advantage.

The performance of the hardware implementation of the mean filter could be improved, at the expense of memory resources, by processing the image volume as fewer, larger slices. Although, this would not improve the overall performance of the segmentation algorithm as the threshold generation, which executed concurrently with the mean filtering, took considerably longer to complete. Additionally, extending this approach to typically-sized clinical image volumes would require more memory resources than were available on the SoC device used here, meaning a larger, and more expensive, SoC device would be needed.

When the algorithm was investigated for extension to process full 4DCT image volumes, it was found to be ineffective at segmenting the volumes into the desired classes. This was due to a disproportionate number of low intensity pixels within the volume.

These findings indicated that the Otsu-based approach was unsuitable for the global segmentation of 4DCT image volumes being considered here. In the context of a larger ART algorithm, however, it may find use as a refining segmentation algorithm following a coarser pre-segmentation.

Haas Algorithm

The software implementation of the Haas algorithm and the hardware and software implementations of the Simplified Haas algorithm presented in Chapter 5 were applied to the 4DCT image data.

The execution times of the implementations were compared and the quality of the segmentations produced were quantitatively assessed. Two methods for assessing the segmentation quality were used. Similarly to the Otsu-based segmentation, the range of motion of a segmented target object in each of the 4DCT datasets was detected and compared to nominal values. In addition, the similarity between the segmentations produced using the Haas and Simplified Haas algorithms was considered.

The segmentations produced using both the Haas and Simplified Haas algorithms allowed the accurate detection of the range of motion of the target object in each of the datasets tested here. However, the segmentations produced using the Simplified Haas algorithm were found to show poor agreement with those produced using the Haas algorithm in the majority of cases. This result corresponded with the finding in Chapter 5 that the Simplified Haas algorithm was adversely affected by noise in the image data. Therefore, the Simplified Haas algorithm is not a reasonable alternative to the Haas algorithm for segmenting 4DCT data due to the noise inherent in 4DCT image data.

Similarly to Chapter 5, the Haas algorithm was found to be significantly slower than both the hardware and software implementations of the Simplified Haas algorithm. However, conversely to the findings in Chapter 5, the software implementation of the Simplified Haas algorithm was found to be slightly faster than the hardware implementation. The reversal of the execution time performance of the two implementations is believed to be related to the considered 4DCT image volumes being composed

of many fewer slices than the image volumes used in Chapter 5. The image volumes used in Chapter 5 were more representative of the size of clinical image volumes and, so, it is anticipated that the execution time performance of the hardware implementation relative to the software implementation reported in this chapter may be pessimistic if the algorithm were extended to clinical 4DCT image data.

Adaptations to the Simplified Haas algorithm were proposed to create a Noise-reduced algorithm with the aim of improving the segmentation quality. The Noise-reduced algorithm was implemented in hardware and software and its performance, in terms of execution time and segmentation quality, was assessed in the same manner as was used for the Simplified Haas algorithm.

The segmentations produced using the Noise-reduced algorithm showed much better agreement with those produced using the Haas algorithm. In the majority of cases tested here, the segmentations produced using the Noise-reduced algorithm showed good agreement with those produced using the Haas algorithm, indicating that the Noise-reduced algorithm is a reasonable alternative to the Haas algorithm with much faster performance for segmenting 4DCT image data.

Again, the hardware and software implementations of the Noise-reduced algorithm were found to execute significantly faster than the Haas algorithm. The hardware and software implementations of the Noise-reduced algorithm were found to execute more slowly than their respective counterparts of the Simplified Haas algorithm. In the case of the hardware implementations, the difference was found to be very small.

In contrast to the Simplified Haas algorithm, the hardware implementation of the Noise-reduced algorithm was found to execute slightly faster than the software implementation. This was attributable to the Noise-reduced algorithm having a deeper processing pipeline than the Simplified Haas algorithm, increasing the concurrent processing compared to the software implementation. It is anticipated that the performance advantage of the hardware implementation would be increased when used with clinical 4DCT data, where the number of slices in the image volume is likely to be greater, therefore increasing the opportunities for concurrent processing.

There was a small increase in resource utilisation between the Noise-reduced and Simplified Haas algorithms. The increase was not sufficient, however, to affect that the bandwidth between programmable logic and system memory was the factor limiting the improvement of the execution time performance on the development board used here.

The results presented in this chapter have illustrated the non-triviality of the task of selecting the most appropriate processing architecture for ART algorithms. The selection can depend not only on the structure of the algorithm, but also on the amount of data to be processed. Although FPGAs are well suited to implementing data streaming algorithms, when processing small amounts of data, the efficiencies achieved by the concurrent processing pipeline of the FPGA are not sufficient to overcome the fast serial processing capabilities of the CPU architecture. However, the longer the processing pipeline, the more advantageous it is to the FPGA architecture, assuming the same throughput can be maintained. With each additional processing stage, the execution time of the FPGA implementation increases only by the latency of the added stage. The execution time of the serial CPU implementation increases by the latency of the added stage for each data element to be processed.

Chapter 7

Conclusion

The aim of this thesis was to investigate the use of hardware acceleration to speed-up algorithms for the segmentation of bony anatomy in CT images, with a view to reducing the plan adaptation time for ART. To achieve this aim, three avenues were investigated:

- the data transfer overhead introduced by integrating an FPGA-based hardware accelerator with existing radiotherapy systems using the DICOM protocol;
- segmenting bony anatomy in three-dimensional image data;
- segmenting bony anatomy in four-dimensional image data.

7.1 Summary

7.1.1 Data Transfer Overhead

Chapter 4 examined the rate at which image data could be transferred to FPGA-based SoC platforms using the DICOM protocol. Two FPGA-based SoC platforms were considered: the ZedBoard, a relatively modest platform that was also used for the hardware implementations elsewhere in this thesis; and the ZCU102, a more expensive platform with a higher performance SoC. The performance of these two platforms was compared with that of a desktop computer and those reported in the literature.

These platforms were chosen primarily due to their availability. Both platforms are general-purpose development boards that have not been specifically optimised for the tasks required by ART. Their performance is, therefore, likely to be lower than that which could be attained using an FPGA-based SoC board specifically developed for ART. However, the data obtained using these platforms has enabled the identification of some of the key criteria when selecting an FPGA-based SoC platform for ART, which are discussed further in Section 7.1.4.

Both the FPGA-based SoC platforms were found to meet or exceed the clinically representative transfer rates reported in the literature in the majority of cases. The ZedBoard was consistently found to have much lower data transfer rates than any of the other platforms tested, both for sending and receiving image data. The ZCU102 was found to have a significantly lower transfer rate than the desktop computer only for receiving image data.

In absolute terms, however, the transfer times of the SoC platforms were found to be sufficiently short that their use as hardware accelerators for ART could not be precluded.

7.1.2 Three-dimensional Images

Chapter 5 considered segmenting bony anatomy in three-dimensional images of bladder cancer patients. The image data was composed of both conventional CT scans and CBCT scans. Modifications to the clinically validated Haas algorithm [5] were proposed to make it more suitable for implementation in hardware. The modified algorithm was termed the Simplified Haas algorithm, and a hardware implementation of it was developed.

The Simplified Haas algorithm was found to execute significantly faster in both hardware and software than the Haas algorithm. On average, the hardware implementation processed CBCT volumes 9.68 times and CT volumes 13.81 times faster than the Haas algorithm. The software implementation, meanwhile, was found to process CBCT volumes 5.11 times and CT volumes 7.73 times faster than the Haas algorithm, on average.

In each of the cases considered, the segmentation produced by the Simplified Haas algorithm showed good agreement ($DSC > 0.7$) with that produced by the Haas algorithm. The Simplified Haas algorithm, and the hardware implementation in particular, therefore demonstrated substantially superior execution time performance to the Haas algorithm, while producing comparable segmentations, and is a reasonable and faster alternative.

The acceleration in real terms was in the order of milliseconds and is inconsequential compared to the typical length of time for a treatment fraction. However, it was shown that, if the relative acceleration achieved with this algorithm were replicated for each of the operations necessary for ART, some of the ART techniques proposed in the literature would then meet the clinically acceptable time requirements and make ART in routine clinical practice a possibility. It also demonstrated the ability of FPGAs to accelerate ART algorithms that can be implemented as data streaming pipelines.

A major obstacle to further improving the execution time performance of the hardware implementation of the Simplified Haas algorithm, by replicating the hardware accelerator to create multiple instances executing in parallel, was identified to be the bandwidth between the hardware accelerator and system memory. It was found that with the FPGA-based SoC platform used, the interconnection between the FPGA fabric and system memory would only be capable of meeting the data transfer requirements of two parallel instances of the hardware accelerator.

It was also noted that the data transfer rates measured for each of the platforms in Chapter 4 were around two orders of magnitude lower than would be necessary to satisfy the processing capacity of a single instance of the Simplified Haas hardware implementation. Based on these findings, it is reasonable to conclude that using the DICOM protocol over an Ethernet connection is likely to be the limiting factor in the performance of hardware accelerators for ART integrated with existing radiotherapy equipment using this method.

7.1.3 Four-dimensional Images

Segmenting bony anatomy in four-dimensional image data of a quality assurance test phantom was investigated in Chapter 6. A novel method was proposed to segment the image data into three classes based on optimal thresholds, derived using Otsu's method [6], and a three-dimensional mean filter to reduce noise in the image. A hardware implementation was developed that was shown to execute 1.95 times faster than the algorithm in software. However, the algorithm was found to be unsuitable for the task of global segmentation of 4DCT images examined here, as it depended on the distribution of pixel intensities in the image being more uniformly spread between the target classes than is reasonable to assume in clinical image data. In the context of a larger ART algorithm, however, it may find use as a refining segmentation algorithm following a coarser pre-segmentation.

The Simplified Haas algorithm developed in Chapter 5 was also applied to the four-dimensional image data. Unlike with the three-dimensional image data used in Chapter 5, the vast majority of segmentations produced from the four-dimensional data showed poor agreement ($DSC < 0.7$) with segmentations produced using the Haas algorithm. The addition of further morphological filtering stages to the Simplified Haas algorithm was proposed to improve the quality of segmentation produced with four-dimensional image data. This new algorithm was termed the Noise-reduced algorithm.

A hardware implementation of the Noise-reduced algorithm was developed and used to segment the four-dimensional image data. The Noise-reduced algorithm showed a great improvement over the Simplified Haas algorithm in terms of segmentation quality. The Noise-reduced algorithm produced segmentations that strongly agreed with those produced by the Haas algorithm in greater than 75% of the cases considered. The hardware implementation of the Noise-reduced algorithm was found to execute 14.96 times faster, on average, than the Haas algorithm, and 1.09 times faster than the software implementation of the Noise-reduced algorithm. Moreover, the execution time of the hardware implementation of the Noise-reduced algorithm only increased marginally over that of the hardware implementation of the Simplified Haas algorithm.

The increase was much smaller than the increase observed between the two software implementations and illustrated a major strength of pipeline parallelism that can be achieved using FPGAs.

Similarly to the Simplified Haas algorithm in Chapter 5, the major obstacle to increasing the parallelism of the hardware implementation of the Noise-reduced algorithm, by replicating multiple instances of the hardware accelerator, was the bandwidth between the accelerators and system memory.

7.1.4 FPGA-based SoCs for ART

A number of criteria were identified that should be considered when selecting or designing an FPGA-based SoC platform for accelerating ART. Given the high cost and power requirements of the typical radiotherapy equipment in clinical use today, the cost and power consumption of the acceleration platform are unlikely to be critical factors.

However, the bandwidth between the FPGA fabric and system memory was recognised as a major obstacle to increasing the level of parallelism exploited by the hardware accelerator. An FPGA-based SoC platform for accelerating ART should have a high bandwidth between the FPGA fabric and system memory. This could be achieved either by increasing the clock frequency of the system memory, or increasing the data bus width between the SoC and system memory. Indeed, there are FPGA-based products that have recently arrived to market that use High Bandwidth Memory (HBM), rather than the DDR memory of the SoC platforms used here [146]. HBM tends to offer drastically increased bandwidth compared to DDR memory, chiefly by utilising much wider data buses.

Selecting an SoC device with a large amount of memory resources in the FPGA fabric is also likely to improve the performance of the platform. Such devices allow more data to be cached in the FPGA fabric and can reduce the number of accesses to system memory. Moreover, access times to data cached in the FPGA fabric tend to be much shorter than those for accessing data in system memory.

Indeed, an SoC device with large amounts of FPGA fabric is desirable more generally. Large amounts of FPGA fabric enables increased parallelism to be achieved, through implementing deep processing pipelines and multiple parallel accelerator instances. Furthermore, FPGA fabric has tended to become faster over time as the process node used for its manufacture has reduced. Selecting the fastest available FPGA fabric may allow the same design to operate at higher clock frequency and, therefore reduce its execution time.

The platform should also have enough system memory to be able to contain all of the data required for an entire ART fraction. This avoids the need to read and write data to much slower non-volatile storage and will increase the rate at which data can be exchanged with existing radiotherapy equipment. The transfer overhead from exchanging data with existing radiotherapy equipment was assessed here using the DICOM protocol over a 1Gb/s Ethernet connection. There are, however, a range of alternative technologies that are likely to provide significant improvements in the transfer rate and should be considered. For example, the SoC device on the ZCU102 platform tested here has a Peripheral Component Interface express (PCIe) interfaces that could be used [97]. There have also been FPGA-based SoC devices introduced to the market recently that incorporate both PCIe and 100Gb/s Ethernet interfaces [147]. Either of these technologies should provide a significant improvement in transfer rate compared to the 1Gb/s Ethernet connection used in this work.

7.2 Further Work

There are a number of opportunities to extend the work presented in this thesis.

Extension to a Full ART Pipeline

Bony anatomy segmentation algorithms are likely to form the very first stage in a long processing pipeline to perform plan adaptation for ART. Subsequent processing stages may include segmentation for other anatomical features, rigid and non-rigid image registration, plan adaptation and dose calculation algorithms. Hardware acceleration

using FPGA-based SoC platforms should be investigated for each of these stages to create a full, hardware-accelerated ART processing pipeline in order to fully assess the feasibility of this approach to enabling ART as a routine clinical technique.

Future Development of ART

ART is still an emerging technology and its clinical implementation is likely to be subject to significant changes and developments in the near future. Improvements in the availability of high quality image data at the time of treatment delivery, such as with the introduction of MR linacs, has tended to increase the interest in ART. This may continue as other imaging modalities, including functional imaging [41], become more routinely available at treatment delivery. Increasing the amount of imaging data available at each treatment fraction could increase the accuracy with which tumours can be targeted. However, it also increases the amount of processing that needs to be performed, and makes selecting the optimal processing architecture even more vital to meet the time requirements of ART. Furthermore, the increasing availability of high quality, real-time imaging during the delivery of treatment is likely to further increase interest in inline ART, with its much more challenging time requirements.

Currently, there is no clear consensus on the most appropriate algorithms for ART and this is an area of very active research. For example, deep-learning methods are starting to emerge as popular candidates for investigation [41]. Given this lack of clarity on what the best ART algorithms are, it is challenging to identify the optimal processing architecture or architectures for ART in general. However, the work presented in this thesis establishes FPGA-based architectures as strong candidates for accelerating ART algorithms that can be implemented as deep data streaming pipelines.

Even once a clearer consensus on the most appropriate ART algorithms is reached, there are still likely to be a variety of ART algorithms used, which will vary with different cancer sites and with different imaging modalities. While the reconfigurability of FPGAs means they can be relatively easily re-programmed to implement different algorithms, it is unlikely that FPGAs will be the most appropriate architecture for all of the algorithms. Selection of the most appropriate architecture should, ideally,

be done on a case-by-case basis. Indeed, different stages of some algorithms may be better suited to different processing architectures. It is, therefore, probable that the acceleration of ART algorithms will be best served by heterogeneous processing architectures with shared memory and the ability to efficiently dispatch tasks to the processing architecture best suited to them.

The relatively laborious nature of FPGA development compared to other architectures may yet present an obstacle to the widespread adoption of FPGA-based architectures for ART. However, significant efforts are being made to simplify the FPGA development process and to make it similar to that for CPUs and GPUs.

7.3 Concluding Remarks

This thesis accelerated bony anatomy segmentation algorithms for ART using an FPGA-based SoC, and demonstrated the potential for hardware acceleration using FPGAs to help enable ART in the clinic by reducing the plan adaptation time. Although there has been considerable interest in accelerating ART algorithms by implementing them on processing architectures other than the ubiquitous CPU, the use of FPGAs to implement custom processing architectures has been largely overlooked in this field. FPGAs provide the ability to create custom processing architectures tailored to the requirements of the algorithm. However, in cases where the requirements of the algorithm are well-suited to a dedicated processing architecture, such as a CPU or GPU, FPGA implementations are likely to be at a severe disadvantage. Given the variety of processing operations required to implement ART, a platform with a range of heterogeneous processing architectures, including FPGAs, would appear to be a promising candidate for implementing ART in the fastest possible timeframe.

Appendix A

Image Data for DICOM Transfer Rate Testing

This appendix lists the DICOM *Unique Identifier* (UID) for the data used in the work presented in Chapter 4.

Fifty studies and fifty series were randomly selected from those in the PROSTATE-DIAGNOSIS collection [123] of TCIA [124]. The *Study UIDs* and *Series UIDs* are listed for the randomly selected studies and series respectively.

A.1 Studies

- 1.3.6.1.4.1.14519.5.2.1.4792.2002.226942730522978355375127754234
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.251248986699057078371521281317
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.256497543454355764493550283953
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.715274401015287661199271453789
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.527348936177830290504929974283
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.289937596945055977296068173104
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.289657662519376183468383514641
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.188945635159044234697126286661

- 1.3.6.1.4.1.14519.5.2.1.4792.2002.208388284708806018229084271566
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.283673389930228934790889933385
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.130996941903934368671182206591
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.329083844110698086185255403810
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.336789281171732592675589569351
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.231189311161071347840729730880
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.255155426083170046699432509461
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.408966338064575230248767257678
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.303748253546622459869186593214
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.302105167530764336558357645458
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.376449827379841987090877335813
- 1.3.6.1.4.1.14519.5.2.1.4793.2002.226613315564674431333961061583
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.238114008950064360545354434916
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.282109840312056580799247662402
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.297593141156589992910905382889
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.265117863364529695705442737331
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.189523095739709751173738978686
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.338621801610608700452868970648
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.160199590397995084811937433399
- 1.3.6.1.4.1.14519.5.2.1.4793.2002.246453091085545123551767634750
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.565068864034569366067102158753

- 1.3.6.1.4.1.14519.5.2.1.4792.2002.252274654986257881821391001634
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.281618486289602641777972668126
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.208539110710641195855467414500
- 1.3.6.1.4.1.14519.5.2.1.4793.2002.136841246334736805333667834346
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.533091514477506692042869439615
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.844035924340532631879159657538
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.228775316018976813333732911408
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.261862605115390886640658075495
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.387539693221715962674080857663
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.390669812782637886659974333876
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.231206157401050134405695297067
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.229434241476122091745021333332
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.269119633629915933563063400237
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.641957959102001371594193952048
- 1.3.6.1.4.1.14519.5.2.1.4794.2002.153827229472044097174834462636
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.969707826520202562379081770522
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.230121956554707593293824747756
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.253679888937002110786708421397
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.920024346241839833195823018831
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.260474708817042900099758697427
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.109320896703960819084104222755

A.2 Series

- 1.3.6.1.4.1.14519.5.2.1.4792.2002.201692855911252231509355142799
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.974321269699417802100111133291
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.297034580842473602763721385474
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.229913814498264977835403918384
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.100014848411669268258437163840
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.175917459437297191793468824742
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.844461591723028958575214634105
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.547859932619263163138072576297
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.724094027168216691989845096805
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.118292634596938021305082493999
- 1.3.6.1.4.1.14519.5.2.1.4793.2002.266061128829233494399571309298
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.136237263416420865661412627627
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.291740854230531166235906183470
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.242742826237099500444317897544
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.200599613636525443228849842837
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.313068846876306863944214870057
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.724451007161666432447927174075
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.277397871450460662011364841784
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.213214392796774544946089585754
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.326070834847479417929952413225

- 1.3.6.1.4.1.14519.5.2.1.4792.2002.448669710529908229140570733660
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.271262052841909559555518598442
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.210561409473093405191930387844
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.268096098263779549646061123539
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.486381358798545140660852870181
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.194554989030301084144241934104
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.602306478149735272752089070548
- 1.3.6.1.4.1.14519.5.2.1.4793.2002.192410886982882241154275002635
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.324261515746447796308356044556
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.123440947254258005332059294934
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.233726231212794709666050860512
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.211718631963740729482447646754
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.939371065319004733843568275859
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.148905023997614579737171972996
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.590502518321849465672425273207
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.205479958475605908390270478944
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.222086594714645919154278216729
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.239745692969312974757081064337
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.137956770522326851616178526259
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.374103389997445025478257817439
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.808325518953056805217365462536

- 1.3.6.1.4.1.14519.5.2.1.4792.2002.138939646057453992715025162382
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.173750612921654303090895528898
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.548594553994577317322914624600
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.328272913700201150291112016673
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.914436865667152895709403568872
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.222670508954246125855524820719
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.304009706256340482269530054975
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.298266279315485073430814787893
- 1.3.6.1.4.1.14519.5.2.1.4792.2002.141390060439549754968061238051

Appendix B

Code Listings

This appendix provides examples of the source code created as part of the work presented in this thesis. The full source code generated from this project can be obtained from [148], [149], [150], [151] and [152].

B.1 DICOM Transfer Rates

B.1.1 Inserting DICOM Data in the PACS

Listing B.1 shows the C++ application used to insert DICOM studies in the PACS. An overview of the operation of this application is given in Algorithm 1 on page 74. Further details of the code and its application can be found in [148].

```
#include "dcmtk/config/osconfig.h"    // include OS specific configuration
#include "dcmtk/dcmnet/diutil.h"
#include "dcmtk/dcmnet/scu.h"

#include <chrono>    // C++11 for high_resolution_clock

#include <iostream>
#include <fstream>

#include "study_list.h"    // File containing list of studies to transfer

#define NUM_TESTS 50    // number of studies to retrieve
#define MAX_INST_STUDY 512    // max number of SOP instances in a single study

/*
 * Print guidelines for usage of application to the standard output
 */
void printUsage() {
    std::cout << "sndspdtest: Application to send a list of DICOM studies to"
              << " a peer using C-STORE service requests and write the time"
              << " taken to send each study to a results file." << std::endl;
    std::cout << "Usage: sndspdtest dataset" << std::endl;
```

```
    std::cout << "\tdataset - directory containing the studies to be"
              << " transferred" << std::endl;
    std::cout << "\nNOTE: The files comprising the dataset are assumed to be"
              << " arranged in a particular order. Each study instance should"
              << " be contained within in its own subdirectory named for the"
              << " study instance UID. Likewise, each series should be"
              << " contained in its own directory within the appropriate study"
              << " directory and named for the series instance UID."
              << std::endl;

}

/*
 * Find a negotiated presentation context for the given SOP class using one of
 * the listed transfer syntaxes
 */
static Uint8 findPresContext(const OFString& sopClass, DcmSCU& scu,
        const OFList<OFString>& ts) {

    Uint8 pc;
    OFListIterator(OFString) ts_it = ts.begin();

    /*
     * Search from the front of the list of transfer syntaxes given in ts
     * argument for one that has been accepted by the peer application
     */
    while (ts_it != ts.end()) {

        pc = scu.findPresentationContextID(sopClass, *ts_it);

        if (pc != 0)
            return pc;

        ts_it++;
    }

    /*
     * No presentation contexts with any of the listed transfer syntaxes have
     * been accepted by the peer application for the SOP class
     */
    return 0;

}


int main(int argc, char *argv[]) {

    // Parse input arguments
    if (argc != 2) {
        printUsage();
        return -1;
    }
    OFString rootdir = OFString(argv[1]);

    OFLog::configure(OFLogger::INFO_LOG_LEVEL);

    DcmSCU scu;
    scu.setAETitle("SND-SPD-TEST");
    scu.setPeerHostName("192.168.1.103");
    scu.setPeerPort(104);
    scu.setPeerAETitle("ORTHANC");

    // Create list of all uncompressed transfer syntaxes
    OFList<OFString> ts;
```

```
87      ts.push_back(UID_LittleEndianExplicitTransferSyntax);
88      ts.push_back(UID_BigEndianExplicitTransferSyntax);
89      ts.push_back(UID_LittleEndianImplicitTransferSyntax);
90
91      /*
92       * Propose MRI storage SOP class to peer with the transfer syntaxes listed
93       * previously. This assumes that the SOP Instances being transferred are
94       * all of the MRI storage SOP class, as is the case with the
95       * Prostate-Diagnosis dataset from the cancer imaging archive.
96       */
97      scu.addPresentationContext(UID_MRImageStorage, ts);
98
99      // Initialize network
100     OFCondition result = scu.initNetwork();
101     if (result.bad()) {
102         DCMNET_ERROR("Unable to set up the network: " << result.text());
103         return -1;
104     }
105
106     // Negotiate Association
107     result = scu.negotiateAssociation();
108     if (result.bad()) {
109         DCMNET_ERROR("Unable to negotiate association: " << result.text());
110         return -1;
111     }
112
113     /*
114      * Find a presentation context for the C-STORE service that has been
115      * accepted by the peer
116      */
117     T_ASC_PresentationContextID pcID = findPresContext(
118         UID_MRImageStorage, scu, ts);
119     if (pcID == 0) {
120         DCMNET_ERROR("There is no accepted presentation context for C-STORE");
121         return -1;
122     }
123
124     // Send a set of DICOM studies to peer
125     int storedStudyCount = 0;
126     OFListIterator(OFString) sopInst;
127     OFList<OFString> storedStudyUIDs;
128     Uint16 storeResp[MAX_INST_STUDY];
129     Uint16 storeRqCount;
130     OFBool testResp;
131     OFString studydir;
132     OFList<OFString> file_list;
133
134     /*
135      * Dummy variables required for call to default DCMTK sendSTORERequest()
136      * function
137      */
138     DcmDataset* dummyDataset = NULL;
139     OFString dummyMoveOrigTitle = "";
140     Uint16 dummyMoveOrigMsgID = 0;
141
142     // https://www.pluralsight.com/blog/software-development/how-to-measure-
    execution-time-intervals-in-c--
143     /*
144      * Create objects to measure and record time taken for each C-MOVE request
145      * to complete:
146      *   - two time_point objects using the high_resolution_clock time_point
147      *     type to record the time at the start and finish of each C-MOVE
148      *     request
149      *   - an array of duration objects to hold the durations of each C-MOVE
```

```
     *     request
     */
    std::chrono::time_point<std::chrono::high_resolution_clock> start, finish;
    std::chrono::duration<double> intervals[NUM_TESTS];
    long int timestamps[NUM_TESTS];

    // Every loop run attempts to send a C-STORE request for a study
    for (int i = 0; i < 5; i++) {

        // Construct path to directory containing study to transfer
        studydir = rootdir;
        studydir += "/";
        studydir += study_uid_list[i];

        std::cout << "Current Study directory: " << studydir << std::endl;

        // Check whether constructed path exists and if it is a directory
        if ( OFStandard::dirExists(studydir) ) {

            // Start with an empty list of files
            file_list.clear();

            /*
             * Get a list of all the files contained within the study directory
             * and its subdirectories. It is assumed that all of these files
             * are MRI Storage SOP class instances
             */
            OFStandard::searchDirectoryRecursively(studydir, file_list);

            /*
             * Issue C-STORE requests for the SOP instances belonging to the
             * selected study
             */

        if(result.good())
            std::cout << "result is OK before sending STORE requests" << std::
    endl;
        else
            std::cout << "result is bad before sending STORE requests" << std::
    endl;

            sopInst = file_list.begin();
            storeRqCount = 0;
            start = std::chrono::high_resolution_clock::now();
            while ((sopInst != file_list.end()) && result.good()) {
                result = scu.sendSTORERequest(pcID, OFFilename(*sopInst),
                  dummyDataset, storeResp[storeRqCount], dummyMoveOrigTitle,
                  dummyMoveOrigMsgID);
                storeRqCount++;
                sopInst++;
            }
            finish = std::chrono::high_resolution_clock::now();


            /*
             * Test whether C-STORE requests were issued and responses received
             * successfully
             */
            if (result.good()) {

                // Test the received response status code(s)
                testResp = OFTrue;
                for(int j = 0; j < storeRqCount; j++) {
                    if(storeResp[j] != 0) {
```

```
                        testResp = OFFalse;
                        break;
                    }
                }

        /*
         * Responses from peer indicate that the C-STORE requests
         * succeeded. Record the time taken to complete the C-STORE
         * requests and the study instance UID of the study that has
         * been stored by the peer
         */
                if(testResp) {
                    intervals[storedStudyCount] = finish - start;
                    timestamps[storedStudyCount] =
                       std::chrono::duration_cast<std::chrono::seconds>
                       (start.time_since_epoch()).count();
                    storedStudyUIDs.push_back(study_uid_list[i]);
                    DCMNET_INFO("Store request succeeded for study: "
                      << study_uid_list[i] << " in "
              << intervals[storedStudyCount].count() << "s");
            storedStudyCount++;
                }
                else {
                    DCMNET_INFO("Responses from peer indicate that store"
                      << " operation failed for study: " << study_uid_list[i]);
                }
            }
            else {
                DCMNET_INFO("Store request failed to issue successfully for
    study: "
                        << study_uid_list[i]);
            }

        }
        else {
            DCMNET_ERROR("Directory for study " << study_uid_list[i]
              << " does not appear to exist");
        }
    }


    // Release association
    scu.closeAssociation(DCMSCU_RELEASE_ASSOCIATION);

    // Process results
    std::ofstream results_file ("sndspdtest_results.txt");
    OFListIterator(OFString) studyUID = storedStudyUIDs.begin();
    double sum = 0.0;
    for(int i = 0; i < storedStudyCount; i++) {
        results_file << timestamps[i] << ", ";
        results_file << *(studyUID) << ", ";
        results_file << intervals[i].count() << std::endl;
        sum += intervals[i].count();
        studyUID++;
    }

    results_file.close();

    std::cout << "Average time per request based on " << storedStudyCount;
    std::cout << " C-STORE requests: " << sum/storedStudyCount << "s" << std::
    endl;

    return 0;

```

```
}
```

Listing B.1: Application to measure the time taken to insert DICOM studies in a PACS

B.1.2 Retrieving DICOM Data from the PACS

The C++ application implementing the C-MOVE SCU for retrieving DICOM series from a PACS, shown in Figure 4.4 on page 75, is given in Listing B.2. Further details of this code and its application can also be found in [148].

```
#include "dcmtk/config/osconfig.h"    // include OS specific configuration
#include "dcmtk/dcmnet/diutil.h"
#include "dcmtk/dcmnet/scu.h"

#include <chrono>    // C++11 for high_resolution_clock and random number seed

#include <iostream>
#include <fstream>

#include "series_list.h" // File containing list of series to transfer

#define NUM_TESTS 50    // number of series to retrieve

// Create a logger for this application
static OFLogger appLogger = OFLog::getLogger("dcmtk.apps.rcvspdtest");

/*
 * Find a negotiated presentation context for the given SOP class using one of
 * the listed transfer syntaxes
 */
static Uint8 findPresContext(const OFString& sopClass, DcmSCU& scu,
        OFList<OFString>& ts) {

    Uint8 pc;

    /*
     * Search from the front of the list of transfer syntaxes given in ts
     * argument for one that has been accepted by the peer application
     */
    while (!ts.empty()) {

        pc = scu.findPresentationContextID(sopClass, ts.front());

        if (pc != 0)
            return pc;

        ts.pop_front();
    }

    /*
     * No presentation contexts with any of the listed transfer syntaxes have
     * been accepted by the peer application for the SOP class
     */
    return 0;

}

int main(int argc, char *argv[]) {

```

```

    OFLog::configure(OFLogger::OFF_LOG_LEVEL);

    DcmSCU scu;
    scu.setAETitle("RCV-SPD-TEST");
    scu.setPeerHostName("192.168.1.103");
    scu.setPeerPort(104);
    scu.setPeerAETitle("ORTHANC");

    // Create list of all uncompressed transfer syntaxes
    OFList<OFString> ts;
    ts.push_back(UID_LittleEndianExplicitTransferSyntax);
    ts.push_back(UID_BigEndianExplicitTransferSyntax);
    ts.push_back(UID_LittleEndianImplicitTransferSyntax);

    /*
     * Propose 2 presentation contexts to peer, one each for the C-ECHO and
     * C-MOVE services, both proposing the transfer syntaxes listed previously.
     */
    scu.addPresentationContext(UID_MOVEStudyRootQueryRetrieveInformationModel,
    ts);
    scu.addPresentationContext(UID_VerificationSOPClass, ts);

    // Initialize network
    OFCondition result = scu.initNetwork();
    if (result.bad()) {
        DCMNET_ERROR("Unable to set up the network: " << result.text());
        return -1;
    }

    // Negotiate Association
    result = scu.negotiateAssociation();
    if (result.bad()) {
        DCMNET_ERROR("Unable to negotiate association: " << result.text());
        return -1;
    }


    // Test connection to peer with C-ECHO request
    result = scu.sendECHORequest(0);
    if (result.bad()) {
        DCMNET_ERROR("Could not process C-ECHO with the server: " << result.text
    ());
        return -1;
    }


    /*
     * Find a presentation context for the C-MOVE service that has been
     * accepted by the peer
     */
    T_ASC_PresentationContextID pcID =
      findPresContext(UID_MOVEStudyRootQueryRetrieveInformationModel, scu, ts);

    if (pcID == 0) {
        DCMNET_ERROR("There is no accepted presentation context for Study Root
    MOVE");
        return -1;
    }

    DcmDataset qry;
    Uint32 movedSeriesCount = 0;
    OFList<RetrieveResponse*> moveResponses;
    OFList<OFString> movedSeriesUIDs;
```

```
111
112     // https://www.pluralsight.com/blog/software-development/how-to-measure-
    execution-time-intervals-in-c--
113     /*
114      * Create objects to measure and record time taken for each C-MOVE request
115      * to complete:
116      *   - two time_point objects using the high_resolution_clock time_point
117      *     type to record the time at the start and finish of each C-MOVE
118      *     request
119      *   - an array of duration objects to hold the durations of each C-MOVE
120      *     request
121      */
122     std::chrono::time_point<std::chrono::high_resolution_clock> start, finish;
123     std::chrono::duration<double> intervals[NUM_TESTS];
124     long int timestamps[NUM_TESTS];
125
126     // Every loop run sends a C-MOVE request for a series
127     for (int i = 0; i < NUM_TESTS; i++) {
128
129         // Add series instance UID to the DICOM dataset to request the C-MOVE
130         qry.putAndInsertOFStringArray(DCM_SeriesInstanceUID, series_uid_list[i])
    ;
131
132         /*
133          * Issue the C-MOVE request, specifying the AET of the C-STORE SCP to
134          * be used by the peer to return the series to
135          */
136         start = std::chrono::high_resolution_clock::now();
137         result = scu.sendMOVERequest(pcID, "STORESCP", &qry, &moveResponses);
138         finish = std::chrono::high_resolution_clock::now();
139
140         // Test whether C-MOVE request succeeded
141         if (result.good() && ((moveResponses.back())->m_status == STATUS_Success
    )) {
142             intervals[movedSeriesCount] = finish - start;
143             timestamps[movedSeriesCount] =
144               std::chrono::duration_cast<std::chrono::seconds>
145               (start.time_since_epoch()).count();
146             movedSeriesUIDs.push_back(series_uid_list[i]);
147             DCMNET_INFO("Move request succeeded for series: " << series_uid_list
    [i]
148               << " in " << intervals[movedSeriesCount].count() << "s");
149             movedSeriesCount++;
150         }
151         else
152             DCMNET_INFO("Move request failed for series: " << series_uid_list[i
    ]);
153
154         // Clear moveResponses list ready for next C-MOVE request
155         moveResponses.clear();
156
157     }
158
159
160     // Release association
161     scu.closeAssociation(DCMSCU_RELEASE_ASSOCIATION);
162
163     // Process results
164     std::ofstream results_file ("rcvspdtest_results.txt");
165     OFListIterator(OFString) seriesUID = movedSeriesUIDs.begin();
166     double sum = 0.0;
167     for(Uint32 ii = 0; ii < movedSeriesCount; ii++) {
168         results_file << timestamps[ii] << ", ";
169         results_file << *(seriesUID) << ", ";
```

```
170         results_file << intervals[ii].count() << std::endl;
171         sum += intervals[ii].count();
172         seriesUID++;
173     }
174 
175     results_file.close();
176 
177     std::cout << "Average time per request based on " << movedSeriesCount;
178     std::cout << " C-MOVE requests: " << sum/movedSeriesCount << "s" << std::
    endl;
179 
180     return 0;
181 
182 }
```

Listing B.2: Application to measure the time taken to retrieve DICOM series from a PACS

B.2 Bone Segmentation Algorithms

B.2.1 Haas Algorithm

Listing B.3 shows the C++ application implementing the original segmentation algorithm proposed by Haas *et al.* [5] and described in detail in Sections 5.1 and 5.2.1, starting on page 103. This application makes use of the OpenCV libraries [128]. Further code associated with this application can be obtained from [149] and [151].

```
1  #include "binvol.h"
2  #include <opencv2/opencv.hpp>
3  
4  #define HU_OFFSET 1024
5  #define BODY_CC_MIN_AREA 800    // Minimum area of connected components in body
       mask in mm^2
6  #define RECT_WIDTH 340          // Width of centralised rectangle for body mask
       components in mm
7  #define RECT_HEIGHT 170         // Height of centralised rectangle for body mask
        components in mm
8  #define BONE_CC_MIN_AREA 25     // Minimum area of connected components in bone
       mask in mm^2
9  
10 void segmentBoneSlice( BinVolume *vol, unsigned short slice_idx,
11                         short lo_thold, short hi_thold, int min_cc_area,
12                         int rect_bound_top, int rect_bound_bottom,
13                         int rect_bound_left, int rect_bound_right,
14                         short lo_bone_thold, short hi_bone_thold,
15                         int min_bone_area ) {
16 
17     // Get pixel data for slice and convert to OpenCV Mat
18     unsigned short orig_pix[vol->nrows * vol-> ncols];
19     vol->getSlice(slice_idx, orig_pix);
20     cv::Mat orig_img = cv::Mat(vol->nrows, vol->ncols, CV_16U,
21                                (void *) orig_pix);
22 
23     // Apply thresholds
24     cv::Mat lo_pixels = cv::Mat(orig_img.size(), CV_16U);
25     cv::Mat hi_pixels = cv::Mat(orig_img.size(), CV_16U);
26     cv::threshold(orig_img, lo_pixels, (double) hi_thold + HU_OFFSET, 255,
```

```
                cv::THRESH_BINARY_INV);
    lo_pixels.convertTo(lo_pixels, CV_8U);
    cv::threshold(orig_img, hi_pixels, (double) lo_thold + HU_OFFSET, 255,
                cv::THRESH_BINARY);
    hi_pixels.convertTo(hi_pixels, CV_8U);
    cv::Mat body_mask = cv::Mat(orig_img.size(), CV_8U);
    cv::bitwise_and(lo_pixels, hi_pixels, body_mask);

    // Morphological opening
    cv::Mat strel = cv::getStructuringElement(cv::MORPH_CROSS, cv::Size(3, 3));
    cv::morphologyEx(body_mask, body_mask, cv::MORPH_OPEN, strel,
                     cv::Point(-1, -1), 2, cv::BORDER_REPLICATE);

    // Find connected components
    cv::Mat labelled_image = cv::Mat(orig_img.size(), CV_32S);
    cv::Mat cc_stats = cv::Mat(orig_img.size(), CV_32S);
    cv::Mat centroids = cv::Mat(orig_img.size(), CV_64F);
    int cc_count = cv::connectedComponentsWithStats(body_mask, labelled_image,
        cc_stats, centroids, 4, CV_32S, cv::CCL_DEFAULT);

    // Discard connected components that are too small or do not overlap a
    // centralised rectangle
    bool discard_label[cc_count];
    for(int i = 1; i < cc_count; i++){ // Indexed from 1, 0 is background label
        if (cc_stats.at<int>(i, cv::CC_STAT_AREA) < min_cc_area ||
            cc_stats.at<int>(i, cv::CC_STAT_LEFT) > rect_bound_right ||
            cc_stats.at<int>(i, cv::CC_STAT_TOP) > rect_bound_bottom ||
            cc_stats.at<int>(i, cv::CC_STAT_LEFT) + cc_stats.at<int>(i, cv::
    CC_STAT_WIDTH) < rect_bound_left ||
            cc_stats.at<int>(i, cv::CC_STAT_TOP) + cc_stats.at<int>(i, cv::
    CC_STAT_HEIGHT) < rect_bound_top)
            discard_label[i] = true;
        else
            discard_label[i] = false;
    }

    for(int i = 0; i < orig_img.rows; i++) {
        for(int j = 0; j < orig_img.cols; j++) {

            int label = labelled_image.at<int>(i, j);

            if(label != 0){
                if(discard_label[label]) {
                    unsigned char *mask_pixel = body_mask.ptr<unsigned char>(i,
    j);
                    *mask_pixel = 0;
                }
            }
        }
    }

    // Apply body mask to original image
    cv::Mat body_img = cv::Mat(orig_img.size(), CV_16U);
    orig_img.copyTo(body_img, body_mask);

    // Apply thresholds
    cv::Mat lo_bone_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::Mat hi_bone_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::threshold(body_img, lo_bone_pixels, (double) hi_bone_thold + HU_OFFSET,
    255,
                cv::THRESH_BINARY_INV);
    lo_bone_pixels.convertTo(lo_bone_pixels, CV_8U);
    cv::threshold(body_img, hi_bone_pixels, (double) lo_bone_thold + HU_OFFSET,
    255,
```

```
                    cv::THRESH_BINARY);
    hi_bone_pixels.convertTo(hi_bone_pixels, CV_8U);
    cv::Mat bone_mask = cv::Mat(orig_img.size(), CV_8U);
    cv::bitwise_and(lo_bone_pixels, hi_bone_pixels, bone_mask);

    // Discard components that are unlikely to be bone
    cc_count = cv::connectedComponentsWithStats(bone_mask, labelled_image,
        cc_stats, centroids, 4, CV_32S, cv::CCL_DEFAULT);

    bool discard_bone_label[cc_count];
    for(int i = 1; i < cc_count; i++){ // Indexed from 1, 0 is background label
        if (cc_stats.at<int>(i, cv::CC_STAT_AREA) < min_bone_area ||
            cc_stats.at<int>(i, cv::CC_STAT_WIDTH) / cc_stats.at<int>(i, cv::
    CC_STAT_HEIGHT) > 6 ||
            cc_stats.at<int>(i, cv::CC_STAT_HEIGHT) / cc_stats.at<int>(i, cv::
    CC_STAT_WIDTH) > 6)
            discard_bone_label[i] = true;
        else
            discard_bone_label[i] = false;
    }

    for(int i = 0; i < orig_img.rows; i++) { // Look in to replacing with
    forEach
        for(int j = 0; j < orig_img.cols; j++) {

            int label = labelled_image.at<int>(i, j);

            if(label != 0){
                if(discard_bone_label[label]) {
                    unsigned char *mask_pixel = bone_mask.ptr<unsigned char>(i,
    j);
                    *mask_pixel = 0;
                }
            }
        }
    }

    return;

}


int main( int argc, char *argv[] ) {

    // Create BinVolume object
    std::string fname = std::string("testvolume.bin");
    if( argc >= 2 )
        fname = std::string(argv[1]);
    BinVolume vol = BinVolume(fname.c_str());

    // Find physical area of pixels in image slice in mm^2
    float pix_area = vol.pixel_height * vol.pixel_width;

    // Choose slice to perform operation on
    unsigned short slice_idx;
    if( argc >= 3 )
        slice_idx = (unsigned short) strtoul(argv[2], NULL, 0);
    else
        slice_idx = 1;

    // Choose thresholds to apply to image
    short lo_thold, hi_thold;
    if( argc >= 4 )
        lo_thold = (short) strtol(argv[3], NULL, 0);
```

```
146     else
147         lo_thold = -175;
148     if( argc >= 5 )
149         hi_thold = (short) strtol(argv[4], NULL, 0);
150     else
151         hi_thold = 1250;
152 
153     // Set minimum number of pixels for body connected components
154     int min_cc_area = (int)(BODY_CC_MIN_AREA / pix_area);
155 
156     // Set bounds of central rectangle with which components must overlap
157     float rect_height = RECT_HEIGHT / vol.pixel_height;
158     float rect_width = RECT_WIDTH / vol.pixel_width;
159 
160     int rect_bound_top = (int)((vol.nrows - rect_height) / 2);
161     int rect_bound_bottom = (int)(rect_bound_top + rect_height);
162 
163     int rect_bound_left = (int)((vol.ncols - rect_width) / 2);
164     int rect_bound_right = (int)(rect_bound_left + rect_width);
165 
166     // Choose thresholds for bone
167     short lo_bone_thold, hi_bone_thold;
168     if( argc >= 6 )
169         lo_bone_thold = (short) strtol(argv[5], NULL, 0);
170     else
171         lo_bone_thold = 145;
172     if( argc >= 7 )
173         hi_bone_thold = (short) strtol(argv[6], NULL, 0);
174     else
175         hi_bone_thold = 1500;
176 
177     // Set minimum number of pixels for bone connected components
178     int min_bone_area = (int)(BONE_CC_MIN_AREA / pix_area);
179 
180     // Call segmentBoneSlice function
181     segmentBoneSlice(&vol, slice_idx, lo_thold, hi_thold, min_cc_area,
182                      rect_bound_top, rect_bound_bottom, rect_bound_left,
183                      rect_bound_right, lo_bone_thold, hi_bone_thold,
184                      min_bone_area);
```

Listing B.3: Application applying the Haas algorithm to segment bony anatomy from a CT slice

B.2.2 Simplified Haas Algorithm

The software implementation of the Simplified Haas algorithm described in Section 5.2.1, on page 111, is shown in Listing B.4. All of the code and data associated with this application can be found in [149] and [151].

```
1  #include "binvol.h"
2  #include <opencv2/opencv.hpp>
3  #include <opencv2/core/utils/trace.hpp> // required for trace macros
4  #include <chrono> // C++11 library for timing performance
5  
6  #define HU_OFFSET 1024
7  
8  void segmentBoneSlice( cv::Mat const& orig_img, cv::Mat &bone_mask,
9                         short lo_thold, short hi_thold, short lo_bone_thold,
10                        short hi_bone_thold ) {
11 
```

```
#ifdef __TRACE__
CV_TRACE_FUNCTION();
CV_TRACE_REGION("BodyMaskThreshold");
#endif


    // Apply thresholds
    cv::Mat lo_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::Mat hi_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::threshold(orig_img, lo_pixels, (double) hi_thold + HU_OFFSET, 255,
                  cv::THRESH_BINARY_INV);
    lo_pixels.convertTo(lo_pixels, CV_8U);
    cv::threshold(orig_img, hi_pixels, (double) lo_thold + HU_OFFSET, 255,
                  cv::THRESH_BINARY);
    hi_pixels.convertTo(hi_pixels, CV_8U);
    cv::Mat body_mask = cv::Mat(orig_img.size(), CV_8U);
    cv::bitwise_and(lo_pixels, hi_pixels, body_mask);


#ifdef __TRACE__
CV_TRACE_REGION_NEXT("BodyMaskMorphologicalFilter");
#endif

    // Morphological opening
    cv::Mat strel = cv::getStructuringElement(cv::MORPH_CROSS, cv::Size(3, 3));
    cv::morphologyEx(body_mask, body_mask, cv::MORPH_OPEN, strel,
                     cv::Point(-1, -1), 2, cv::BORDER_REPLICATE);


#ifdef __TRACE__
CV_TRACE_REGION_NEXT("BodyMaskApplication");
#endif

    // Apply body mask to original image
    cv::Mat body_img = cv::Mat(orig_img.size(), CV_16U);
    orig_img.copyTo(body_img, body_mask);

#ifdef __TRACE__
CV_TRACE_REGION_NEXT("BoneMaskThreshold");
#endif

    // Apply thresholds
    cv::Mat lo_bone_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::Mat hi_bone_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::threshold(body_img, lo_bone_pixels, (double) hi_bone_thold + HU_OFFSET,
    255,
                  cv::THRESH_BINARY_INV);
    lo_bone_pixels.convertTo(lo_bone_pixels, CV_8U);
    cv::threshold(body_img, hi_bone_pixels, (double) lo_bone_thold + HU_OFFSET,
    255,
                  cv::THRESH_BINARY);
    hi_bone_pixels.convertTo(hi_bone_pixels, CV_8U);
    cv::bitwise_and(lo_bone_pixels, hi_bone_pixels, bone_mask);

    return;

}


int main( int argc, char *argv[] ) {

    // Create BinVolume object
    std::string fname = std::string("testvolume.bin");
    if( argc >= 2 )
```

```
74          fname = std::string(argv[1]);
75      BinVolume vol = BinVolume(fname.c_str());
76
77      // Choose thresholds to apply to image
78      short lo_thold, hi_thold;
79      if( argc >= 3 )
80          lo_thold = (short) strtol(argv[2], NULL, 0);
81      else
82          lo_thold = -175;
83      if( argc >= 4 )
84          hi_thold = (short) strtol(argv[3], NULL, 0);
85      else
86          hi_thold = 1250;
87
88      // Choose thresholds for bone
89      short lo_bone_thold, hi_bone_thold;
90      if( argc >= 5 )
91          lo_bone_thold = (short) strtol(argv[4], NULL, 0);
92      else
93          lo_bone_thold = 145;
94      if( argc >= 6 )
95          hi_bone_thold = (short) strtol(argv[5], NULL, 0);
96      else
97          hi_bone_thold = 1500;
98
99      // Initialise a BinVolume file to write bone mask pixel data to
100     BinVolume bone_vol("bone_volume.bin", vol.nrows, vol.ncols, vol.nslices,
101                        vol.pixel_height, vol.pixel_width);
102
103 #ifdef __TIME_SLICE__
104
105     /*
106      * Initialise a file and variables for recording the time for each call to
107      * segmentBoneSlice function
108      */
109
110     std::chrono::time_point<std::chrono::high_resolution_clock> slice_start,
        slice_finish;
111     std::chrono::duration<double> slice_interval;
112
113 #endif
114
115 #ifdef __TIME_VOLUME__
116
117     /*
118      * Initialise a file and variables for recording the time to process entire
        volume
119      */
120
121     std::chrono::time_point<std::chrono::high_resolution_clock> volume_start,
        volume_finish;
122     std::chrono::duration<double> volume_interval;
123
124     volume_start = std::chrono::high_resolution_clock::now();
125
126 #endif
127
128     // Call segmentBoneSlice function for each slice of volume
129     for(unsigned short slice_idx = 0; slice_idx < vol.nslices; slice_idx++) {
130
131         // Get pixel data for slice and convert to OpenCV Mat
132         unsigned short orig_pix[vol.nrows * vol.ncols];
133         vol.getSlice(slice_idx, orig_pix);
134         cv::Mat orig_img(vol.nrows, vol.ncols, CV_16U, (void *) orig_pix);
```

```
135         cv::Mat bone_mask(orig_img.size(), CV_8U);
136 
137 #ifdef __TIME_SLICE__
138 
139         slice_start = std::chrono::high_resolution_clock::now();
140 
141 #endif
142 
143         segmentBoneSlice(orig_img, bone_mask, lo_thold, hi_thold, lo_bone_thold,
144                          hi_bone_thold);
145 
146 #ifdef __TIME_SLICE__
147 
148         slice_finish = std::chrono::high_resolution_clock::now();
149         slice_interval = slice_finish - slice_start;
150 
151         std::cout << vol.filename << ";" << slice_idx << ";"
152                 << slice_interval.count() << std::endl;
153 
154 #endif
155 
156         // Write pixel data for bone mask slice to output file
157         std::vector<unsigned char> slice_pix;
158         if(bone_mask.isContinuous())
159             slice_pix.assign(bone_mask.datastart, bone_mask.dataend);
160         else {
161             for(int i = 0; i < bone_mask.rows; i++) {
162                 slice_pix.insert(slice_pix.end(),
163                                  bone_mask.ptr<unsigned char>(i),
164                                  bone_mask.ptr<unsigned char>(i) +
165                                  bone_mask.cols);
166             }
167         }
168 
169         bone_vol.appendSlice(slice_pix);
170 
171     }
172 
173 #ifdef __TIME_VOLUME__
174 
175     volume_finish = std::chrono::high_resolution_clock::now();
176     volume_interval = volume_finish - volume_start;
177 
178     std::cout << vol.filename << ";" << volume_interval.count() << std::endl;
179 
180 #endif
181 
182 
183     return 0;
184 
185 }
```

Listing B.4: Application applying the Simplified Haas algorithm to segment bony anatomy from a region of interest in a CT slice

B.2.3 Noise-reduced Algorithm

Listing B.5 shows the C++ software implementation of the Noise-reduced algorithm illustrated in Figure 6.10 on page 166. Further code and data for this application can be obtained from [151].

```
#include "binvol.h"
#include <opencv2/opencv.hpp>
#include <opencv2/core/utils/trace.hpp> // required for trace macros
#include <chrono> // C++11 library for timing performance

#define HU_OFFSET 1024

void segmentBoneSlice( cv::Mat const& orig_img, cv::Mat &bone_mask,
                       short lo_thold, short hi_thold, short lo_bone_thold,
                       short hi_bone_thold ) {

#ifdef __TRACE__
CV_TRACE_FUNCTION();
CV_TRACE_REGION("BodyMaskThreshold");
#endif


    // Apply thresholds
    cv::Mat lo_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::Mat hi_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::threshold(orig_img, lo_pixels, (double) hi_thold + HU_OFFSET, 255,
                  cv::THRESH_BINARY_INV);
    lo_pixels.convertTo(lo_pixels, CV_8U);
    cv::threshold(orig_img, hi_pixels, (double) lo_thold + HU_OFFSET, 255,
                  cv::THRESH_BINARY);
    hi_pixels.convertTo(hi_pixels, CV_8U);
    cv::Mat body_mask = cv::Mat(orig_img.size(), CV_8U);
    cv::bitwise_and(lo_pixels, hi_pixels, body_mask);


#ifdef __TRACE__
CV_TRACE_REGION_NEXT("BodyMaskMorphologicalFilter");
#endif

    // Morphological opening
    cv::Mat strel = cv::getStructuringElement(cv::MORPH_CROSS, cv::Size(3, 3));
    cv::morphologyEx(body_mask, body_mask, cv::MORPH_OPEN, strel,
                     cv::Point(-1, -1), 2, cv::BORDER_REPLICATE);


#ifdef __TRACE__
CV_TRACE_REGION_NEXT("BodyMaskApplication");
#endif

    // Apply body mask to original image
    cv::Mat body_img = cv::Mat(orig_img.size(), CV_16U);
    orig_img.copyTo(body_img, body_mask);

#ifdef __TRACE__
CV_TRACE_REGION_NEXT("BoneMaskThreshold");
#endif

    // Apply thresholds
    cv::Mat lo_bone_pixels = cv::Mat(orig_img.size(), CV_16U);
    cv::Mat hi_bone_pixels = cv::Mat(orig_img.size(), CV_16U);
```

```
56      cv::threshold(body_img, lo_bone_pixels, (double) hi_bone_thold + HU_OFFSET,
        255,
57                    cv::THRESH_BINARY_INV);
58      lo_bone_pixels.convertTo(lo_bone_pixels, CV_8U);
59      cv::threshold(body_img, hi_bone_pixels, (double) lo_bone_thold + HU_OFFSET,
        255,
60                    cv::THRESH_BINARY);
61      hi_bone_pixels.convertTo(hi_bone_pixels, CV_8U);
62      cv::bitwise_and(lo_bone_pixels, hi_bone_pixels, bone_mask);
63
64  #ifdef __TRACE__
65  CV_TRACE_REGION_NEXT("BoneMaskMorphologicalFilter");
66  #endif
67
68      /*
69       * Second morphological filtering stage
70       * This stage was specifically added to reduce the amount of noise
71       * typically found in 4DCT scans from being incorrectly segmented as bone
72       */
73      cv::morphologyEx(bone_mask, bone_mask, cv::MORPH_OPEN, strel,
74                       cv::Point(-1, -1), 2, cv::BORDER_REPLICATE);
75
76      return;
77
78  }
79
80
81  int main( int argc, char *argv[] ) {
82
83      // Create BinVolume object
84      std::string fname = std::string("testvolume.bin");
85      if( argc >= 2 )
86          fname = std::string(argv[1]);
87      BinVolume vol = BinVolume(fname.c_str());
88
89      // Choose thresholds to apply to image
90      short lo_thold, hi_thold;
91      if( argc >= 3 )
92          lo_thold = (short) strtol(argv[2], NULL, 0);
93      else
94          lo_thold = -175;
95      if( argc >= 4 )
96          hi_thold = (short) strtol(argv[3], NULL, 0);
97      else
98          hi_thold = 1250;
99
100     // Choose thresholds for bone
101     short lo_bone_thold, hi_bone_thold;
102     if( argc >= 5 )
103         lo_bone_thold = (short) strtol(argv[4], NULL, 0);
104     else
105         lo_bone_thold = 145;
106     if( argc >= 6 )
107         hi_bone_thold = (short) strtol(argv[5], NULL, 0);
108     else
109         hi_bone_thold = 1500;
110
111     // Initialise a BinVolume file to write bone mask pixel data to
112     BinVolume bone_vol("bone_volume.bin", vol.nrows, vol.ncols, vol.nslices,
113                        vol.pixel_height, vol.pixel_width);
114
115 #ifdef __TIME_SLICE__
116
117     /*
```

```
118      * Initialise variables for recording the time for each call to the
119      * segmentBoneSlice function
120      */
121
122     std::chrono::time_point<std::chrono::high_resolution_clock> slice_start,
         slice_finish;
123     std::chrono::duration<double> slice_interval;
124
125 #endif
126
127     // Call segmentBoneSlice function for each slice of volume
128     for(unsigned short slice_idx = 0; slice_idx < vol.nslices; slice_idx++) {
129
130         // Get pixel data for slice and convert to OpenCV Mat
131         unsigned short orig_pix[vol.nrows * vol.ncols];
132         vol.getSlice(slice_idx, orig_pix);
133         cv::Mat orig_img(vol.nrows, vol.ncols, CV_16U, (void *) orig_pix);
134         cv::Mat bone_mask(orig_img.size(), CV_8U);
135
136 #ifdef __TIME_SLICE__
137
138         slice_start = std::chrono::high_resolution_clock::now();
139
140 #endif
141
142         segmentBoneSlice(orig_img, bone_mask, lo_thold, hi_thold, lo_bone_thold,
143                          hi_bone_thold);
144
145 #ifdef __TIME_SLICE__
146
147         slice_finish = std::chrono::high_resolution_clock::now();
148         slice_interval = slice_finish - slice_start;
149
150         std::cout << vol.filename << ";" << slice_idx << ";"
151                 << slice_interval.count() << std::endl;
152
153 #endif
154
155         // Write pixel data for bone mask slice to output file
156         std::vector<unsigned char> slice_pix;
157         if(bone_mask.isContinuous())
158             slice_pix.assign(bone_mask.datastart, bone_mask.dataend);
159         else {
160             for(int i = 0; i < bone_mask.rows; i++) {
161                 slice_pix.insert(slice_pix.end(),
162                                  bone_mask.ptr<unsigned char>(i),
163                                  bone_mask.ptr<unsigned char>(i) +
164                                  bone_mask.cols);
165             }
166         }
167
168         bone_vol.appendSlice(slice_pix);
169
170     }
171
172
173     return 0;
174
175 }
```

Listing B.5: Application applying the Noise-reduced algorithm to segment bony anatomy from a region of interest in a 4DCT slice

B.2.4 Otsu's Method

The software implementation of Otsu's method described in Section 6.2.1, starting on page 143, and as applied in the 4DCT Optimal Threshold algorithm, is shown in Listing B.6. This implementation uses an un-normalised histogram and recursive operations to compute the between-class variance. Further code and data associated with this application can be obtained from [150].

```
#include <stdint.h>
#include <stdio.h>

#define GREY_LEVELS 2645

/* values contained in this array should be in interval [0, 1] */
unsigned hist[GREY_LEVELS];
/* value to offset array index to account for negative pixel values */
uint16_t index_offset = 1000;
/* array to hold set of optimal thresholds */
uint32_t opt_tholds[2];

/*****************************************************************************
 *
 * Function to create normalised histogram of image
 *
 * @param   Pointer to array of image pixels
 *
 * @param   Number of pixels in image
 *
 * @return  None
 *
 * @notes   None
 *
 *****************************************************************************/
void normaliseHistogram(int16_t *imagePtr, unsigned numPix) {

    unsigned i = 0;

    /*
     * Initialise hist array elements to zero
     */
    for(i = 0; i < GREY_LEVELS; i++) {
        hist[i] = 0;
    }

    /*
     * Iterate through image pixels incrementing the appropriate bin in
     * histogram
     */
    for (i = 0; i < numPix; i++) {
        hist[*imagePtr + index_offset]++;
        imagePtr++;
    }

}

/*****************************************************************************
 *
 * Function to compute optimal thresholds to segment image into 3 classes
 *
```

```
52 * @param   Pointer to normalised image histogram
53 *
54 * @return  None
55 *
56 * @notes   None
57 *
58 ****************************************************************************/
59 void computeThresholds(unsigned *hist) {
60
61    uint32_t i, j;
62    unsigned cumsum1, cumsum2, cumsum3;
63    unsigned long cummean1, cummean2, cummean3;
64    float mod_bcv;
65    unsigned long cum_mean[GREY_LEVELS];
66    unsigned cum_sum[GREY_LEVELS];
67    uint16_t max_thold1 = 0;
68    uint16_t max_thold2 = 1;
69    float max_mod_bcv = 0;
70
71    /*
72     * Iterate through cum_mean and cum_sum arrays populating each element with
73     * the cumulative mean and cumulative sum of the normalised histogram to
74     * that index.
75     */
76    cum_sum[0] = hist[0];
77    cum_mean[0] = 0;
78
79    for(i = 1; i < GREY_LEVELS; i++) {
80
81        cum_sum[i] = cum_sum[i-1] + hist[i];
82        cum_mean[i] = cum_mean[i-1] + (i * hist[i]);
83
84    }
85
86    /*
87     * Iterate through every combination of thresholds calculating the
88     * between class variance and find the thresholds that produce the
89     * maximum between class variance
90     */
91    for(i = 0; i < (GREY_LEVELS - 2); i++) {
92
93        for(j = (i + 1); j < (GREY_LEVELS - 1); j++) {
94
95            cumsum1 = cum_sum[i];
96            cumsum2 = cum_sum[j] - cum_sum[i];
97            cumsum3 = cum_sum[GREY_LEVELS - 1] - cum_sum[j];
98            cummean1 = cum_mean[i];
99            cummean2 = cum_mean[j] - cum_mean[i];
100           cummean3 = cum_mean[GREY_LEVELS - 1] - cum_mean[j];
101
102           /*
103            * If the cumulative sum for any of the classes is zero, this
104            * implies that no pixels belong to that class and therefore that
105            * is not a valid solution
106            */
107           if(!((cumsum1 == 0) || (cumsum2 == 0) || (cumsum3 == 0))) {
108               /*
109                * Compute modified between class variance
110                */
111               mod_bcv = ((float)cummean1 * (float)cummean1/(float)cumsum1)
112                           + ((float)cummean2 * (float)cummean2/(float)cumsum2)
113                           + ((float)cummean3 * (float)cummean3/(float)cumsum3)
114   ;
```

```
                /*
                 * Test if computed variance is greater than maximum previously
                 * computed between class variance
                 * If it is, record the new maximum value and record the set of
                 * thresholds for which it occurred
                 */
                if(mod_bcv > max_mod_bcv) {
                    max_mod_bcv = mod_bcv;
                    max_thold1 = i;
                    max_thold2 = j;
                }

            }
        }
    }

    opt_tholds[0] = max_thold1;
    opt_tholds[1] = max_thold2;

}

int main(){

    FILE *filePtr;
    unsigned numPix = 262144;
    int16_t pixels[numPix];

    /*
     * Obtain pixel values from binary filePtr
     */
    filePtr = fopen("image.bin", "rb");

    if(filePtr == NULL) {
        printf("Unable to open input file\n\r");
        return -1;
    }

    fread(pixels, 2, numPix, filePtr);
    fclose(filePtr);

    /*
     * Compute optimal thresholds
     */
    normaliseHistogram(pixels, numPix);
    computeThresholds(hist);

    printf("\n\rOptimal Thresholds: [%d, %d]\n\r",
            (opt_tholds[0] - index_offset), (opt_tholds[1] - index_offset));

    return 1;

}
```

Listing B.6: Implementation of Otsu's method used in the 4DCT Optimal Threshold algorithm

Bibliography

[1] C. K. Glide-Hurst and I. J. Chetty, "Improving radiotherapy planning, delivery accuracy, and normal tissue sparing using cutting edge technologies," *Journal of Thoracic Disease*, vol. 6, no. 4, pp. 303–318, 2014.

[2] B. F. O'Connell, D. M. Irvine, A. J. Cole, G. G. Hanna, and C. K. McGarry, "Optimizing geometric accuracy of four-dimensional CT scans acquired using the wall- and couch-mounted Varian® Real-time Position Management™ camera systems," *British Journal of Radiology*, vol. 88, no. 1046, 2015.

[3] A. Vestergaard, S. Hafeez, L. P. Muren, S. Nill, M. Høyer, V. N. Hansen, C. Grønborg, E. M. Pedersen, J. B. Petersen, R. Huddart, and U. Oelfke, "The potential of MRI-guided online adaptive re-optimisation in radiotherapy of urinary bladder cancer," *Radiotherapy and Oncology*, vol. 118, no. 1, pp. 154–159, Jan. 2016.

[4] E. E. Ahunbay, C. Peng, S. Holmes, A. Godley, C. Lawton, and X. A. Li, "Online Adaptive Replanning Method for Prostate Radiotherapy," *International Journal of Radiation Oncology • Biology • Physics*, vol. 77, no. 5, pp. 1561–1572, Aug. 2010.

[5] B. Haas, T. Coradi, M. Scholz, P. Kunz, M. Huber, U. Oppitz, L. André, V. Lengkeek, D. Huyskens, A. van Esch, and R. Reddick, "Automatic segmentation of thoracic and pelvic CT images for radiotherapy planning using implicit anatomic knowledge and organ-specific segmentation strategies," *Physics in Medicine and Biology*, vol. 53, no. 6, pp. 1751–1771, Mar. 2008.

[6] N. Otsu, "A threshold selection method from gray-level histogram," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[7] F. H. Martini, J. Nath, and E. F. Bartholomew, *Fundamentals of anatomy & physiology.* Harlow: Pearson, 2014.

[8] I. F. Tannock, R. P. Hill, R. G. Bristow, and L. Harrington, Eds., *The basic science of oncology*, 5th ed. New York: McGraw-Hill, 2013.

[9] F. Bray, J. Ferlay, I. Soerjomataram, R. L. Siegel, L. A. Torre, and A. Jemal, "Global cancer statistics 2018: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries," *CA: A Cancer Journal for Clinicians*, vol. 68, no. 6, pp. 394–424, 2018.

[10] J. Ferlay, M. Colombet, I. Soerjomataram, C. Mathers, D. Parkin, M. Pineros, A. Znaor, and F. Bray, "Estimating the global cancer incidence and mortality in 2018: Globocan sources and methods," *International Journal of Cancer*, vol. 144, no. 8, pp. 1940–1953, 2019.

[11] Cancer Research UK, "Cancer Statistics for the UK," accessed: 9th May 2020. [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics-for-the-uk

[12] B. Emami, J. Lyman, A. Brown, L. Cola, M. Goitein, J. Munzenrider, B. Shank, L. Solin, and M. Wesson, "Tolerance of normal tissue to therapeutic irradiation," *International Journal of Radiation Oncology • Biology • Physics*, vol. 21, no. 1, pp. 109–122, May 1991.

[13] N. Suntharalingam, E. Podgorsak, and J. Hendry, "Basic Radiobiology," in *Radiation Oncology Physics: A Handbook for Teachers and Students*, ii ed., E. Podgorsak, Ed. Vienna: International Atomic Energy Agency, 2005.

[14] E. Podgorsak, "External photon beams: physical aspects," in *Radiation Oncology Physics: A Handbook for Teachers and Students*, ii ed., E. Podgorsak, Ed. Vienna: International Atomic Energy Agency, 2005.

[15] ——, "Treatment machines for external beam radiotherapy," in *Radiation Oncology Physics: A Handbook for Teachers and Students*, ii ed., E. Podgorsak, Ed. Vienna: International Atomic Energy Agency, 2005.

[16] S. K. Agarwal, R. V. Scheele, and J. Wakley, "Physical Measurements Including Depth Dose Data and Isodose Curves for 8 MV Roentgen Rays," *Acta Radiologica: Therapy, Physics, Biology*, vol. 11, no. 1, pp. 97–105, Jan. 1972.

[17] K. J. Kim, J. Y. Lee, and K. R. Park, "Characteristics of 15 MV Photon Beam from a Varian Clinac 1800 Dual Energy Linear Accelerator," *J Korean Soc Ther Radiol*, vol. 9, no. 1, pp. 131–142, 1991.

[18] C. Fong, P. Sanghera, A. Hartley, J. Cashmore, D. Ford, and S. Green, "Proton radiotherapy: Important clinical and technical aspects for UK patients," *Imaging and Oncology*, pp. 6–13, 2017.

[19] P. Han, M. Rotman, A. R. Schulsinger, B. R. Pieters, C. C. E. Koning, F. J. Pos, M. C. C. M. Hulshof, and P. Poortmans, "Bladder Cancer," in *Technical Basis of Radiation Therapy*, S. H. Levitt, J. A. Purdy, C. A. Perez, and P. Poortmans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 801–827.

[20] J. M. Michalski and T. Wiegel, "Prostate," in *Technical Basis of Radiation Therapy*, S. H. Levitt, J. A. Purdy, C. A. Perez, and P. Poortmans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 949–1025.

[21] L. C. Cho, V. Fonteyne, W. DeNeve, S. S. Lo, and R. D. Timmerman, "Stereotactic Body Radiotherapy," in *Technical Basis of Radiation Therapy*, S. H. Levitt, J. A. Purdy, C. A. Perez, and P. Poortmans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 363–400.

[22] R. Li, P. Keall, and L. Xing, "Linac-Based Image Guided Intensity Modulated Radiation Therapy," in *Technical Basis of Radiation Therapy*, S. H. Levitt, J. A. Purdy, C. A. Perez, and P. Poortmans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 275–312.

[23] C. Ménard, U. Nestle, and D. Jaffray, "Imaging in Radiation Therapy," in *Technical Basis of Radiation Therapy*, S. H. Levitt, J. A. Purdy, C. A. Perez, and P. Poortmans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 63–83.

[24] J. A. Purdy, P. Poortmans, and C. A. Perez, "Three-Dimensional Treatment Planning and Conformal Therapy," in *Technical Basis of Radiation Therapy*, S. H. Levitt, J. A. Purdy, C. A. Perez, and P. Poortmans, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 253–273.

[25] T. J. Kinsella, J. Sohn, and B. Wessels, "Principles of Radiation Oncology," in *Oncology: An Evidence-Based Approach*, A. E. Chang, P. A. Ganz, D. F. Hayes, T. J. Kinsella, H. I. Pass, J. H. Schiller, R. M. Stone, and V. J. Strecher, Eds. New York: Springer, 2006, pp. 41–57.

[26] N. G. Burnet, "Defining the tumour and target volumes for radiotherapy," *Cancer Imaging*, vol. 4, no. 2, pp. 153–161, 2004.

[27] J. da Silva, R. Ansorge, and R. Jena, "Sub-second pencil beam dose calculation on GPU for adaptive proton therapy," *Physics in Medicine and Biology*, vol. 60, no. 12, pp. 4777–4795, Jun. 2015.

[28] P. J. Keall, D. T. Nguyen, R. O'Brien, V. Caillet, E. Hewson, P. R. Poulsen, R. Bromley, L. Bell, T. Eade, A. Kneebone, J. Martin, and J. T. Booth, "The first clinical implementation of real-time image-guided adaptive radiotherapy using a standard linear accelerator," *Radiotherapy and Oncology*, vol. 127, no. 1, pp. 6–11, Apr. 2018.

[29] F. Lagerwaard, A. Bruynzeel, S. Tetar, S. Oei, C. Haasbeek, B. Slotman, S. Senan, O. Bohoudi, and M. Palacios, "Stereotactic MR-Guided Adaptive Radiation Therapy (SMART) for Prostate Cancer," *International Journal of Radiation Oncology • Biology • Physics*, vol. 99, no. 2, pp. E681–E682, Oct. 2017.

[30] B. W. Raaymakers *et al.*, "First patients treated with a 1.5 T MRI-Linac: clinical proof of concept of a high-precision, high-field MRI guided radiotherapy treatment," *Physics in Medicine & Biology*, vol. 62, no. 23, pp. L41–L50, Nov. 2017.

[31] S. A. Mangar, E. Scurr, R. A. Huddart, S. A. Sohaib, A. Horwich, D. P. Dearnaley, and V. S. Khoo, "Assessing intra-fractional bladder motion using cine-MRI as initial methodology for Predictive Organ Localization (POLO) in radiotherapy for bladder cancer," *Radiotherapy and Oncology*, vol. 85, no. 2, pp. 207–214, Nov. 2007.

[32] K. M. Langen, T. R. Willoughby, S. L. Meeks, A. Santhanam, A. Cunningham, L. Levine, and P. A. Kupelian, "Observations on Real-Time Prostate Gland Motion Using Electromagnetic Tracking," *International Journal of Radiation Oncology • Biology • Physics*, vol. 71, no. 4, pp. 1084–1090, Jul. 2008.

[33] S. Acharya *et al.*, "Online Magnetic Resonance Image Guided Adaptive Radiation Therapy: First Clinical Applications," *International Journal of Radiation Oncology • Biology • Physics*, vol. 94, no. 2, pp. 394–403, Feb. 2016.

[34] J. Lamb, M. Cao, A. Kishan, N. Agazaryan, D. H. Thomas, N. Shaverdian, Y. Yang, S. Ray, D. A. Low, A. Raldow, M. L. Steinberg, and P. Lee, "Online Adaptive Radiation Therapy: Implementation of a New Process of Care," *Cureus*, Aug. 2017.

[35] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, vol. 21, pp. 977–1000, 2003.

[36] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in Medicine and Biology*, vol. 46, pp. R1–R45, 2001.

[37] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," *Medical Image Analysis*, vol. 2, no. 1, pp. 1–36, 1998.

[38] W. Plishker, O. Dandekar, S. Bhattacharyya, and R. Shekhar, "A Taxonomy for Medical Image Registration Acceleration Techniques," in *IEEE/NIH Life Science Systems and Applications Workshop (LISSA 2007)*. IEEE, 2007, pp. 160–163.

[39] Y. Wang, T. R. Mazur, J. C. Park, D. Yang, S. Mutic, and H. H. Li, "Development of a fast Monte Carlo dose calculation system for online adaptive radiation therapy quality assurance," *Physics in Medicine and Biology*, vol. 62, no. 12, pp. 4970–4990, Jun. 2017.

[40] J. Bertholet, G. Anastasi, D. Noble, A. Bel, R. van Leeuwen, T. Roggen, M. Duchateau, S. Pilskog, C. Garibaldi, N. Tilly, R. García-Mollá, J. Bonaque, U. Oelfke, M. C. Aznar, and B. Heijmen, "Patterns of practice for adaptive and real-time radiation therapy (POP-ART RT) part II: Offline and online plan adaption for interfractional changes," *Radiotherapy and Oncology*, vol. 153, pp. 88–96, Dec. 2020.

[41] O. L. Green, L. E. Henke, and G. D. Hugo, "Practical clinical workflows for online and offline adaptive radiation therapy," *Seminars in Radiation Oncology*, vol. 29, no. 3, pp. 219–227, Jul. 2019.

[42] C. Kontaxis, G. H. Bol, B. Stemkens, M. Glitzner, F. M. Prins, L. G. W. Kerkmeijer, J. J. W. Lagendijk, and B. W. Raaymakers, "Towards fast online intrafraction replanning for free-breathing stereotactic body radiation therapy with the MR-linac," *Physics in Medicine & Biology*, vol. 62, no. 18, pp. 7233–7248, Aug. 2017.

[43] L. P. Muren, A. T. Redpath, H. Lord, and D. McLaren, "Image-guided radiotherapy of bladder cancer: Bladder volume variation and its relation to margins," *Radiotherapy and Oncology*, vol. 84, no. 3, pp. 307–313, Sep. 2007.

[44] D. W. Litzenberg, J. M. Balter, S. W. Hadley, H. M. Sandler, T. R. Willoughby, P. A. Kupelian, and L. Levine, "Influence of intrafraction motion on margins for prostate radiotherapy," *International Journal of Radiation Oncology • Biology • Physics*, vol. 65, no. 2, pp. 548–553, Jun. 2006.

[45] Cancer Research UK, "Bladder cancer statistics," accessed: 9th May 2020. [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/bladder-cancer

[46] L. P. Muren, R. Smaaland, and O. Dahl, "Organ motion, set-up variation and treatment margins in radical radiotherapy of urinary bladder cancer," *Radiotherapy and Oncology*, vol. 69, no. 3, pp. 291–304, Dec. 2003.

[47] S. D. Collins and M. M. Leech, "A review of plan library approaches in adaptive radiotherapy of bladder cancer," *Acta Oncologica*, vol. 57, no. 5, pp. 566–573, 2018.

[48] Cancer Research UK, "A trial looking at different ways of giving radiotherapy for bladder cancer (RAIDER)," accessed: 4th January 2021. [Online]. Available: https://www.cancerresearchuk.org/about-cancer/find-a-clinical-trial/a-trial-looking-at-different-ways-of-giving-radiotherapy-for-bladder-cancer-raider

[49] ——, "Prostate cancer statistics," accessed: 9th May 2020. [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/prostate-cancer

[50] M. van Herk, A. Bruce, A. Guus Kroes, T. Shouman, A. Touw, and J. V. Lebesque, "Quantification of organ motion during conformal radiotherapy of the prostate by three dimensional image registration," *International Journal of Radiation Oncology • Biology • Physics*, vol. 33, no. 5, pp. 1311–1320, Dec. 1995.

[51] J. A. Antolak, I. I. Rosen, C. H. Childress, G. K. Zagars, and A. Pollack, "Prostate target volume variations during a course of radiotherapy," *International Journal of Radiation Oncology • Biology • Physics*, vol. 42, no. 3, pp. 661–672, Oct. 1998.

[52] A. Wong and S. Lou, "Medical Image Archive, Retrieval, and Communication," in *Handbook of medical image processing and analysis*, 2nd ed., I. N. Bankman, Ed. Amsterdam: Elsevier/Academic Press, 2009, pp. 861–873.

[53] D. P. Bovet and M. Cesati, *Understanding the Linux kernel.* Beijing ; Cambridge, Mass: O'Reilly, 2001.

[54] G. Frantz, "Digital signal processor trends," *IEEE micro*, vol. 20, no. 6, pp. 52–59, 2000.

[55] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating Compute-Intensive Applications with GPUs and FPGAs," in *2008 Symposium on Application Specific Processors.* Anaheim, CA, USA: IEEE, Jun. 2008, pp. 101–107.

[56] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund *et al.*, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 451–460.

[57] P. Richmond, "GPU computing: GPU architecture," accessed: 17th January 2021. [Online]. Available: http://paulrichmond.shef.ac.uk/teaching/NVIDIA/rabat/02%20-%20Architectures%20Overview.pdf

[58] S. Sirowy and A. Forin, "Where's the Beef? Why FPGAs Are So Fast," White Paper, Sep. 2008, accessed: 20th August 2018. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2008-130.pdf

[59] J. T. Teubner and L. Woods, *Data processing on FPGAs*, ser. Synthesis lectures on data management. San Rafael, Calif.: Morgan & Claypool Publ, 2013, no. 35.

[60] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA '12.* Monterey, California, USA: ACM Press, 2012, p. 47.

[61] Wei-Ning Huang, Sheng-Wei Cheng, C.-W. Chang, Yu-Chen Wu, T.-W. Kuo, Y.-C. Hsu, W.-Y. I. Tseng, and Shih-Hao Hung, "The acceleration of pipeline workloads under the FPGA area and bandwidth constraints," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. Chongqing, China: IEEE, Aug. 2014, pp. 1–9.

[62] G. Georgis, G. Lentaris, and D. Reisis, "Acceleration techniques and evaluation on multi-core CPU, GPU and FPGA for image processing and super-resolution," *Journal of Real-Time Image Processing*, vol. 16, no. 4, pp. 1207–1234, 2019.

[63] M. E. Angoletta, "Digital signal processor fundamentals and system design," 2008.

[64] J. Eyre, "The digital signal processor derby," *IEEE Spectrum*, vol. 38, no. 6, pp. 62–68, 2001.

[65] F. Zhang, Y. Gao, and J. D. Bakos, "Lucas-Kanade optical flow estimation on the TI C66x digital signal processor," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2014, pp. 1–6.

[66] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, Mar. 2015.

[67] D. G. Bailey, *Design for embedded image processing on FPGAs*. Singapore: John Wiley & Sons (Asia), 2011.

[68] X. Jia, P. Ziegenhein, and S. B. Jiang, "GPU-based high-performance computing for radiation therapy," *Physics in Medicine and Biology*, vol. 59, no. 4, pp. R151–R182, Feb. 2014.

[69] C. Gendrin, H. Furtado, C. Weber, C. Bloch, M. Figl, S. A. Pawiro, H. Bergmann, M. Stock, G. Fichtinger, D. Georg, and W. Birkfellner, "Monitoring tumor motion by real time 2d/3d registration during radiotherapy," *Radiotherapy and Oncology*, vol. 102, no. 2, pp. 274–280, Feb. 2012.

[70] J. Spoerk, C. Gendrin, C. Weber, M. Figl, S. A. Pawiro, H. Furtado, D. Fabri, C. Bloch, H. Bergmann, E. Gröller, and W. Birkfellner, "High-performance GPU-based rendering for real-time, rigid 2d/3d-image registration and motion prediction in radiation oncology," *Zeitschrift für Medizinische Physik*, vol. 22, no. 1, pp. 13–20, Feb. 2012.

[71] S. S. Samant, J. Xia, P. Muyan-Özçelik, and J. D. Owens, "High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy: Novel high performance computing for deformable image registration," *Medical Physics*, vol. 35, no. 8, pp. 3546–3553, Jul. 2008.

[72] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. Choi, E. Castillo, A. Majumdar, T. Guerrero, and S. B. Jiang, "Implementation and evaluation of various demons deformable image registration algorithms on a GPU," *Physics in Medicine and Biology*, vol. 55, no. 1, pp. 207–219, Jan. 2010.

[73] G. Yu, Y. Liang, G. Yang, H. Shu, B. Li, Y. Yin, and D. Li, "Accelerated gradient-based free form deformable registration for online adaptive radiotherapy," *Physics in Medicine and Biology*, vol. 60, no. 7, pp. 2765–2783, Apr. 2015.

[74] C. Men, X. Jia, and S. B. Jiang, "GPU-based ultra-fast direct aperture optimization for online adaptive radiation therapy," *Physics in Medicine and Biology*, vol. 55, no. 15, pp. 4309–4319, Aug. 2010.

[75] A. Hagan, A. Sawant, M. Folkerts, and A. Modiri, "Multi-GPU configuration of 4d intensity modulated radiation therapy inverse planning using global optimization," *Physics in Medicine & Biology*, vol. 63, no. 2, p. 025028, Jan. 2018.

[76] T. Jagt, S. Breedveld, R. van Haveren, B. Heijmen, and M. Hoogeman, "An automated planning strategy for near real-time adaptive proton therapy in prostate cancer," *Physics in Medicine & Biology*, vol. 63, no. 13, p. 135017, Jul. 2018.

[77] P. Ziegenhein, I. N. Kozin, C. P. Kamerling, and U. Oelfke, "Towards real-time photon Monte Carlo dose calculation in the cloud," *Physics in Medicine and Biology*, vol. 62, no. 11, pp. 4375–4389, Jun. 2017.

[78] X. Gu, D. Choi, C. Men, H. Pan, A. Majumdar, and S. B. Jiang, "GPU-based ultra-fast dose calculation using a finite pencil beam model," *Physics in Medicine and Biology*, vol. 54, no. 20, pp. 6287–6297, Oct. 2009.

[79] Y. Chi, Z. Tian, and X. Jia, "Modeling parameterized geometry in GPU-based Monte Carlo particle transport simulation for radiotherapy," *Physics in Medicine and Biology*, vol. 61, no. 15, pp. 5851–5867, Aug. 2016.

[80] R. W. Keyes, C. Romano, D. Arnold, and S. Luan, "Radiation therapy calculations using an on-demand virtual cluster via cloud computing," *ArXiv e-prints*, Sep. 2010.

[81] G. Pratx and L. Xing, "Monte Carlo simulation of photon migration in a cloud computing environment with MapReduce," *Journal of Biomedical Optics*, vol. 16, no. 12, p. 125003, 2011.

[82] C. M. Poole, I. Cornelius, J. V. Trapp, and C. M. Langton, "Radiotherapy Monte Carlo simulation using cloud computing technology," *Australasian Physical & Engineering Sciences in Medicine*, vol. 35, no. 4, pp. 497–502, Dec. 2012.

[83] K. Whitton, X. Hu, C. Yi, and D. Chen, "An FPGA Solution for Radiation Dose Calculation," in *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Napa, CA: IEEE, Apr. 2006, pp. 227–236.

[84] A. S. Pasciak and J. R. Ford, "High-speed evaluation of track-structure Monte Carlo electron transport simulations," *Physics in Medicine and Biology*, vol. 53, no. 19, pp. 5539–5553, Oct. 2008.

[85] J. Luu, K. Redmond, W. Lo, P. Chow, L. Lilge, and J. Rose, "FPGA-based Monte Carlo Computation of Light Absorption for Photodynamic Cancer Therapy," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. Napa, CA, USA: IEEE, 2009, pp. 157–164.

[86] B. Zhou, X. S. Hu, D. Z. Chen, and C. X. Yu, "A multi-FPGA accelerator for radiation dose calculation in cancer treatment," in *2009 IEEE 7th Symposium on Application Specific Processors*. San Francisco, CA, USA: IEEE, Jul. 2009, pp. 70–79.

[87] M. Gokhale and P. S. Graham, *Reconfigurable computing: accelerating computation with field-programmable gate arrays*. Dordrecht: Springer, 2005.

[88] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, R. W. Stewart, and University of Strathclyde, Eds., *The Zynq Book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*, 1st ed. Glasgow: Strathclyde Academic Media, 2014.

[89] Xilinx Inc., "Zynq-7000 SoC: Technical Reference Manual UG585, v1.12.2," Jul. 2018, accessed: 24th August 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[90] ——, "Zynq UltraScale+ MPSoC Data Sheet: Overview DS891 v1.5," Jul. 2017, accessed: 21st August 2018. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[91] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, and R. Stewart, *EXPLORING ZYNQ MPSOC: with pynq and machine learning applications.* Strathclyde Academic Media, 2019.

[92] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A quantitative analysis of the speedup factors of FPGAs over processors," in *Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays - FPGA '04*. Monterey, California, USA: ACM Press, 2004, p. 162.

[93] B. Cope, P. Cheung, W. Luk, and S. Witt, "Have GPUs made FPGAs redundant in the field of video processing?" in *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.* Singapore, China: IEEE, 2005, pp. 111–118.

[94] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *2009 International Conference on Field Programmable Logic and Applications*. Prague, Czech Republic: IEEE, Aug. 2009, pp. 126–131.

[95] L. Daoud, D. Zydek, and H. Selvaraj, "A Survey of High Level Synthesis Languages, Tools, and Compilers for Reconfigurable High Performance Computing," in *Advances in Systems Science*, J. Swiatek, A. Grzech, P. Swiatek, and J. M. Tomczak, Eds. Cham: Springer International Publishing, 2014, vol. 240, pp. 483–492.

[96] A. Agne, M. Happe, A. Keller, E. Lubbers, B. Plattner, M. Platzner, and C. Plessl, "ReconOS: An Operating System Approach for Reconfigurable Computing," *IEEE Micro*, vol. 34, no. 1, pp. 60–71, Jan. 2014.

[97] Xilinx Inc., "Zynq UltraScale+ Device: Technical Reference Manual UG1085, v1.8," Aug. 2018, accessed: 24th August 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf

[98] ——, "Zynq-7000 SoC Data Sheet: Overview DS190 v1.11.1," Jul. 2018, accessed: 21st August 2018. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[99] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, and D. Stroobandt, "An overview of today's high-level synthesis tools," *Design Automation for Embedded Systems*, vol. 16, no. 3, pp. 31–51, Sep. 2012.

[100] J. Cong, Bin Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Zhiru Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[101] P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "High-level synthesis of accelerators in embedded scalable platforms," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. Macao, Macao: IEEE, Jan. 2016, pp. 204–211.

[102] B. Draper, J. Beveridge, A. Bohm, C. Ross, and M. Chawathe, "Accelerated image processing on FPGAs," *IEEE Transactions on Image Processing*, vol. 12, no. 12, pp. 1543–1551, Dec. 2003.

[103] Takashi Saegusa, Tsutomu Maruyama, and Yoshiki Yamaguchi, "How fast is an FPGA in image processing?" in *2008 International Conference on Field Programmable Logic and Applications*. Heidelberg, Germany: IEEE, 2008, pp. 77–82.

[104] F. Grull and U. Kebschull, "Biomedical image processing and reconstruction with dataflow computing on FPGAs," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. Munich, Germany: IEEE, Sep. 2014, pp. 1–2.

[105] Jun Jiang, W. Luk, and D. Rueckert, "FPGA-based computation of free-form deformations in medical image registration," in *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No.03EX798)*. Tokyo, Japan: IEEE, 2003, pp. 234–241.

[106] O. Dandekar and R. Shekhar, "FPGA-Accelerated Deformable Image Registration for Improved Target-Delineation During CT-Guided Interventions," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 2, pp. 116–127, Jun. 2007.

[107] J. Koo, A. Evans, and W. Gross, "3-D Brain MRI Tissue Classification on FPGAs," *IEEE Transactions on Image Processing*, vol. 18, no. 12, pp. 2735–2746, Dec. 2009.

[108] I. Chiuchisan, "A new FPGA-based real-time configurable system for medical image processing," in *2013 E-Health and Bioengineering Conference (EHB)*. IASI, Romania: IEEE, Nov. 2013, pp. 1–4.

[109] H. Okuhata, R. Imai, M. Ise, R. Y. Omaki, H. Nakamura, S. Hara, and I. Shirakawa, "Implementation of dynamic-range enhancement and super-resolution algorithms for medical image processing," in *2014 IEEE International Conference on Consumer Electronics (ICCE)*. Las Vegas, NV, USA: IEEE, Jan. 2014, pp. 181–184.

[110] P. Li, T. Page, G. Luo, W. Zhang, P. Wang, P. Zhang, P. Maass, M. Jiang, and J. Cong, "FPGA Acceleration for Simultaneous Medical Image Reconstruction and Segmentation," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. Boston, MA, USA: IEEE, May 2014, pp. 172–172.

[111] T. Hussain, O. Palomar, A. Cristal, E. Ayguade, and A. Haider, "ViPS: Visual processing system for medical imaging," in *2015 8th International Conference on Biomedical Engineering and Informatics (BMEI)*. Shenyang, China: IEEE, Oct. 2015, pp. 40–45.

[112] K. Neshatpour, A. Koohi, F. Farahmand, R. Joshi, S. Rafatirad, A. Sasan, and H. Homayoun, "Big biomedical image processing hardware acceleration: A case study for K-means and image filtering," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. Montréal, QC, Canada: IEEE, May 2016, pp. 1134–1137.

[113] S. Afifi, H. GholamHosseini, and R. Sinha, "A low-cost FPGA-based SVM classifier for melanoma detection," in *2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*. Malaysia: IEEE, Dec. 2016, pp. 631–636.

[114] M. Birk, E. Kretzek, P. Figuli, M. Weber, J. Becker, and N. V. Ruiter, "High-Speed Medical Imaging in 3d Ultrasound Computer Tomography," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 455–467, Feb. 2016.

[115] E. Samei, J. A. Seibert, K. Andriole, A. Badano, J. Crawford, B. Reiner, M. J. Flynn, and P. Chang, "AAPM/RSNA Tutorial on Equipment Selection: PACS Equipment Overview: General Guidelines for Purchasing and Acceptance Testing of PACS Equipment," *RadioGraphics*, vol. 24, no. 1, pp. 313–334, Jan. 2004.

[116] J. Philbin, F. Prior, and P. Nagy, "Will the Next Generation of PACS Be Sitting on a Cloud?" *Journal of Digital Imaging*, vol. 24, no. 2, pp. 179–183, Apr. 2011.

[117] J. C. Honeyman, W. Huda, M. M. Frost, C. K. Palmer, and E. V. Staab, "Picture archiving and communication system bandwidth and storage requirements," *Journal of Digital Imaging*, vol. 9, no. 2, pp. 60–66, May 1996.

[118] M. Vossberg, T. Tolxdorff, and D. Krefting, "DICOM Image Communication in Globus-Based Medical Grids," *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 2, pp. 145–153, Mar. 2008.

[119] Oracle Corporation, "A Performance Evaluation of Storage and Retrieval of DICOM Image Content in Oracle Database 11g Using HP Blade Servers and Intel Processors (White Paper)," Redwood Shores, CA., White Paper, Jul. 2008, accessed: 22nd May 2018. [Online]. Available: http://www.oracle.com/technetwork/products/multimedia/overview/ora-dicom-bench-2008-129543.pdf

[120] ——, "A Performance Evaluation of Storage and Retrieval of DICOM Image Content in Oracle Database 11g Using HP Blade Servers and Intel Processors: A Summary (White Paper)," Redwood Shores, CA., White Paper, Jan. 2010, accessed: 22nd May 2018. [Online]. Available: http://www.oracle.com/us/industries/healthcare/058477.pdf

[121] S. Jodogne, C. Bernard, M. Devillers, E. Lenaerts, and P. Coucke, "Orthanc - A lightweight, restful DICOM server for healthcare and medical research." IEEE, Apr. 2013, pp. 190–193.

[122] K. Barlee, personal communication, Aug. 2017.

[123] B. N. Bloch, A. Jain, and C. C. Jaffe, "Data From PROSTATE-DIAGNOSIS," 2015, dOI: 10.7937/K9/TCIA.2015.FOQEUJVT, accessed: 8th May 2018. [Online]. Available: https://wiki.cancerimagingarchive.net/x/xgEy

[124] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox, and F. Prior, "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository," *Journal of Digital Imaging*, vol. 26, no. 6, pp. 1045–1057, Dec. 2013.

[125] D. Rowntree, *Statistics without tears: an introduction for non-mathematicians*, reprinted with minor corr. and new further reading ed., ser. Penguin mathematics. London: Penguin, 2000.

[126] T. C. Urdan, *Statistics in plain English*, fourth edition ed. New York, NY: Routledge, Taylor & Francis Group, 2017.

[127] S. Godard, "SYSSTAT," accessed: 1st June 2018. [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/

[128] OpenCV, "OpenCV," Feb. 2018, accessed: 2nd May 2020. [Online]. Available: https://opencv.org

[129] Xilinx, "xfOpenCV Library," Sep. 2017, accessed: 2nd May 2020. [Online]. Available: https://github.com/Xilinx/xfopencv/tree/v2017.2

[130] Xilinx Inc., "Vivado Design Suite: AXI Reference Guide UG1037, v4.0," Jul. 2017, accessed: 13th April 2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf

[131] Avnet Inc., "ZedBoard Hardware User's Guide," Jan. 2014, accessed: 13th April 2020. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[132] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology*, vol. 26, no. 3, pp. 297–302, Jul. 1945.

[133] P. Keall, "4-Dimensional Computed Tomography Imaging and Treatment Planning," *Seminars in Radiation Oncology*, vol. 14, no. 1, pp. 81–90, 2004.

[134] P. J. Keall, G. Starkschall, H. Shukla, K. M. Forster, V. Ortiz, C. W. Stevens, S. S. Vedam, R. George, T. Guerrero, and R. Mohan, "Acquiring 4d thoracic CT scans using a multislice helical method," *Physics in Medicine & Biology*, vol. 49, pp. 2053–2067, 2004.

[135] S. B. Jiang, J. Wolfgang, and G. S. Mageras, "Quality assurance challenges for motion-adaptive radiation therapy: gating, breath holding, and four-dimensional computed tomography," *International Journal of Radiation Oncology • Biology • Physics*, vol. 71, no. 1, pp. S103–S107, 2008.

[136] "QUASAR Respiratory Motion Phantom Product Datasheet," 2019, accessed: 18th April 2020. [Online]. Available: https://modusqa.com/wp-content/uploads/2020/02/QUASAR_pRESP_PDS.pdf

[137] "QUASAR Respiratory Motion Phantom Inserts Product Datasheet," 2016, accessed: 18th April 2020. [Online]. Available: https://modusqa.com/wp-content/uploads/2020/02/MMDI_QUASAR_Inserts_PDS-1.pdf

[138] W. Jianlai, Y. Chunling, Z. Min, and W. Changhui, "Implementation of Otsu's thresholding process based on FPGA," in *2009 4th IEEE Conference on Industrial Electronics and Applications*. IEEE, 2009, pp. 479–483.

[139] H. Tian, S. Lam, and T. Srikanthan, "Implementing Otsu's thresholding process approximation unit using area-time efficient logarithmic," in *Proceedings of IEEE Symposium on Circuits and Systems*. IEEE, 2003.

[140] J. G. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "A Novel Architecture for FPGA Implementation of Otsu's Global Automatic Image Thresholding Algorithm," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. IEEE, 2014, pp. 300–305.

[141] P.-S. Liao, T.-S. Chen, and P.-C. Chung, "A fast algorithm for multilevel thresholding," *Journal of Information Science and Engineering*, vol. 17, no. 5, pp. 713–727, 2001.

[142] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Pearson, Apr. 1992.

[143] M. van Herk, "Errors and Margins in Radiotherapy," *Seminars in Radiation Oncology*, vol. 14, no. 1, pp. 52–64, 2004.

[144] A. T. Redpath and L. P. Muren, "An optimisation algorithm for determination of treatment margins around moving and deformable targets," *Radiotherapy and Oncology*, vol. 77, pp. 194–201, 2005.

[145] P. Campadelli, E. Casiraghi, S. Pratissoli, and G. Lombardi, "Automatic abdominal organ segmentation from CT images," *ELCVIA: electronic letters on computer vision and image analysis*, pp. 1–14, 2009.

[146] Xilinx Inc., "Alveo U50 Data Center Accelerator Card Data Sheet DS965 v1.7," Jun. 2020, accessed: 19th July 2020. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds965-u50.pdf

[147] ——, "Versal ACAP Technical Reference Manual AM011 v1.0," Jul. 2020, accessed: 19th July 2020. [Online]. Available: https://www.xilinx.com/support/documentation/architecture-manuals/am011-versal-acap-trm.pdf

[148] F. Robinson, "Dicom transfer rates using fpga-based systems on chip," 2018, accessed: 20th February 2021. [Online]. Available: https://doi.org/10.15129/F8EF7BE8-9A14-4674-B7F4-5497DAD226CC

[149] ——, "Performance of hardware accelerated bone segmentation," 2019, accessed: 20th February 2021. [Online]. Available: https://doi.org/10.15129/1A667DBC-8202-443D-A52B-45B5F8B498D2

[150] ——, "High-level synthesis of hardware accelerated 3d image segmentation based on otsu's method," 2016, accessed: 20th February 2021. [Online]. Available: https://doi.org/10.15129/A80F62FC-F2B2-4866-BB34-62CA39F76525

[151] ——, "Performance of hardware accelerated bone segmentation on 4dct images," 2021, accessed: 20th February 2021. [Online]. Available: https://doi.org/10.15129/E46D23A5-227B-4E2B-8A51-0F4BDD17D644

[152] ——, "Performance of hardware accelerated bone segmentation with noise reduction on 4dct images," 2021, accessed: 20th February 2021. [Online]. Available: https://doi.org/10.15129/89D99E2A-A2C6-4F82-9626-B849FC93E6C2