UNIVERSITY OF CALGARY

Medical Image Registration using OpenCL

by

Sachitsing Dwarkan

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

BIOMEDICAL ENGINEERING GRADUATE PROGRAM

CALGARY, ALBERTA

APRIL, 2012

UNIVERSITY OF
CALGARY

Abstract

Medical image registration is a computational task involving the spatial realignment of multiple sets of images of the same or different modalities. A novel method of using the Open Computing Language (OpenCL) framework to accelerate affine image registration across multiple processing architectures is presented. The use of this method on graphics processors results in a 40-fold speedup against equivalent algorithms running on regular computer processors. At the same time, the use of the normalized mutual information metric during execution of the OpenCL based image registration method results in an accuracy of less than one pixel deviation from the ideal realignment parameters. From a computer and software engineering perspective, the results demonstrate the viability of OpenCL for high performance computing.

Preface

Acknowledgements

I thank all members of the Imaging Informatics lab and the Bone Imaging lab at the University of Calgary for their support over the past two years. In particular, I thank Quinn Thomson and Simon Maia for their valuable input on concepts ranging from general programming to image registration and optimization.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

2D	Two-Dimensional
3D	Three-Dimensional
ALU	Arithmetic Logic Unit
CC	Correlation Coefficient
CPU	Central Processing Unit
CT	Computed Tomography
CUDA	Compute Unified Device Architecture
FLOP	Floating Point Operation
GLSL	OpenGL Shading Language
GUI	Graphical User Interface
GPGPU	General-Purpose computation on Graphics Processing Units
GPU	Graphics Processing Units
GPUCV	GPU-accelerated image processing and Computer Vision library
HLL	High Level Language
HLSL	High Level Shading Language
IEEE	Institute of Electrical and Electronics Engineers
MI	Mutual Information
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
NMI	Normalized Mutual Information

NMR	Nuclear Magnetic Resonance
OpenCL	Open Computing Language
OpenGL	Open Graphics Language
OS	Operating System
PACS	Picture Archiving and Communications System
PD	Proton Density
PET	Positron Emission Tomography
RAM	Random Access Memory
RF	Radio Frequency
SAD	Sum of Absolute Differences
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Threads
SISD	Single Instruction Single Data
SSD	Sum of Squared Differences

Chapter One: **Introduction**

Medical images are one of the most important assets available in clinical practice. Imaging techniques provide physicians with a visual representation of internal organs and tissues such that they are able to analyze and diagnose a patient's condition. Today, diagnosis and monitoring of pathology often involves the acquisition of medical image volumes with multiple contrasts or modalities. Those include but are not limited to methods such as x-ray, computed tomography (CT) or magnetic resonance imaging (MRI). Medical images are very helpful when monitoring patients over long periods of time. Comparison of multiple images can show physiological changes that may have occurred and aid in long-term treatment of a patient's condition.

1.1 Overview of Image Registration

Image registration is the task of aligning and mapping multiple images into a common coordinate space. Coloquially, image registration is sometimes referred to as image fusion. However, image fusion goes one step beyond registration in that the aligned images are merged into a single output image. Figure 1.1 shows an example of medical image registration. On the left, the input misaligned CT and MR images are shown. After a registration procedure, the output images shown on the right are obtained. In this illustration, we can observe that the skull and other internal tissues of the brain from the CT (blue) and MR (red) image slices are correctly overlayed on top of each other; this makes salient features in the images stand out.

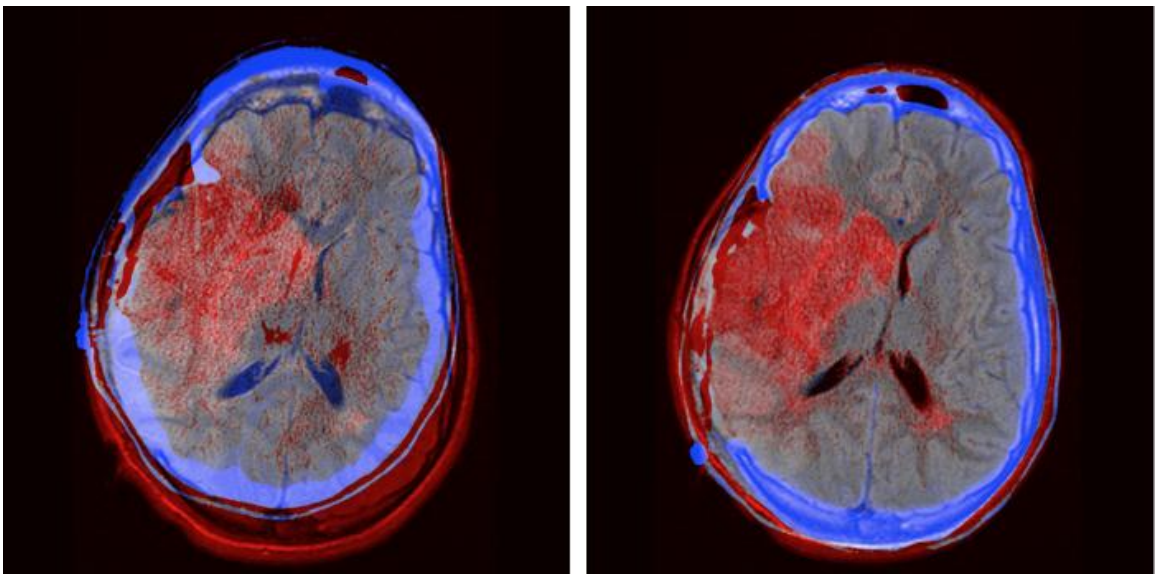

Figure 1.1 - Registration of CT image (blue) to an MR image (red)

Conventionally, medical practitioners mentally map and reconstruct images viewed from a computer screen or printed film and interpret the data to identify soft tissue such as tumours. Expertise and training in this field is required to be able to correctly delineate tissues that may be indicative of the presence of pathology. The process of mentally interpreting data from multiple image volumes can result in variability in the accuracy of diagnosis due to the variability of human interpretation of the images. To overcome this issue, computational methods of performing image registration have been developed but most of those are application specific – algorithms have to be carefully selected depending on the input images. Klein et al. [1] have developed a modular toolbox (Elastix) for rigid and non-rigid medical image registration to provide users with a system that allows for configuration, testing and comparison of multiple registration methods. This software is based on the Insight Segmentation and Registration Toolkit (ITK) [2], which is a well-known software suite for image analysis.

The methods included in the Elastix toolbox run on central processing units (CPUs) and configuration is achieved by typing sets of parameters in text files, which are then used as part of the input process. Therefore, the system is highly user dependent. Manual input of several registration conditions causes it to be sensitive to user error. Such errors may be due to mistyping registration conditions or choice of incorrect parameters by an inexperienced user. One example would be the choice of a global or a local optimizer to find the registration solution based on the degree of misalignment of the input images. However, it is important to note that having fixed sets of parameters is also not an ideal solution due to variations in input images. In the future, with advanced development of artificial intelligence, machine learning algorithms could be used to alleviate this problem. Research by Daewon et al. has shown that it is possible to develop a computational method to learn a similarity measure for multi-modal image registration [3]. Image registration, viewed as a mathematical and computational problem, is an iterative optimization problem as shown in Figure 1.2. The individual components of the registration procedure will be discussed in Chapter 4.

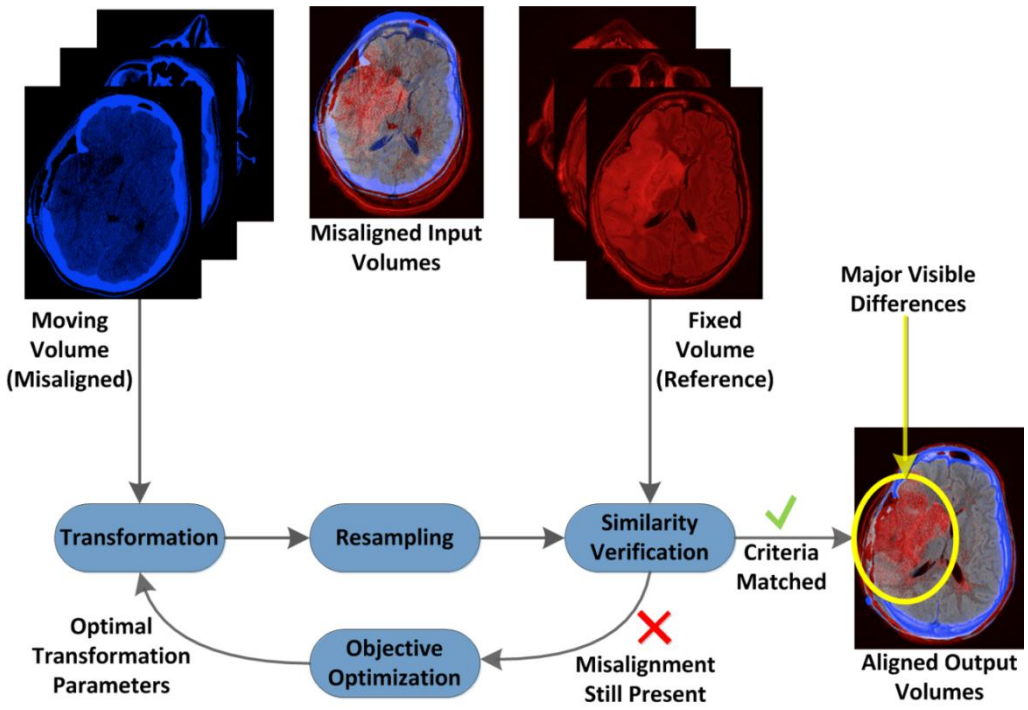

Figure 1.2 - Complete cycle a registration procedure

1.2 Types of Image Registration

Registration methods can be classified into various groups based on criteria defined by Van Den Elsen et al. [4]. In 1998, Maintz and Viergever [5] refined this classification into nine classes. The registration method discussed in this thesis relate to the following four of those criteria:

- Nature of registration basis – Extrinsic or intrinsic methods.
- Nature of transformation – Linear or non-linear.
- Image modalities involved – Mono-modal or multi-modal.
- Subject – Intrasubject, intersubject or atlas.

1.2.1 Extrinsic versus Intrinsic Registration

The basis of registration relies on matching corresponding sets of observations from multiple data sources. The observations can be artificially constructed or can be defined as some existing property within the images. Extrinsic registration relies on artificial markers for the generation of image transformation parameters. Fiducial markers can be implanted into a patient [6] or a stereotactic frame can be externally fixed to the patient [5] to provide reference points for registration. Due to the known exact position of those objects, rigid transformation parameters can be directly computed and applied to the whole set of images. Extrinsic registration represents the gold standard for precise rigid-body registration [5–8].

Intrinsic registration relies on features that can be visually or computationally extracted from the input set of images. In this project, the intensity of the individual image elements is used to determine similarity of two images. Other intrinsic methods use anatomical landmarks or segmentation-based methods to determine point of correspondence between input data sets. For example, a prominent bone structure appearing in multiple images can be used as target point. In segmentation-based method, localization, delineation and matching of a specific tissue visible in the images can be used to generate transformation parameters. Maintz and Viegever [5] reference numerous cases where landmark-based or segmentation-based registration has been used. However, the correctness of those methods is limited by the accuracy of locating and choosing the anatomical landmark or the accuracy of the segmentation method.

1.2.2 Linear versus Non-Linear Registration

Linear registration can be categorized into rigid or affine registration. Rigid methods allow for transformation parameters with six degrees of freedom – three each for translation and rotation about the three coordinates axes. The affine transforms used in this project accommodate for three additional degrees of freedom – anisotropic scaling of image elements can be performed along the coordinates axes. Rigid and affine transformations preserve the linearity of the images and can be described by a single transformation matrix [5]. By applying the corresponding reverse transforms to the output images, the original input images are obtained.

Non-linear registration (also known as deformable registration) allows for more freedom of manipulating the input images [9]. Transformations can be applied to specific local regions of the images instead applying them globally to the entire image. Those operations result in warping of those different sections. The combination of those multiple localized distortions cannot be mapped to a single transformation matrix applicable to an input image to generate the output image in only one mathematical operation. In extreme cases, it may be impossible to revert back to the original image from the output image. For this project, registration was restricted to only rigid and affine methods.

1.2.3 Mono-Modal versus Multi-Modal Registration

As previously discussed, images with multiple brightness and contrasts can be acquired using a single imaging technique. Mono-modal registration involves the use of images of the same modality, e.g., re-alignment of MRI T1 against T2 images. Multi-modal

registration involves the alignment of images of different modalities, e.g., CT to MRI. In this project, the registration methods developed are applicable for both of those cases. The relative similarity between input and output images are determined by the use of similarity metrics. The choice of metric is usually determined by type of input images used. This is discussed in Chapter 4.

1.2.4 Input Image (Subject) Sources

Three categories of input images can be used for registration procedures. Intrasubject means that all the input images come from a single patient. Intersubject involves the use of images from different patients. An atlas image is usually a reference image reconstructed artificially from an image information database [5]. Intersubject and atlas images have a high probability of showing significant dissimilarities between data sets. Accurate registration in those cases would require the use of non-linear transformations. Intra-subject images will not have non-linear distortions due to factors such as different skull size or patient motion during image acquisition. Therefore, a rigid or affine registration procedure would be appropriate. However, some cases of intra-subject imaging may show non-linear characteristics and would require non-linear registration techniques. An example would be sets of images taken before and after surgical operation involving removal of pathology such as tumors.

1.3 Use of GPUs for High Performance Computing

There are various methods of enhancing computational speeds. These include optimization methods that pragmatically check for the best variables to achieve the final

result or writing specialized code in a way to achieve the best possible speed for a particular calculation. Another means of achieving high computational performance is through the use of hardware clusters. This may include the use of centralized interconnected servers and computers. Decentralization of processing units and interconnecting them across a wide area network, commonly known as grid computing, is also another solution. Within a single modern computer, it is also possible to make use of multi-threading to execute multiple tasks at the same time. Multi-threading is based on a task control mechanism that synchronizes the timely execution of instructions based on data dependency and other low-level computational factors. This can help achieve improved processing speeds. However, multi-threading does not offer a purely parallel computing environment discussed in this thesis.

The past decade has been marked by the introduction of consumer-level graphics processors that have significantly higher computational speeds than conventional computer processors. As mentioned by Owens [10], graphics processing units (GPUs) may be seen as the first generation of commodity parallel coprocessors. One of the main differences between a GPU and a CPU is the way that computations are performed. The arithmetic logic unit (ALU) is the component within a processor that is responsible for executing operations. As illustrated in Figure 1.3, for processors of the same size, the number of ALUs (green boxes) in a GPU is much larger than in a CPU. This enables GPUs to execute thousands of operations simultaneously, while a modern multi-core CPU can handle only 2 to 16 operations at the same time.

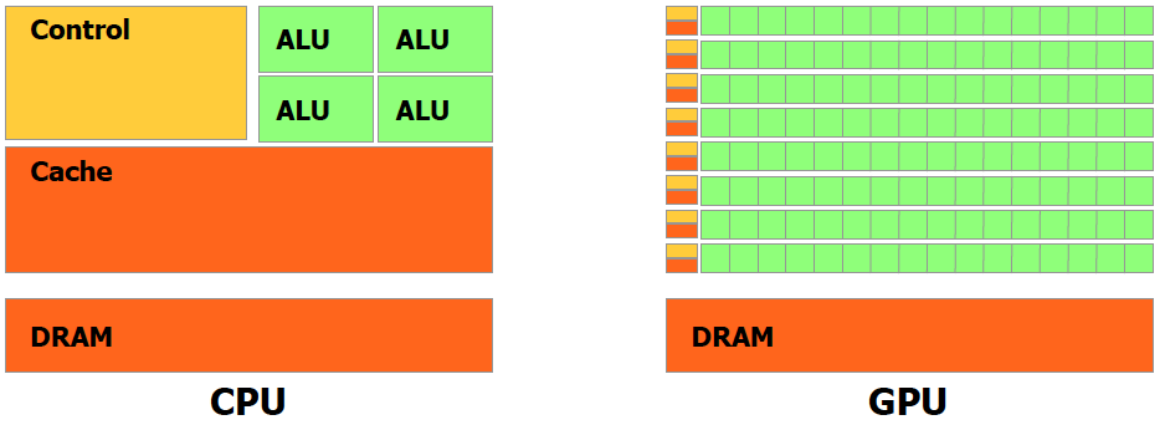

Figure 1.3 - Simplified architecture of CPU vs. GPU

Image Source: NVIDIA CUDA C Programming Guide [11]

Conventionally, graphics processors are used for calculations necessary to render on-screen visuals in real-time – typically in video games. This offloads such tasks from the CPU and helps to maximize the speed at which output can be generated. In 2012, top-of-the-line GPUs are able to perform 15 to 75 times more calculations than modern CPUs. Current trends show that the computational speed of GPUs has a larger growth than CPUs (Figure 1.4). Differences in computing speeds of NVIDIA and ATI/AMD GPUs are attributed to the different designs and manufacturing processes (e.g. silicon die size on the nanoscale level) from those two competing manufacturers. Five years ago, a dual-core CPU performed at 28 giga-floating point operations (gigaFLOPS) while a GPU performed at 500 gigaFLOPS. The best consumer-level CPUs can achieve a rate of 51 gigaFLOPS while this value is 3789 gigaFLOPS for GPUs. Those graphics processors cost between $400 and $600, whilst the CPU costs $1100.

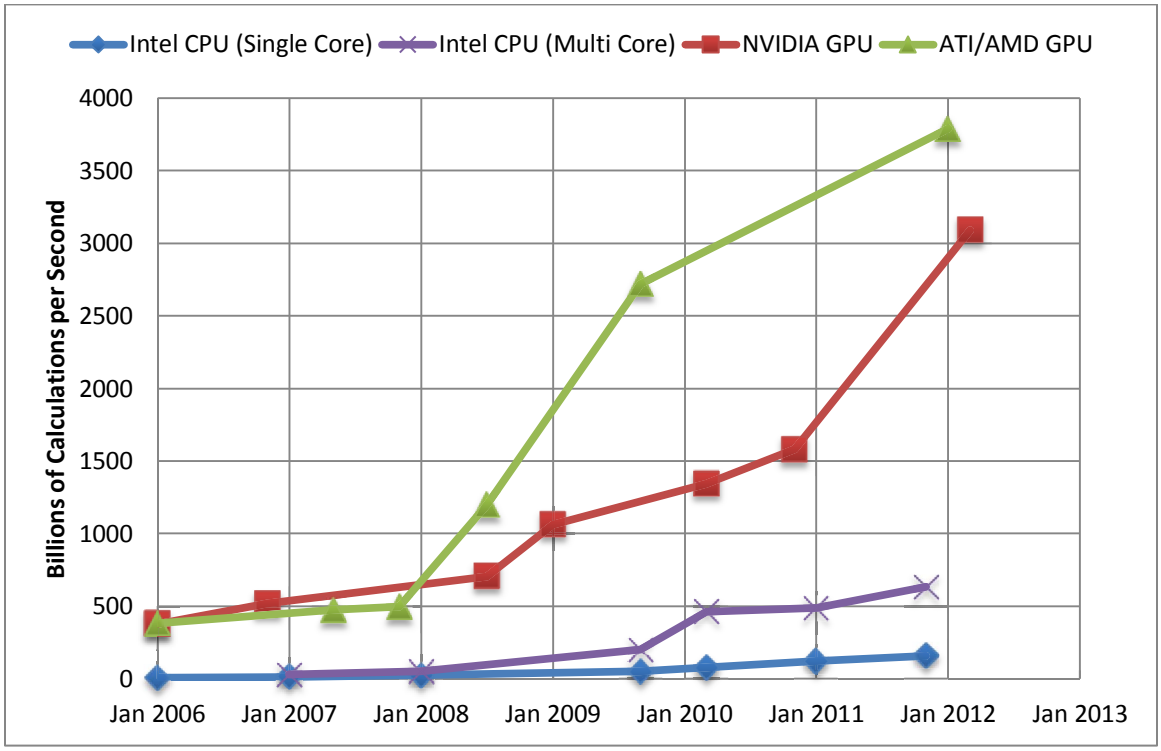

Figure 1.4 - Computational Performance over Time

In the past years, various programming interfaces that enable GPU computations have been introduced. The most noticeable ones have been NVIDIA Compute Unified Device Architecture (CUDA) [12] and Open Computing Language (OpenCL) [13] – both allow parallelized operations to be programmed and run on graphics processors. CUDA and OpenCL are programming interfaces that interact directly with the underlying hardware. They do not rely on intermediate software, such as the Open Graphics Library (OpenGL) to provide general purpose programming functionality for software development. Currently, the most commonly used interface is NVIDIA's CUDA. This is mainly due to the fact that it was introduced in 2007 while OpenCL was publicly made available in 2009. Those extra years in the public domain have resulted that CUDA is more easily recognized as a general-purpose computation on graphics processing units

(GPGPU) computing platform. One differentiating factor of OpenCL is that it has been designed to be a heterogenous platform – software tools built using OpenCL can be run on different types of hardware (CPUs, GPUs or other specialized processors) as opposed to only Nvidia GPUs for CUDA.

1.4 Clinical Applications of GPU Accelerated Medical Image Registration

Image registration currently has varied clinical applications. Some intra-subject uses include image-guided neurosurgery [14] or alignment of sequences of images of a beating heart to observe tissue deformation over time [15]. One inter-subject use of registration involves comparison of images from multiple patients during clinical studies to identify differences in sampled populations [16]. The repetitive registration cycle has enormous computational requirements and would be clinically viable if the computations could be completed within a few seconds. Existing software tools are generally too complicated to use in a clinical setting due to the need of inputting multiple parameters. Experiments performed by Klein et al. [1] have shown registration times in the order of ten minutes when using CT scans for lung cancer screening trials. In the latter cases, to decrease the computational load, the images were shrunk down to a smaller size by a factor of two. In practice, multiple datasets have to be used for accurate diagnosis. Downsizing images can result in loss of valuable information. Therefore, processing huge amounts of data without distorting the original input would require hours and is thus not viable for rapid diagnosis.

There are various methods of performing image registration. The most commonly used technique is to find levels of similarity between the images being analyzed and align

the images based on those criteria. One of those similarity measures is mutual information [17]. The process of computing the mutual information for various images is to construct statistical probability models from the image data. The probabilities are then compared against each other - this involves one or more iterative cycles until a best match is found. As the image sizes grow, so do the probability models. Shams et al. [18] investigated the parallel computation of mutual information on the GPU for real-time registration of medical images. Their results showed a 2x to 4x speed improvement over a CPU implementation when comparing CT scans to MRI T1 and T2 scans and a two-fold speed improvement against proton density weighted images.

One novel use of image registration and fusion involves combining images of different modalities. Ewertsen [19] investigated the possibility of registering real-time ultrasound to pre-recorded positron emission tomography (PET) and computed tomography (CT) images. This is a difficult computational problem, since an ultrasound image is not in the same acquisition plane as CT or PET. The goal was to reconstruct a 3D image like that shown in Figure 1.5. The procedure was however performed offline (not at the time of acquisition of the ultrasound) and took approximately ten minutes with the reconstruction algorithm running on a CPU.

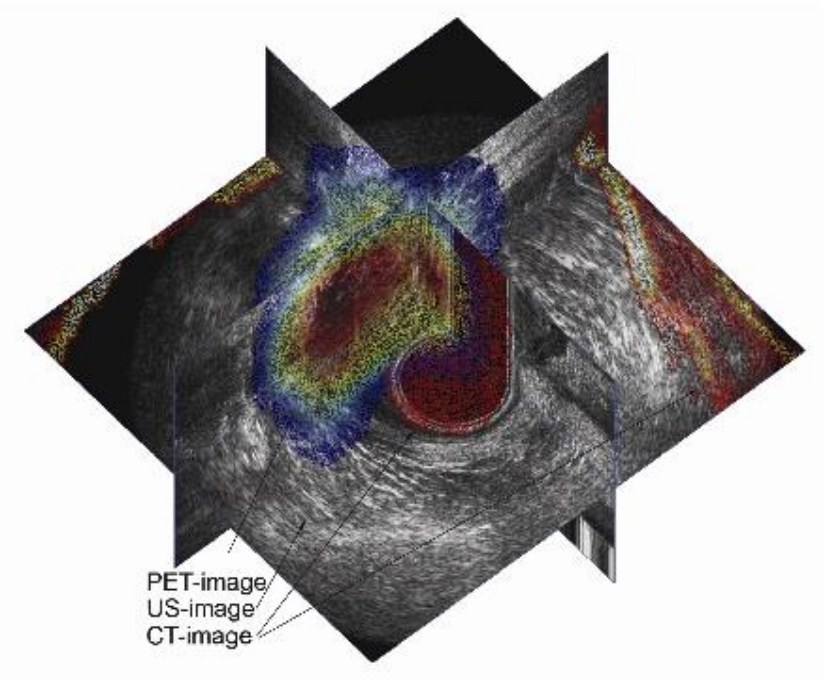

Figure 1.5 - 3D Reconstruction of ultrasound with PET and CT

Image Source: C. Ewertsen [19]

S. Chan [20] and D. Adler [7] developed methods of three-dimensional image registration that can run on GPUs. Those methods were based on the OpenGL API. This programming interface is mainly used in the graphics and video games industry to create special visual effects. Due to this specific graphical purpose, such image registration methods cannot be easily improved. Those implementations were also tied to the Apple Mac operating system and were therefore not portable to other systems. The OpenGL based algorithms demonstrated that parallelized operations on the GPU make it possible to generate a registered image output within 15 seconds using rigid or affine registration techniques. To achieve such speedup, the iterative registration cycle was separated into two parts – one that is comprised of methods that can be enhanced by the GPU and the

other consists of methods that can be more efficiently performed on a CPU. This is shown in Figure 1.6.

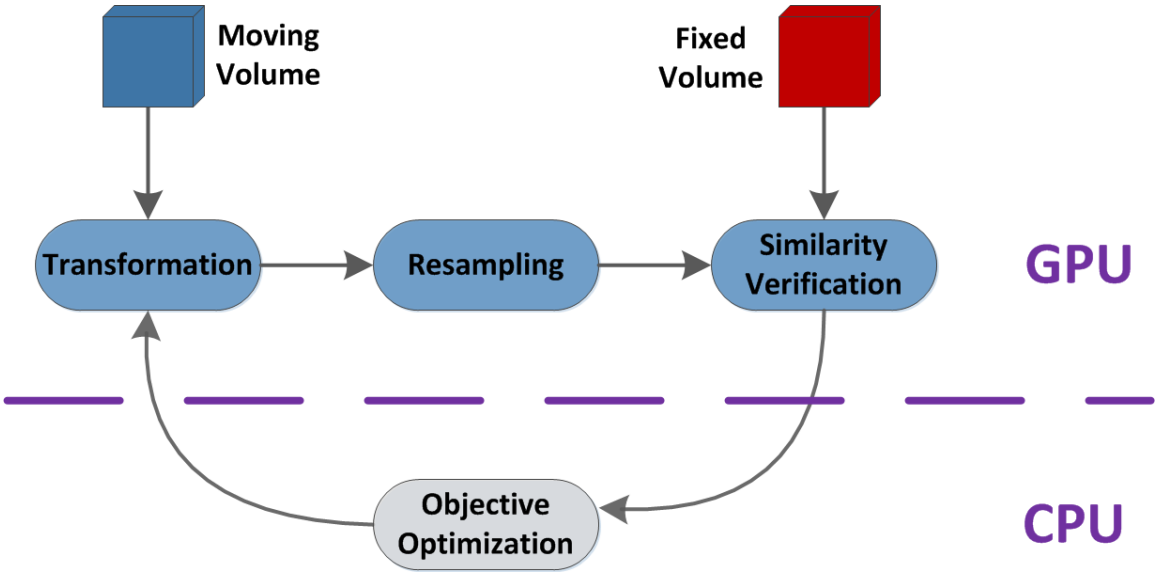

Figure 1.6 - Separation of registration modules on CPU and GPU

D. Adler [7] also demonstrates the possibility of performing non-linear registration on GPUs using a technique based on the Demons algorithm [21]. The use of a freely available APIs such as OpenCL, to construct parellelized registration software packages, can lead to decreased costs of research and development. This could eventually translate to reduced cost of implementation and usage in a clinical setting.

1.5 Thesis Project

1.5.1 Primary Objective

The main objective of the work undertaken for this thesis was to design and validate a portable method of performing affine medical image registration using OpenCL The designed OpenCL code should be able to run on multiple computer architectures (e.g.

GPU, CPU or other processors supporting the OpenCL standard) without requiring modifications to be considered portable.

1.5.2 Hypothesis

Rigid image registration using OpenCL on GPU will provide accurate results on medical images within a perceptibly short time frame of ten seconds and the following conditions:

- A mean re-alignment error less or equal to the largest voxel dimension of the input datasets can be achieved.
- OpenCL based registration will result in at least a ten-fold speedup as compared to equivalent sequential CPU algorithms.

1.5.3 Specific Goals

This work presented in this thesis had the following objectives:

1. Implementation of a modular software codebase for image registration that is independent of processing architecture.
2. Acceleration of rigid and affine image registration by massive parallelization of calculations.
3. Speed and accuracy validation of implemented registration modules.
4. Determination of scalability of cross-platform performance using different processing hardware.

1.5.4 Methods

The goals of this project were achieved through different phases, described below, to ensure that a functional codebase was available for subsequent additions.

1. Structuring of modular codebase to accommodate for the selectable use of OpenCL, OpenGL and CPU implementation of registration algorithms with different processors.
2. Implementation of OpenCL based image registration routines using multiple memory access patterns.
3. Unit testing to determine the best parameters for full testing and validation of the OpenCL implementation.
4. Testing and analysis of final implementation using data sets from the Retrospective Image Registration Evaluation (RIRE) database [22].

1.5.5 Contributions

The work accomplished for this thesis seeks to bridge the gap between decentralized low-cost high performance computing and medical image analysis. Typically, computation involving large datasets is performed on large and costly processing clusters. The use of consumer-level GPUs can significantly increase computational performance of medical image processing routines. In this project, the focus has been on accelerating medical image registration using the recently created Open Computing Language. From a computer and software engineering perspective, OpenCL has yet to gain widespread recognition and use. The results obtained during this project indicate the viability of using the OpenCL platform for both regular software and medical software uses. At the same

time, experiments conducted indicate the scalable performance of OpenCL routines across different processing architectures.

Some of the results presented in this thesis have been presented at the Alberta Graduate Conference (2011), the Care for Cancer Conference (2011), and the Alberta Biomedical Engineering Conference (2011). At the time of writing, a technical note detailing the results of the OpenCL based registration is also being prepared for submission to a scholarly journal.

At a later time, specific parts of the implementation pertaining to the use of OpenCL for the application of image transformations and calculation of mutual information may be made available as an open-source code base.

1.5.6 Future Work

The emphasis of this project has been to showcase a proof of concept of using OpenCL for the potential real-world medical application of image registration. Testing and validation was done on a limited number of processing architectures. Future work would involve using a broader range of processors. Additionally, OpenCL could be used to target other computational procedures, such as optimization, which were not within the scope of this project's goals.

Chapter Two: **Medical Imaging**

Medical imaging is a field encompassing multiple scientific and engineering aspects. Medical imaging can be classified into two broad categories. The first is physics of medical imaging. This would include methods of acquiring and reconstructing images from the scanning machines. Some of the scientific aspects at this level revolve around understanding the atomic interactions that allow the production of images and using such knowledge to improve or develop new scanning techniques. The second part of medical imaging involves interpretation and analysis of reconstructed images to aid in diagnosis or monitoring of patients. Over the past three decades, there have been notable advances in radiology. Figure 2.1 below shows a chest MRI scan obtained in 1977 against a contemporary MRI image of the abdominal cavity. Although artificial colours have been added to the old image for better delineation of features, it can be easily seen that the quantity and quality of information from the modern acquisition show substantial improvements. In this chapter, the three imaging modalities (MRI, CT and PET) used for experimental test cases are briefly reviewed. The subsequent section provides an overview of different types of image registration and how they relate to the work accomplished for this project.

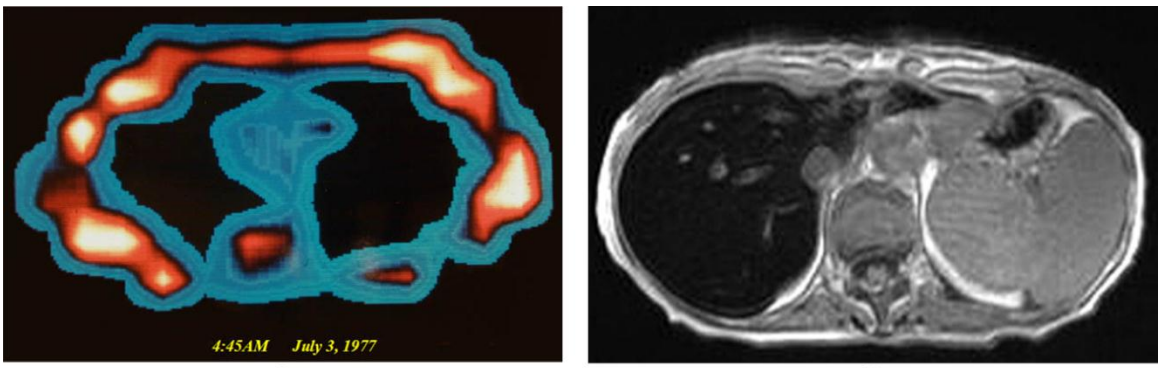

Figure 2.1 - Evolution of MRI reconstruction

Image A is an MRI scan from 1977. Image B is an MRI scan from 2008.

Image A Source [Online]: Dr. Raymond Damadian et al.
http://www.fonar.com/news/images/ics-48.jpg
Last Accessed and Retrieved: February 14, 2012

Image B Source [Online]: Dr. Frank Gaillard
http://radiopaedia.org/articles/haemochromatosis
Last Accessed and Retrieved: February 14, 2012

2.1 Medical Imaging Modalities

2.1.1 Magnetic Resonance Imaging

Some atomic nuclei can absorb or emit electromagnetic waves due to the effect of external magnetic fields. Such nuclei, known as magnetic nuclei, contain an uneven number of protons and neutrons. In the 1940s, Bloch and Purcell demonstrated the phenomenon of nuclear magnetic resonance (NMR), which was later used in the identification of small amounts of chemicals in unknown samples [23].

Magnetic resonance imaging (MRI) is based on nuclear magnetic resonance [24]. The first documented uses of NMR with a focus for potential medical applications stem from the 1970s [25–28]. Modern MRI machines make use of the same techniques to obtain raw data that is processed to obtain reconstructed images shown in Figure 2.1.

Enhanced image quality results from the use of more sensitive equipment (e.g. 3 Tesla magnets), better electronics, and refined reconstruction algorithms. MRI produces images that have very good contrast between soft tissues. For example, brain tumors can be visually distinguished from surrounding white and gray matter. This is possible because soft tissues have a high concentration of magnetic nuclei (mainly hydrogen or carbon atoms) as compared to other tissues such as bone. Therefore, the exhibition of nuclear magnetic resonance is prominent for soft tissues.

The main components of an MRI machine are the magnet, gradient coils and the radio frequency (RF) coil (Figure 2.2). Those components are connected to electronics such as encoders and decoders of the various signals that are part of the imaging procedure. The main magnet produces a constant magnetic field that causes the magnetic moments of nuclei within the body to spin around an axis aligned with the magnetic field, thus causing the effect of nuclear magnetic resonance. A majority of these spins are aligned with, rather than against, the magnetic field, creating a net magnetization in the direction of the field within the object to be imaged. Electromagnetic pulses are input and transmitted via the RF coil, perturbing the steady state system. The signals received in the coil constitute the raw image data. The gradient coils are used to produce slight variations of the main magnetic field within a localized section of the part of the body to be imaged. This is achieved through the use of phase or frequency encoding techniques. Gradient coils allow multiple image slices to be acquired during a scanning session.

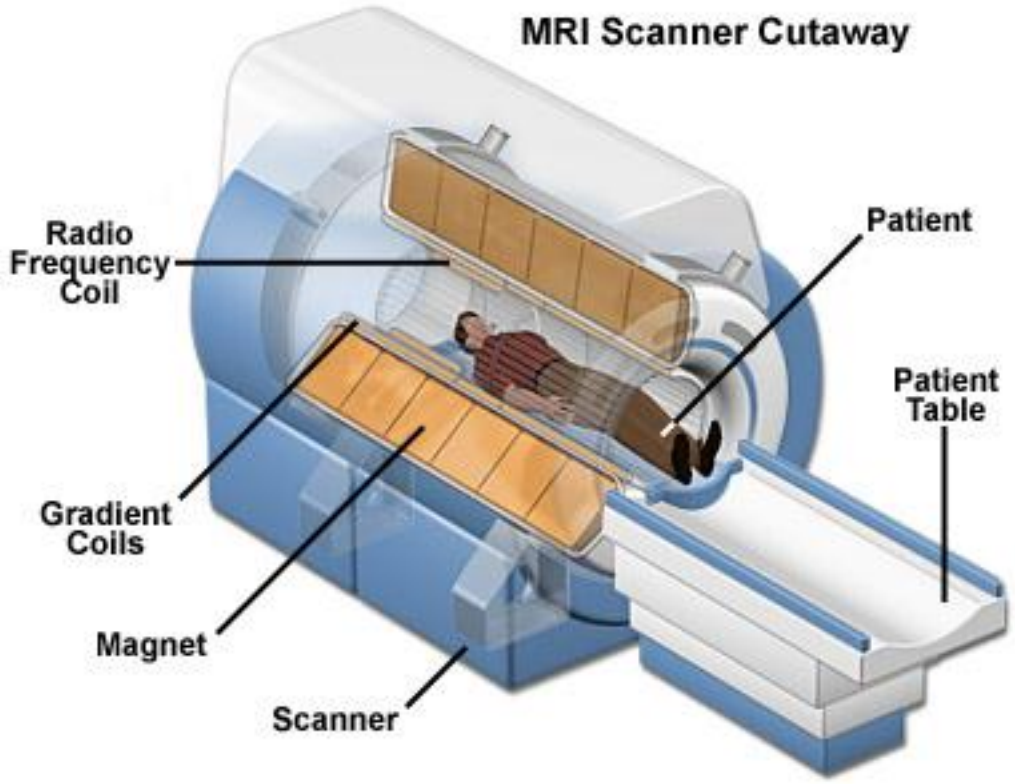

Figure 2.2 - Components of an MRI machine

Image Source [Online]:
http://www.magnet.fsu.edu/education/tutorials/magnetacademy/mri/images/mri-scanner.jpg
Last Accessed and Retrieved: February 14, 2012

MRI can be used to produce images with multiple contrasts of the soft tissues being scanned. Different pulse sequences of the input RF signals will result in those contrast variations. For spin-echo techniques, the length of echo time and repetition time of the input excitation RF pulse determine the type of MR image. Some of the modalities thus produced are denoted as being T1, T2 or proton density (PD) weighted. Tissues such as white matter or fat will be bright in T1-weighted images. In a T2-weighted scan, those tissues will be darker while grey matter or water will be bright. PD has a variation in between T1 and T2 weighted images. Examples of MR images are shown in Figure 2.3.

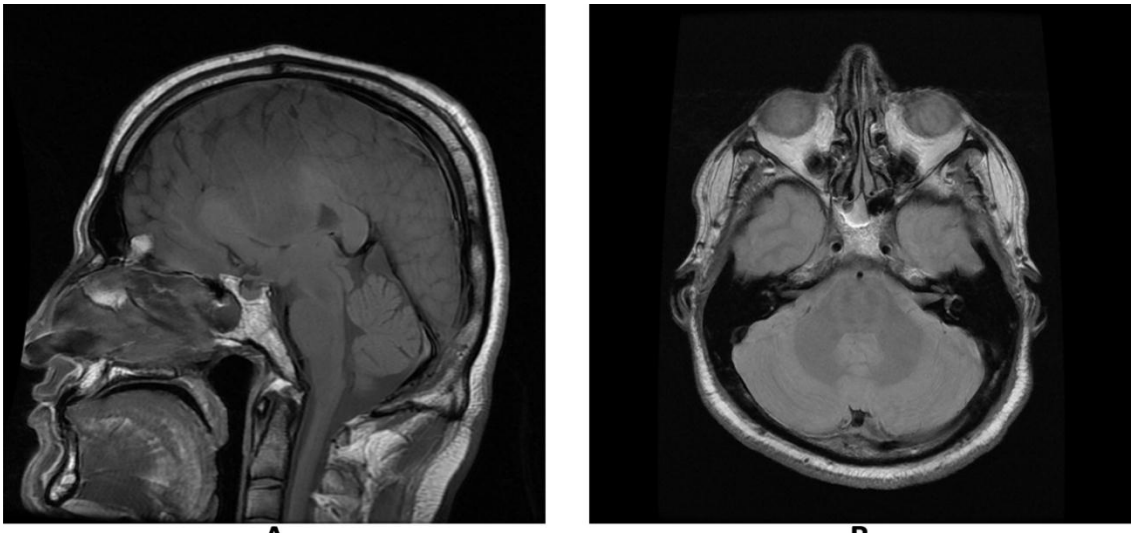

Figure 2.3 - Example MR images
A is proton density MR image obtained in the sagittal plane.
B is T1-weighted MR image obtained in the axial plane.

Image Sources [Online]: Dr. Hani Alsalam
http://radiopaedia.org/cases/glioblastoma-multiforme-1
Last Accessed and Retrieved: February 14, 2012

2.1.2 Computed Tomography

Computed tomography (CT) is an imaging modality based on x-ray imaging. Different tissues within the body have different attenuation coefficients of x-ray photons. This physical property enables acquisition of data necessary for CT reconstruction. A cross-section of a CT scanner is illustrated in Figure 2.4. As x-ray beams pass through a patient's body, they are attenuated by internal tissues. The detector then measures their resulting energies. Multiple readings are taken by rotating the x-ray emitter and detector around the gantry surrounding the patient. The easiest method of reconstructing the image is to sum up all the projections obtained. However, the overlapping projected measurements of the region surrounding the object results in a blurred output image.

Simple backprojection is not used in commercial CT scanners due to the low quality images produced [23]. Instead, frequency domain filters are be applied to the recorded signals to minimize or remove the components that result in the blurring artifacts. The filtered signals can then be used to reproduce the frequency domain representation of the entire image. The inverse transform of the filtered reconstruction results in the production of high quality CT images.

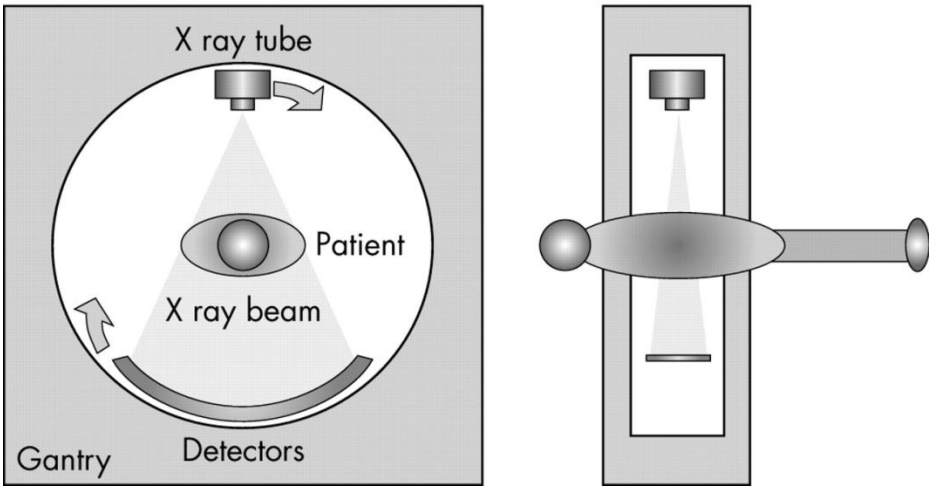

Figure 2.4 - Cross-Section of a CT scanner

Image Source: N. Mollet [29]

Dense tissues have high attenuation coefficients and therefore can absorb large amounts of x-ray photons. One example is bone structure – bones have a bright intensity appearance in CT images (Figure 2.5). The high sensitivity of contemporary CT scanners also allows soft tissues to be imaged – visually distinguishable contrasts between those tissues can be observed. Compared to MRI, CT has faster imaging speed, lower cost, the ability to accommodate larger patients, and no requirements to pre-screen patients for the presences of pace makers, implants, and other potentially MR incompatible devices. However, one downfall of CT is the use of ionizing radiation. Clinically, CT can be used

for diagnosis of medical conditions such as acute stroke [30]. Similar to MRI, CT images can be acquired with varied brightness and contrasts. Depending on the part of the body to be imaged, contrast agents can be administered to patients orally, intravenously or rectally. Examples of contrast agents are iodine or barium sulphates solutions. The flow of those agents through the bloodstream, tissues and organs results in changes in the attenuation coefficient. A common use is for performing CT angiography.

Figure 2.5 - Axial CT slice

2.1.3 Positron Emission Tomography

Positron Emission Tomography (PET) is another medical imaging technique that is based on interaction of nuclear particles. Since it is a tomographic imaging technique, cross-sectional images of internal tissues and organs can be reconstructed using methods based on Radon's method [31]. The apparatus used for imaging has a similar mechanical structure to the CT scanner shown in Figure 2.4. However a full detector ring is fixed around the patient and an external source of radiation is not present. Before the imaging

procedure, a radioisotope is injected into the patient's blood stream. Radioactive decay (positive beta decay) of the isotope results in the emission of a positron. Annihilation occurs when this positron interacts with an electron and gamma radiation (also referred to as gamma photons) is released. The detection of corresponding directional pairs of gamma radiation constitutes the raw data necessary for PET reconstruction. The images produced through PET imaging have poor resolution as compared to CT or MRI. However, PET can show unique physiologic and metabolic information [23]. PET is useful in the diagnosis of nervous system conditions such as Alzheimer's disease or Parkinson's disease [32]. Figure 2.6 shows a PET image with coloured functional regions of the brain.

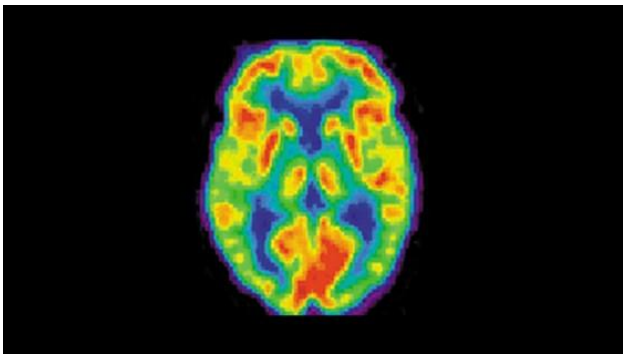

Figure 2.6 - PET image slice

Image Source [Online]:
http://www.cam.ac.uk/research/news/what-do-drugs-do-to-the-brain/
Last Accessed and Retrieved: February 14, 2012

2.1.4 Analysis of Images

MRI, CT and PET rely on different concepts for the reproduction of medical images. Each of those imaging techniques is fine-tuned to outline salient features of tissues that may be indicative of pathology. By merging different image modalities, the quantity and

quality of information presented to physicians can be increased. Computerized and automated image registration can have various clinical applications. The re-alignment of multiple images of the same modality can facilitate the diagnosis or monitoring of pathology across time, e.g., growth of tumors or metastasis of diseases. Image guided surgery is another field that can benefit from image registration. Fast and precise registration of surgical tools against patient images is a basic requirement for such procedures. This thesis project focuses on accelerated image registration on multiple computer processing architectures. To comprehend how fast image registration can be enabled using modern computer hardware, it is important to understand the evolutionary aspects of computing that enable this type of processing. The following chapter provides a review of those aspects.

Chapter Three: **Evolution of Computing**

Over the last decade, there has been a significant shift in how the processing components inside computing devices are designed and utilized. The emergent change has been the introduction of parallel computing and software design accompanied by increased use of GPUs for general purpose computation. As mentioned in Chapter 1, modern GPUs have much more processing power than CPUs. In the past years, the scientific community has sought to harness the power of commodity graphics hardware for general purpose computing [10]. If we consider the current cost of those GPUs (less than $700) against the cost high performance computer clusters (hundreds of thousands of dollars), GPUs provide the highest computational power to cost ratio [7]. In this chapter, the evolving trends of computing and their significance on scientific computing are reviewed. This background information serves as support for the choice of using the recently introduced OpenCL computing platform for medical image registration. Since the focus of the medical image registration presented in this thesis relies on mainly on a GPU implementation, the last section of this chapter reviews OpenCL memory access patterns on those processors.

3.1 History of GPUs and GPU Computing

Prior to 1986, programmable graphics processors did not exist. At that time, specialized processing modules were embedded and hardwired on circuit boards to perform required computations. In 1986, Texas Instruments [33] introduced the world's first programmable graphics systems processor, the TMS34010 [34]. It was innovative because it supported a high level language (HLL) in addition to assembly language [35]. The HLL enabled

programming without the need of intricate knowledge of the hardware and other low level functionality. However, this was a fixed-function processor as it was designed primarily for graphics rendering. Its chief purpose was to handle the interaction between the connected visual display and other parts of the computer system, i.e., it processed the video data such that pixels could be transferred and rendered to the screen.

For most graphical tasks, GPUs follow a specific functional pipeline as shown in Figure 3.1. It is important to note that most of our viewing devices can only show a 2D representation of a 3D object or scene.

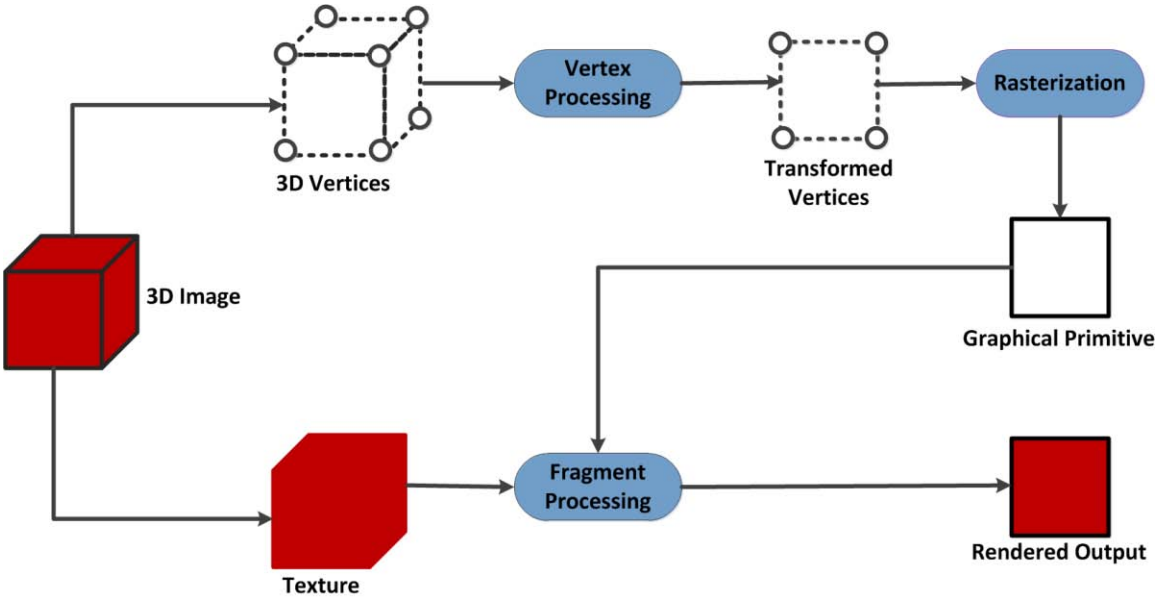

Figure 3.1 - GPU rendering pipeline

The main stages involved in rendering are the vertex and fragment processing operations. In the vertex processing stage, a vectorized representation of the 3D scene is manipulated to generate the vertices of the 2D image, which are then rasterized into fragments. The fragment processing stage then applies the image textures to the

fragments before sending the final output to the screen. Processors like the TSM34010 did not provide the ability to modify any of those stages.

Contemporary GPUs share the same basic operations as the TSM34010, while at the same time incorporating more advanced graphical capabilities. Those allow customizations to be built within the vertex and fragment processing stages of the graphical rendering pipeline to enable enhanced effects on the images being processed. In the early 2000s, the gaining traction in the use of modern graphics processors led to the creation of various high level languages for graphics manipulation and rendering. In addition to having their own specific low level instruction set, GPUs were made available with those high level languages for enhanced programmability. The introduction of the programmable vertex and fragment processing stages in 2002 [36] defined the foundations of allowing general purpose computations on the GPU. For graphics rendering in video games or other graphics design software packages, the most prominent programming interfaces have been the Open Graphics Library (OpenGL) [37] or Microsoft's DirectX [38], both of which are still widely used. Subsequently, the OpenGL Shading Language (GLSL) [39] was introduced as a means of providing refined control over the graphics pipeline. GLSL was specifically aimed at creating modifiable functions, known as kernels, which operate within the vertex and fragment processing stages. Other languages that were developed were Cg (C for graphics) [40] and HLSL (High Level Shading Language) [41]. Subsequently, programming languages and interfaces such as Brook GPU [42] were created with the intention to hide differences between CPU and GPU [43] whilst enabling general purpose computation on GPUs. More recently, high

speed computing on GPUs has been enabled through APIs, such as OpenCL, CUDA or DirectCompute.

The video gaming industry was among the first to adopt graphics processors for general-purpose computation. The purpose was to include high levels of realism into video games by simulating real-life physical models. An example is simulated motion of cloth or other fabric in windy conditions. Recent investigations have shown that this new computational modality can be applied to various computationally demanding tasks other than video gaming. An industrial application is the accelerated processing of large datasets such as those in the oil and gas industry or for seismic activity monitoring [44]. Lately, there has been a focus on combining GPU computing with various aspects of medical image processing. Some of those are filtering, segmentation or registration of medical images. More generalized applicative uses for parallelized computing on GPUs are also becoming more common, especially in scientific fields involving heavily mathematical operations. One discipline that could benefit from this computing modality is finite element analysis. One of the main issues that have to be overcome involves the creation of methods of increasing the performance of those tasks and generating output in real-time.

In 2008, Che et al. [45] conducted a performance study of using CUDA in general purpose applications. The focus of this study was limited to algorithms that involved combinational logic or dense linear algebra. The algorithms were implemented in a similar fashion for both the GPU and CPU. They obtained 2 to 37 times speedup for combinational logic tests and 5 to 17 times speedup for the linear algebra tests. The reason for the wide range of speedup is due to the type of calculation being done or the

way that the algorithms were programmed to run on the processors – computation time is dependent on the efficient and even distribution of tasks.

An important task during image analysis is the reduction or removal of noise by filtering. Image data often contain artifacts and other discrepancies that affect the accuracy and quality of results obtained after processing. Filters can be used to compensate for abnormalities or to detect different features in an image, e.g., a high-pass filter can outline the edges in an image. In 2009, Waage [46] programmed some commonly used filters for image processing using OpenCL. In this study, comparisons were made between CUDA and OpenCL implementations. This assessment led to the conclusions that both GPU implementations have similar performance and OpenCL is a viable choice for general-purpose computation on graphics processors. The growing capabilities of those languages are now promoting the concept of GPGPU computing.

3.2 CPU vs. GPU Architecture

Computing architectures can be classified into four distinct groups according to Flynn's taxonomy [47]:

- Single Instruction Single Data (SISD)
- Multiple Instructions Single Data (MISD)
- Single Instruction Multiple Data (SIMD)
- Multiple Instructions Multiple Data (MIMD)

SISD and MISD fall out of scope of the computing models discussed in this thesis and are therefore not reviewed. SIMD and MIMD relate to modern processing hardware

as they define the concept of parallel computing architectures. In a SIMD system, one operation is performed by multiple processors on multiple sets of data simultaneously. In a MIMD system, different operations are performed by one or more processors on multiple sets of data at the same time. Those two computing modalities are referred to as data parallelism and task parallelism respectively.

Current generation GPUs can be regarded as SIMD processors while CPUs are MIMD processors. In any computer system, the chain of execution of a task involves the controlled process of fetching an instruction, decoding it and dispatching it to the processing units. The design architecture of an SIMD system is simple as compared to its MIMD counterpart. In the fixed-function pipeline of a GPU, a single controller decodes an instruction and then broadcasts it to all the processors. In contrast, a CPU is designed to handle different types of tasks. This results in CPUs having a complex pipeline involving several control paths and code prediction stages. This complexity scales up with the number processing cores in a CPU. Each core incorporates its own separate instruction fetch and decode cycles. Essentially, a CPU is designed to distribute the different stages of a task in a pipeline across time due to the small number of internal processing cores. A GPU is capable of splitting the workload across space [48] (the whole array of thousands of internal processing cores). The distinction described above between SIMD and MIMD processors is a simplification [49]. In fact, there are other factors such as scheduling of tasks that have to be taken into consideration when evaluating those computing architectures.

Consider a two-step task involving floating point operation instructions A and B for a data set containing three members. A simplified representation of the division of the

task on a CPU and GPU is shown in Figure 3.2. In this case, the execution time on the CPU is three times that of a GPU. If overhead time for instruction fetching and decoding is taken into consideration, the CPU runtime would be longer.

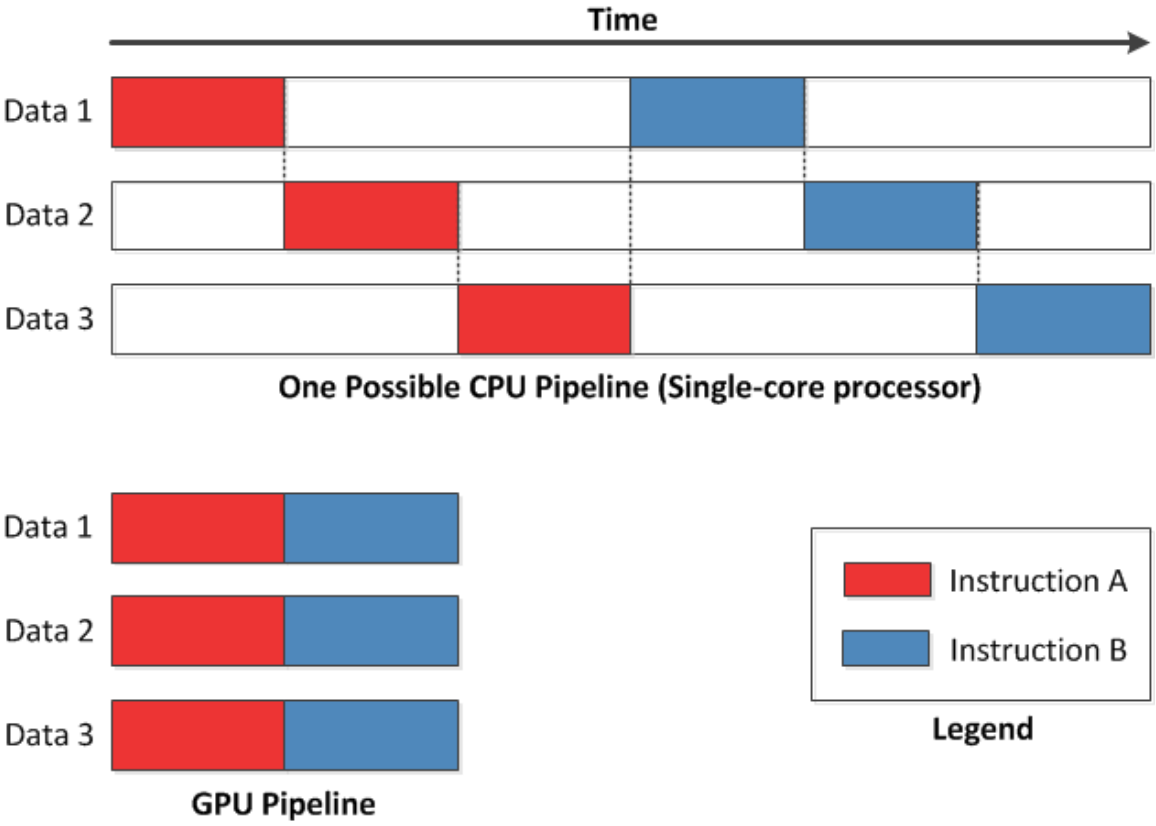

Figure 3.2 - Execution of instructions on a CPU and a GPU

For any practical applications, data parallelism is well suited when there are thousands of information sets to be processed. Using an octa-core CPU for such data would result in a pipeline that shares significant similarity to the single-core pipeline shown in Figure 3.2. The only difference would be a slight reduction in time due to the possibility of eight cores running simultaneously. In contrast, using an AMD Radeon HD5870 GPU would enable scheduling of 1600 simultaneous floating point operations using its 1600 stream processing units.

Due to its hardware simplicity, a GPU is able to achieve a high throughput with low latency. However, one drawback of the GPU architecture is that it is mostly appropriate for algorithms that do not keep track of the state of the data being processed. If the next instruction is dependent on the output from a previous stage, a stall would be generated on a GPU. This could result in significant increases in execution time since the entire set of the first instruction has to be completed on all the processing units before any subsequent operations can be performed. In contrast, due to the more complex scheduling schemes of CPU, cascaded pipelined instructions can be created to minimize or eliminate stall time. In such cases of task parallelism, a multi-core CPU would be more efficient.

Next-generation CPUs and GPUs are trying to approach both types of parallelism efficiently. The result is that the distinguishing features of CPU and GPU architectures are diminishing. Essentially, GPUs are being designed to incorporate subsets of the control mechanisms of CPUs and will thus resemble MIMD designs. The changes to traditional processing architectures entail the requirement of new computing models and software platforms that can make efficient use of the hardware.

3.3 The Need for new Computing Platforms

The pace at which regular CPUs are developed and improved follows Moore's law. This states that the transistor density of microprocessors will double approximately every one and a half years. An equivalent increase in computational performance is also observed. For nearly three decades since the 1970s, this has been achieved by shrinking transistors and increasing the frequency at which processors run. This method of improvement hit a standstill around year 2000 due to numerous reasons. One of those was that increasing

frequencies would result in higher operating temperatures of the CPU; this can lead to incorrect binary logic at the transistor level especially when voltage and current tolerances are very low. The workaround was to keep the frequency the same but instead pack multiple processing cores within the same CPU package. The development of multi-core processors thus resulted in the upkeep of Moore's law.

The issue that arises is that traditional computing models follow a sequential process that can be easily be handled by one processor. By increasing transistor count and frequencies, a natural performance gain would be observed without modification of any algorithms. The use of multiple independent cores within the same CPU does not result in the same effect. Therefore, algorithms have to be redesigned such that they can make use of either task parallelism or data parallelism. Depending on the computational problem, an algorithm running in O(N) time sequentially has the potential of being parallelized to run in O(log N), where N is the size of the input data set [50]. As mentioned by Owens et al. [48], "Parallelism is the future of computing". The evolution of microprocessors has led to this new computing model becoming more and more prevalent. The need for massive parallelization of existing computing architectures can also be attributed to the following major factors:

- Computational requirements for multiple types of tasks are becoming large. The amount to data to be handled at a given instant can be substantial, on the order of gigabytes to petabytes of information. This ranges from huge volumes of data for scientific computing to databases of social networks such as Facebook or Google+.

- Real-time or close to real-time processing of data is being sought. This is especially prevalent for web applications for which consumers expect an immediate response to their interaction with a web server when that particular server may have to serve thousands of consumers at the same time.

Unlike CPUs, modern GPUs are designed with a parallel architecture due to the specific function that they are required to achieve. As discussed in Section 3.2, a GPU does not incorporate complicated sequential control paths. The freed up transistors may be used to implement additional computational circuitry instead. Therefore, the performance of GPUs can be increased much more easily than for CPUs. Since graphics performance is roughly doubled every six months [10], the rate of improvement of GPUs outpaces Moore's law. To keep up with this pace, graphics programming interfaces such as GLSL and HLSL are constantly updated to provide new functionality through the graphics rendering pipeline. Those languages primarily provide functions to manipulate graphics primitives. The use of such constructs for scientific computing is complex, non-intuitive, and does not always provide the ability to define custom data types. As identified by Farrugia et al.[36], the use of those languages for image processing or computer vision application is hard since they are not image specialized. Circa 2006, they introduced the GPU-accelerated Computer Vision library (GpuCV) that uses the computing power and parallelism of GPUs to facilitate the implementation of image processing routines. The open-source GpuCV framework provides a layer of abstraction on top of GLSL by managing the graphical hardware capabilities [43] without the

programmer having to use the underlying graphics primitives. Since this interface focuses on computer vision applications, it is not useful for more generalized computations.

With the possibility of GPUs being able to perform non-graphics related tasks across the traditional graphics rendering pipeline, there is a need to introduce computing platforms that can accommodate for a wide variety of potential applications. Additionally, the production of accurate and consistent results across multiple executions of the same routine is one of the most essential aspects of any computational task. Those issues were first tackled by the Brook for GPUs framework [51] which is now a project that is no longer regularly maintained. Brook focused only on the aspect of data parallelism. In many situations, both task and data parallelism may be required which made Brook a non-ideal solution.

CUDA was the first GPGPU platform that addressed both types of parallelism. By architectural design, the GPU is seamlessly capable of data parallelism. Through the use of stream processing and intricate thread control, CUDA can be used to achieve task parallelism. One or more processing streams can be created to execute different kernel and synchronized against each other. Since this method deviates from the regular SIMD definition, the term "Single Instruction Multiple Threads (SIMT)" has been used refer to this computing modality. The main issues with this method is that task parallelism is achieved using low-level routines available only through CUDA, which adds to the complexity of developing task parallel instructions on a GPU. The other issue with CUDA is that of cross-platform capability; CUDA is only available for GPUs produced by Nvidia Corporation. On the other hand, Microsoft's DirectCompute API is built upon the DirectX 10 and 11 libraries which are supported by most GPUs built in the past 3

years. As discussed above, Brook, CUDA or DirectCompute target only GPGPU applications. Their purpose is to make software development on GPUs accessible to developers who may not have had prior experience with any form of graphics or GPU computing.

Even after the introduction of various parallel processing libraries, software is still designed to run in a sequential manner. With the shift in design of the traditional CPU to include multiple cores, computing models that make efficient use of those processors have to be developed and promoted. Introduced in 2008, OpenCL is the newest platform that enables parallelization of routines on multiple types of processors. OpenCL builds upon the advantages of previous GPGPU implementations and incorporates functionality that allows the same source code to be executed on both CPUs and GPUs. The distinguishing functions from CUDA or DirectCompute is that the OpenCL framework was built to natively include both task and data parallelization routines. Major technology companies are working towards building products that are OpenCL compliant across the three major operating systems (Microsoft Windows, Apple Mac OS and Linux) in use to date.

The best way to make use of parallelism is through heterogeneous co-processing. This means that both the CPU and the GPU are used for a complete task; the CPU is used for sequential code containing critical control paths while the GPU can be used for data parallelizable sections [52]. Today, processors in smartphones or tablets integrate both a CPU and a GPU on the same silicon die. With the high rate of adoption of those devices, it is critical to devise computing models that are scalable with the processors' architectures and that also allow future upgrades to be implemented with ease. The

current OpenCL standard defines some methods that are currently redundant but that would be usable by future hardware. One example is the current generation of Radeon HD7900 series GPUs from AMD which have a completely revised architecture allowing them to operate in an MIMD fashion if required. Although OpenCL is currently an underused resource, it has the potential of bringing significant changes to hardware and software design.

3.3.1 Potential Applications

The computing world has now reached a point where the demand for high speed computation (for e.g. video gaming or playback of high definition movies) is being sought in relatively small devices such as smartphones or tablets. Although the local processing power of such devices is increasing rapidly, they are unable to store huge amounts of information. Nowadays, such data is decentralized and stored on a remotely connected server. Cloud computing is thus becoming pervasive. The concept of cloud computing involves moving computationally expensive tasks away from consumer devices and performing those on a central server. In such networks, the request from thousands of consumers may have to be processed simultaneously. The traditional methods of using clusters of CPUs may soon not be able to cope with the processing demands of such tasks. One potential method to overcome such computational problem involves distributed or grid computing. This involves linking various computing resources across multiple networks. The combined pool of processors can therefore provide enough computing power to process large datasets. However, there are significant disadvantages to this method. Foremost are the hardware level differences –

speed, processor type and supported instruction sets are some items to be considered. Additionally, the computing resources may not have access to identical network speed. The compounded effect of those issues increases the complexity of setting up and synchronizing grid computing systems. Comparatively, GPUs have very high computational speeds and local clusters of GPUs take up very little space as compared to traditional computing clusters – three to four graphics cards can easily fit in desktop computer enclosures. Therefore, clusters of GPUs do not need to be decentralized. The use of GPUs and new computing models built upon massive parallelization therefore have the potential of enabling real-time processing in situations where low latency and high bandwidth are important.

In clinical practice, such methods would be highly applicable to medical image processing, where accurate and consistent results are required. Patient data stored on PACS (Picture Archiving and Communication System) servers could be processed very rapidly before being sent to a physician's computer station. Computerized procedures of image registration or segmentation have not gained popularity since those tasks could take anywhere from minutes to hours to execute for a single set of patient data using traditional CPUs. Lack of accuracy and precision of intrinsic methods of image registration also contribute to the low adoption rate of this technology. Quantification of the accuracy of a computerized image registration procedure is hard to accomplish in a clinical setting due to the variability in the images being used, for e.g., parameters used to register CT to MR images of one patient will likely be different for images of another patient. However, one possibility could involve measurement of similarity or alignment of input or output registered images against reference artificial image atlases.

Additionally, clinical computer systems are often comprised of different types of hardware and software. The cross-platform capability of OpenCL could help overcome this latter barrier. As a focus of the work accomplished for this thesis, the validation of OpenCL for image registration can drive interest to this computing platform. This could lead to significant improvement of the speed and efficiency of delivering health care.

3.4 Memory Access Patterns using OpenCL

Memory access patterns on a GPU are different from those on a CPU. Conventional data read and write methods used by a CPU follow a streamlined and controlled pipeline to increase the accuracy and consistency of the system as a whole. Due to its parallelized architecture, those conventional methods are not appropriate for high speed computation on GPUs. A simplified diagram of the memory layout of an OpenCL device is shown in Figure 3.3.

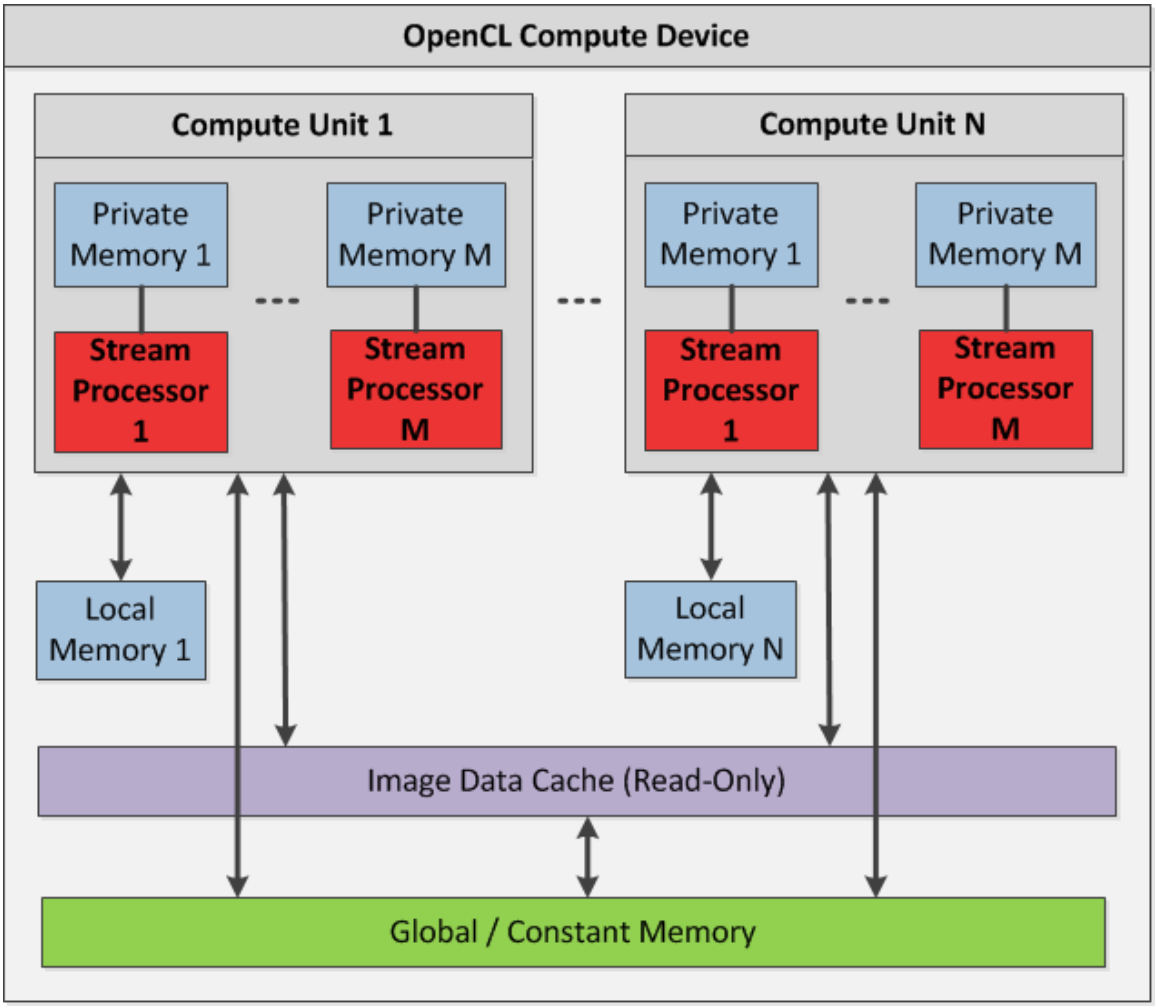

Figure 3.3 - Memory domains and interaction for OpenCL

Depending on the type of processing to be done and the expected output, the choice of memory access method can greatly influence the computational speeds achieved on a GPU. Under the OpenCL platform, the compute device is split into 'N' compute units, each having its local memory domain. Local memory is extremely fast but has a very small capacity. Most current generation GPUs have 32KB or less local memory making this ideal only for temporary storage of values during an operation. Within each compute unit, there are 'M' stream processors. Stream processors are the modules that are used to

split the parallel workload for each compute unit. The private memory space is reserved for and managed internally by each stream processor; this memory domain cannot be controlled through the OpenCL interface. For the image registration application discussed in this thesis, the following 3 memory access methods were available:

- Use of global GPU memory
- Use of read-only image memory
- Use of read-write image memory

3.4.1 Global Memory

Global memory is commonly referred to as video random access memory (VRAM) for GPUs. This is the largest memory space available on a compute device and offers full read-write capabilities. Compute units can directly access this memory space. Addressing global memory can be done using blocking or non-blocking operations. Blocking or uninterruptible operations can help ensure that the contents of global memory are not overwritten or corrupted during data transfers. This can result in an overall slowdown of the system since read/write accesses may be stalled during those operations. Regardless of whether blocking or non-blocking operations are used, global memory has the slowest data path of all the available memory access methods on a GPU. However, it is important to note that the peak memory bandwidth achieved by a GPU in this mode is significantly greater than CPU access to regular computer memory. Such differences depend on hardware components and configuration of the computer being used. As specified in section 5.5, at peak ideal performance, the GPU used for this project has a memory bandwidth that is at least five times larger than the CPU used. It is

important to note that, under regular computational workloads, ideal performance cannot be achieved. Due to the larger number of control paths in a CPU, it is expected that memory bandwidth achieved using the GPU would be greater than the five-fold difference.

3.4.2 Read-Only Image Memory

Since GPUs are built for rapid graphical operations, special control paths are built into the hardware for fast access to stored images or textures. To achieve this speedup over global memory, image/texture memory space has to be pre-allocated. This enables the texture's address space to be cached in the image/constant data area of the device. Once loaded into GPU memory, access to the textures can be restricted to unilateral read-only operations through the use of memory address offsets. This means that the images loaded cannot be modified within any kernel operations. For GPGPU applications, vector data types can be stored into image memory. However, the data has to be formatted to make efficient use of the way that it will be accessed. One such method is coalescing. This involves bunching a set of data points together such that they are all read in one cycle.

3.4.3 Read-Write Image Memory

Read-write image memory uses the same domain as read-only memory. The only difference is that the control path is bi-directional to allow both read and write operations to be performed. Due to the additional control constructs, access is slightly slower than read-only memory. However, OpenCL imposes the restriction that memory defined as read-write can be used for only one of those operations at a time within a kernel. This

ensures that any image memory space is not overwritten or corrupted due to simultaneous write operations from multiple compute units or stream processors.

3.5 Summary

Evolutionary aspects of computing have led to the introduction of new hardware and software development platforms. Those advances invariably come with increased processing speeds, especially for the highly parallelized GPU architectures. In the next chapter, applications of computational methods to intrinsic medical image registration are discussed.

Chapter Four: **Accelerated Image Registration**

Over the past decades, various methods of improving medical image registration have been developed. While some have focussed on improving the accuracy of registration, others have looked into speeding up this computational procedure. As presented in Chapter 6, Freiman et al. [53] have developed a highly accurate multi-modal image registration algorithm. However, even with the optimization of their sampling method to estimate the mutual information metric, they obtained CPU based runtimes averaging at more than 1 minute in the best case. A possible new approach to accelerating computationally intensive tasks in a clinical setting is to make use of GPUs. To date, there are very few published methods of speeding medical image registration using graphics processors. Due to the number of components and different algorithms involved in image registration, it can be a hard procedure to accelerate.

In this chapter, the main components of the intrinsic image registration system that was re-designed to make use of OpenCL are discussed. In addition to data input, as can be seen in Figure 1.2 and Figure 1.6, there are four main procedures in this image registration cycle.

Those are:

- Transformation
- Resampling
- Similarity metric calculation
- Optimization

4.1 System Input

Input can be separated into 2 parts – image volumes and registration parameters. Two volumes are input into the registration cycle; the first is referred to as the fixed volume and the other is the moving volume. The fixed volume is a reference volume and is not modified during the registration procedure. The moving volume is the one that is manipulated and re-aligned to the fixed volume. At the end of the cycle, an output volume is generated from the moving volume.

To store or represent an image on computer memory, only the full image dimensions (width, height and depth) and the intensity values of the picture or volume elements are required. This information enables the re-construction and display of the image graphically. This representation of the image or volume, when copied to GPU memory, is commonly referred to as a texture.

A registration procedure is dependent on the choice of initial parameters that define the conditions of iterative cycle and how the output image is generated. Some examples of those are type of similarity metric or number of histograms used when calculating mutual information. The parameters chosen are discussed in the "Methodology" and "Discussion" sections of this thesis.

4.2 Transformation

Transformations are the most basic operations involved in an image registration procedure. Rigid or affine transformations include rotations, translations or scaling (affine only). Those transformations preserve the linearity of input images as well as their original dimensional aspect ratio. In contrast, other linear transformations (e.g. shearing)

and non-linear transformations will modify the input images resulting in distortions in one or more dimensions.

Geometric operations cannot be applied directly to a texture due to the lack of spatial information. A preliminary step to the transformation process is to generate the spatial coordinates (also known as texture coordinates) of the stored volumes using the known real dimensions of voxels; those coordinates can be stored in a 2D or 3D matrix. This enables the use of simple matrix operations for the application of a desired transformation. For rigid or affine transformations, a 4x4 matrix that defines the desired geometric operations can be constructed. This 4x4 matrix can then be multiplied to every spatial coordinate of the input volumes. The result of those computations represents the coordinates of an output volume.

The operations involved during transformation are performed on a per pixel or per voxel basis. This property allows such operations to be performed concurrently. On a GPU, transformations can be parallelized and completed in significantly less time as opposed to using a sequential process to perform the calculations. An example of a real-time application of such calculations is the handling of rigid body movements in video games.

4.3 Resampling

After applying transformations, it is necessary to obtain a graphical representation of the resulting volume for either visualization or further processing. The output coordinates from the transformation operations may not match an exact voxel location in the input volume. A resampling procedure is required to reconstruct an output volume from those

coordinates. Interpolation can be performed using a nearest neighbour method, linear method or a higher order algorithm, such as, polynomial or spline. The OpenCL API provides access to sampler objects that can be setup with the desired resampling conditions. The two techniques built into the OpenCL API are nearest neighbour and linear interpolation. This abstracts the need to write specialized algorithms for handling this procedure. Higher order methods are not directly available through the API and are therefore not reviewed and used in this project. Resampling can be conducted on individual voxels at the same time and can therefore be parallelized.

4.3.1 Nearest Neighbour Interpolation

The simplest resampling method is known as nearest neighbour interpolation. In this procedure, the transformed coordinates are clipped or rounded to the closest voxel location. The intensity value of that element is set at the output location. This method is very fast but will result in poor image quality. The output image will have substantial aliasing; neighbouring pixels may not have a matching colour gradient as shown in Figure 4.1.

Figure 4.1 - Image quality due to aliasing

Image Source [Online]:
http://www.gemaga.com/wp-content/uploads/2007/12/anti-aliasing.thumbnail.png
Last Accessed and Retrieved: February 14, 2012

4.3.2 Linear Interpolation

The second method makes use of linear interpolation techniques. When using 3D volumes, it is possible to use a tri-linear interpolation method. As illustrated in Figure 4.2, the 8 voxels (P1 – P8) forming a cubic lattice around the transformed coordinates of the voxel P are used to calculate its resulting voxel intensity. Since the spatial distance of the neighbouring voxels is used to calculate a weighted average output intensity, this interpolation technique produces a much smoother colour gradient for adjacent voxels of the output volume.

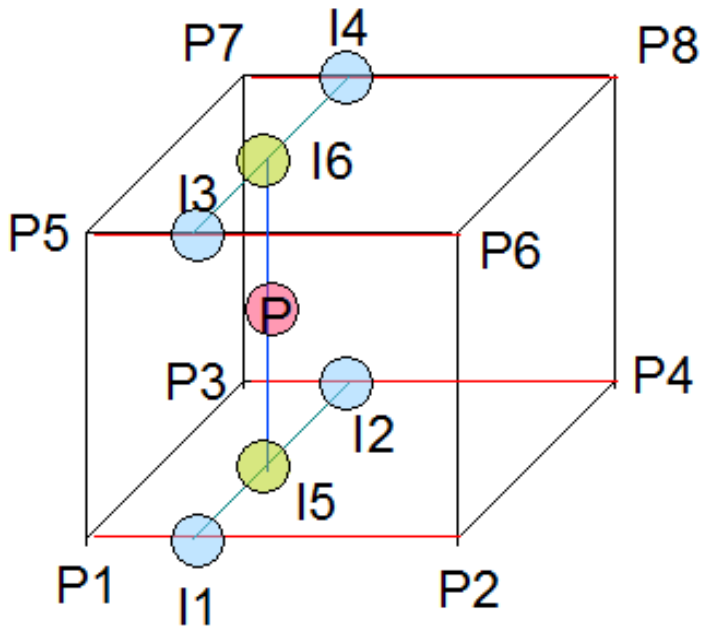

Figure 4.2 - Trilinear interpolation

Image Source [Online]:
http://gilgamesh.cheme.cmu.edu/doc/_images/trilinear-interpolation.png
Last Accessed and Retrieved: February 14, 2012

The equation defining one interpolation is as follows:

$$\boldsymbol{I_{out} = \frac{(P_x \times d_x) + (P_y \times d_y)}{d}} \qquad \textbf{Equation 4.1}$$

where I_{out} is the output intensity, P_x and P_y are the voxel intensities at input locations x and y, d_x and d_y are the distances from the output location to location x and y respectively, and d is the distance between x and y. The voxel pairs used to determine the final intensity of location P shown in Figure 4.2 are described in Table 4.1.

Input Pair	Output Intensity
P1, P2	I1
P3, P4	I2
P5, P6	I3
P7, P8	I4
I1, I2	I5
I3. I4	I6
I5, I6	P

Table 4.1 - Trilinear Interpolation Pairs

4.4 Similarity Metric Calculation

The level of correspondence between the input fixed volume and the output resampled volume can be assessed using a similarity metric. Similarity metrics for image registration can be calculated based on different types of measures depending on the input images used. Methods based on the minimization of absolute or squared intensity differences are most likely to be used to compare images of the same modality [5], e.g., two MRI volumes of the same patient. Mutual information (MI) and normalized mutual information (NMI) do not assume any dependence between images [9] and can therefore be used for multimodal image registration, e.g., MRI against CT images. All four of those metrics were implemented in our registration method. The registration cycle can be stopped when the calculated similarity metric shows significant correspondence between the two volumes, or when a desired similarity criterion is met.

4.4.1 Difference-Based Measures

Computed differences between the intensities of corresponding pixels from two images can be indicative of their relative similarity. Minimization of the sum of the squared differences (SSD) or the sum of absolute differences (SAD) denotes a high level of correspondence between the input images. The normalized versions of those metrics are commonly used by taking into account the number of elements in the domain being analyzed. In literature, the normalized metrics are also referred to as mean squared difference or mean absolute difference.

The equations defining the calculation of those metrics are:

$$\boldsymbol{SSD(A,B) = \frac{1}{N}\sum_{x\in\Omega}\big(A(x) - B(x)\big)^2} \qquad \textbf{Equation 4.2}$$

$$\boldsymbol{SAD(A,B) = \frac{1}{N}\sum_{x\in\Omega}|A(x) - B(x)|} \qquad \textbf{Equation 4.3}$$

where *A* and *B* are the two images to be compared, Ω represents the overlapping domain of the images, and N is the number of elements in the domain. Since those metrics measure the intensity differences, they tend to be sensitive to large intensity values that may be present in one image and not in the other. However, the use of SAD instead of SSD can mitigate the effect of those large intensity differences. For cases where the two images only differ by Gaussian noise, SSD can be proven to be optimum measure [54].

SSD and SAD metric calculation can be easily programmed to run in a parallelized manner. The difference operation can be performed individually for every pair of corresponding pixels or voxels. The summation can be reduced into multiple sums where an initial sum of all values in a particular row or column may be computed

concurrently. The resulting sums can then be combined to obtain the final sum. The efficacy of performing the concurrent sums can diminish when the number of values is small. In such cases, a sequential process running on a CPU may be faster. The algorithms used to compute the difference metrics is discussed in Chapter 5.

4.4.2 Correlation Coefficient

One of the failure modes of the SSD and SAD metrics arises when images differ by factors other than Gaussian noise. Depending on the conditions prevailing or parameters chosen at the time of image acquisition, the images obtained may only have differences in brightness or contrast. Without prior knowledge of those conditions, one can make a conservative assumption that there exists a linear relationship between the intensity values of the images. In such cases, the optimum similarity measure would the correlation coefficient (CC) [54], defined as:

$$\boldsymbol{CC(A,B)} = \frac{\sum_{x\in\Omega}\left(\boldsymbol{A(x)} - \overline{\boldsymbol{A}}\right).\left(\boldsymbol{B(x)} - \overline{\boldsymbol{B}}\right)}{\sqrt{\sum_{x\in\Omega}\left(\boldsymbol{A(x)} - \overline{\boldsymbol{A}}\right)^{2}.\sum_{x\in\Omega}\left(\boldsymbol{B(x)} - \overline{\boldsymbol{B}}\right)^{2}}} \qquad \textbf{Equation 4.4}$$

where $\overline{A}$ and $\overline{B}$ are the mean voxel intensities in image *A* and *B* respectively. During image registration, the maximization of the correlation coefficient is indicative of better alignment of the input images. GPU acceleration of the computation of the correlation coefficient can be achieved using an algorithm similar to the calculation of SSD or SAD.

4.4.3 Mutual Information

Multi-modal image registration as compared to mono-modal registration presents a different set of parameters to consider for the evaluation of similarity between the two images. For mono-modal registration, a direct colinearity can be assumed to exist between the image sets, for e.g., changes in brightness or contrast. In such cases, those variations are expected to be similar across the whole domain of overlapping pixels or voxels of the input volume. However, images captured using different imaging techniques result in significant differences in intensity and contrast for the same overlapping domain. Additionally, some anatomical structures may be visibly absent or may have poor visual quality in one modality. For example, bone structure will be prominent on a CT image, while a tumor with high water content would show up well on a MRI scan.

One method of comparing two images of different modalities is to determine how much information from one image is contained in the other. A method of estimating the information content of a signal was first introduced by Shannon [55] in 1948. This measure, now known as the Shannon entropy, is defined as:

$$\boldsymbol{H = -\sum_{i=1}^{n} p_i \log p_i} \qquad \textbf{Equation 4.5}$$

where n is the set of possible discrete values contained in the signal and p_i is the probability of occurrence of each discrete value over the entire signal. For an image whose intensity values are stored as 8-bit values on computer memory, the discretization of those values can be viewed as a set of 256 grey scale levels ranging from 0 to 255. A

1D histogram can first be constructed for this set and an entropy value can subsequently be calculated.

This concept can be extended for data sets that have a joint probability distribution. The joint entropy is defined as:

$$\boldsymbol{H(A,B) = -\sum_{a\in A}\sum_{b\in B} p_{AB}(a,b)\log p_{AB}(a,b)} \qquad \textbf{Equation 4.6}$$

where $p_{AB}(a,b)$ is the probability of an event occurring in both signals A and B. For two images, the pixel intensity values existing in the two images would first be normalized to the same range of N discrete samples. A 2D histogram can then be constructed to determine the number of corresponding intensities in the two images. The 2D histogram can be viewed as a 2D graph with the horizontal and vertical axes each divided into N sections, resulting in a total of $N \times N$ subsections (bins). This is shown in Figure 4.3, which shows the 2D histogram (logarithmic scale) obtained using an MR and a CT image from the same patient. Pixel intensity values in the two images are first normalized to a range of 0 to 256. Corresponding pairs of intensity values are obtained from identical spatial locations from each of the images. The MR intensity value of the corresponding pair is plotted on the horizontal axis and the CT value on the vertical axis. Each location of the 2D histogram plot represents the total number of corresponding pairs detected from the two images. Zones with a low colour temperature (blue) are indicative of a low count of corresponding pairs of pixel intensities in the two images, while zones with a high colour temperature (red) denote a high number of corresponding pairs of pixel intensities.

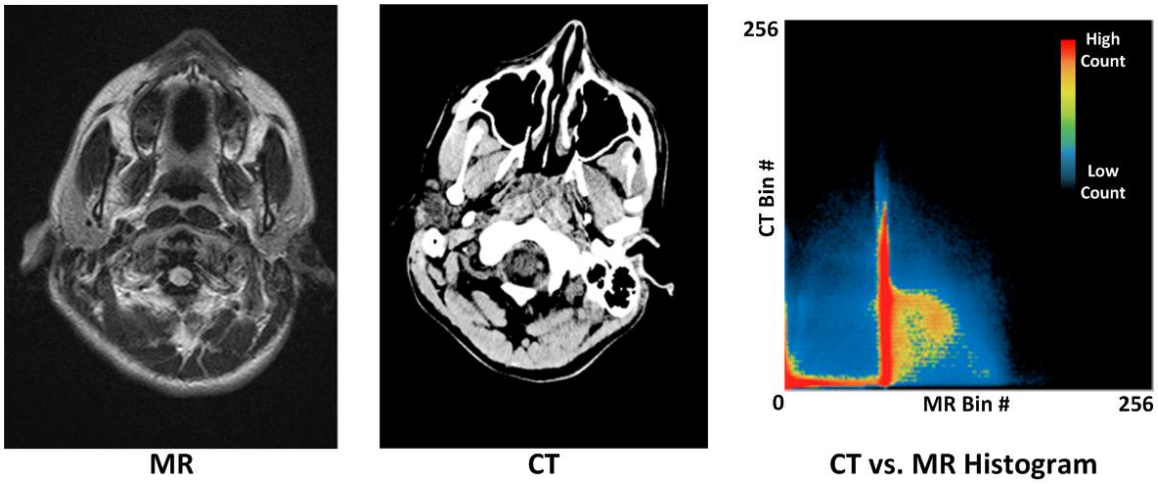

Figure 4.3 - 2D histogram computed using MR and CT images

The joint distribution is computed by comparing values at the same location in each image. Therefore, the 2D histogram obtained will differ when the spatial location of pixels in one image is changed by a transformation such as a rotation. This property can be useful when comparing the re-alignment during registration. In Figure 4.4, the joint histogram of a three-dimensional T1-weighted MR image is computed against a copy of itself with rotations of 0, 1, 2, 3, 4, 5, 10 and 20 degrees applied in the axial direction [7].

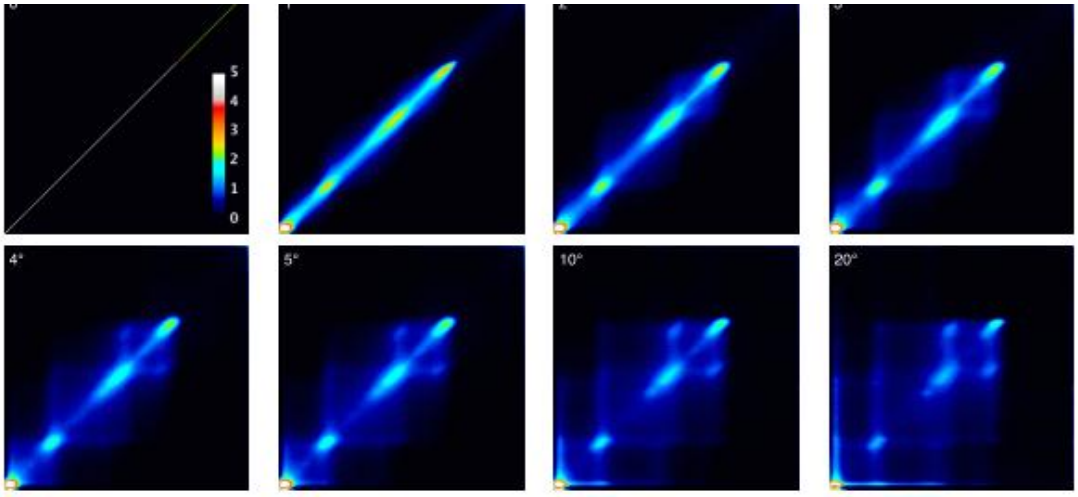

Figure 4.4 - Joint histogram of MR images with various rotation angles applied

Image Source: D. Adler **[7]**

With increasing misalignment, we can observe that there is substantial scattering of discrete observations. This results in a high joint entropy value. In the ideal case (no rotation and no other misalignment), the calculated joint entropy is minimal. The mutual information of the two images can be calculated using Equation 4.7, where *H(A)* and *H(B)* are the marginal entropies of image *A* and *B* respectively.

$$\boldsymbol{MI(A,B) = H(A) + H(B) - H(A,B)} \qquad \textbf{Equation 4.7}$$

In 1995, Viola et al. [17] and Collignon et al. [56] introduced a method of applying the mutual information metric to register sets of images; the marginal entropies of the images are maximized while the joint entropy is minimized. The maximization of MI is indicative of significant correspondence between the images. However, as mentioned by Hill et al. [54], changes in overlap of very low-intensity voxels can disproportionately contribute to the mutual information. As shown by Studholme et al., the mutual information metric value may increase [57] in such cases. In response to this issue, they introduced the normalized mutual information (NMI, shown in Equation 4.8), a metric that is less sensitive to the changes in overlap.

$$\boldsymbol{NMI(A,B) = \frac{H(A) + H(B)}{H(A,B)}} \qquad \textbf{Equation 4.8}$$

Since mutual information and normalized mutual information consider the entropies of the images and do not presume a linearity of differences between them, they can be used for multi-modal registration, e.g., CT to MR or PET to MR. The computation of MI or NMI on a GPU is a non-trivial task. There are a number of issues that have to be factored in when implementing the calculation of the metric using a graphics processor. Those are discussed in section 4.6 of this chapter and also in chapter 5. Barring those

limitations, after loading the input volumes as textures on GPU memory, voxels at the same location within the volumes can be retrieved and compared. The resulting histogram can then be either stored as a texture or on the GPU global memory pool. The marginal and joint entropies can then be calculated by partial reduction techniques of the generated histogram.

4.5 Objective Optimization

One important step of the registration process is the generation of transformation parameters that will be applied to the moving image at every iteration. Image registration presents a non-linear problem. For rigid registration, changes in six parameters (three each for translation and rotation about each coordinate axis) result in non-linear variation of the similarity correspondence between the input and output images. This complexity increases for affine registration due to the addition of three more parameters for scaling of the images. The parameter space of valid transformations is extremely huge and it would not be reasonable to try every possible combination to find a solution to the registration process. In this project, a local optimization method is used for the registration process. The optimizer is concerned with finding a maxima or minima through the repeated use of an objective function that tries to find an optimal set of transformation parameters. Those transformations are then applied to the whole image.

One such technique is the Nelder-Mead downhill simplex method [58], which has been used in various medical image registration implementations [7, 18, 20, 59, 60]. This is a minimization method that operates on the multidimensional parameter space of possible transformations. A simplex is a geometric construct. For a problem with n

degrees of freedom, the simplex will consist of $n+1$ vertices. When two degrees of freedom are available, the simplex will have three vertices and is thus a triangle. The complexity of the shape of the simplex grows with the number of vertices. Under the Nelder-Mead algorithm, the objective function is evaluated at every vertex corresponding to a set of parameters. Combinations of reflections, expansions and contractions are used to crawl through the simplex until a function minimum is reached.

The downhill simplex optimization method requires the storage of multiple variables. This includes vertices, edges, heights and other parameters associated with the transformations being evaluated. The storage requirement is of the order N^2 for an N-dimensional minimization [58]. Some of those variables relate to the previous or current state of the optimization process. Due to those limitations, a GPU implementation of this method is non-trivial.

4.6 Image Registration on Parallel Computing Architectures

As of December 2011, there have been few published methods of performing a complete registration procedure using parallel computing architectures and more specifically GPGPU interfaces. As discussed in Chapter 5, testing and validation of the designed OpenCL based image registration tool was primarily performed using the normalized mutual information metric. Due to this, the following review focuses on various techniques of calculating the mutual information metric, which is then used for medical image re-alignment.

Prior to the introduction of CUDA, S. Chan [20] and D. Adler [7] investigated the use of the specialized OpenGL interface in combination with GLSL to accelerate medical

image registration. Their research led to the development of software that could register sets of 3D image volumes rapidly and accurately. Tests conducted on RIRE image sets [22] resulted in an average registration time of 11.8 secs for CT to MR with a median error equal to the planar dimensions of one image slice. In the case of PET to MR registration, mean alignment time was 3.4 seconds with a median error of half the slice thickness of the PET images [7]. This implementation incorporated the groundwork for a modular tool that enabled different implementations of the components shown in Figure 1.2 to be easily modified or swapped depending on the input parameters and desired output. The OpenCL based image registration techniques presented in this thesis are primarily based on this GPU implantation.

In 2007, Shams and Barnes presented a method of computing mutual information on CUDA compatible hardware [59]. At that time, the version ("Compute Capability 1.0") of CUDA available did not support atomic (uninterruptible) read-write operations to the GPU's global or local memory buffers. To work around this issue, their method involved computing partial histograms and storing the results on global memory. By making use of specialized threading techniques, they were able to mimic atomic operations. Using the same GPU threading techniques, they computed the final MI value through the reduction of the partial histograms. One item of note with this implementation is that they limited the size of the computed joint histogram to 10000 bins. This resulted in a computational gain of 21 to 25 times over the equivalent CPU implementation. Using simplex optimization, they obtained an 8.6 seconds registration time for two 3D volumes each containing approximately 7 x 10^6 voxels [59]. In 2009, Shams et al. applied a similar mutual information based registration method to re-align

RIRE image sets [18]. They made use of three different methods of computing mutual information along with a Powell optimizer to generate their transformation parameters. The stated range of interest of MI computations was 100 to 256 bins for the individual images. Using the three MI calculation methods, the throughput values using 100 bins were approximately 250, 200 and 150 mega voxels/s, while the values using 256 bins were 75, 150, 40 mega voxels/s. Those represent 3.33, 1.33 and 3.75 fold differences respectively in using the different bin sizes. The average execution times obtained ranged from 0.83 to 3.77 seconds. In the best case, this represented a 50-fold improvement over the CPU version. Although the authors present performance values of MI computation using 0 to 256 bins for each image, they do not clearly state the number of bins used when performing the RIRE image registrations.

Recent revisions (versions 1.1 and above) of CUDA have included support for atomic operations. In 2009, Jung and Wesarg [60] took advantage of this feature to implement a NMI based registration method similar to the one presented by Shams et al [59]. They implemented two methods computing the probability distributions. The first involved independently computing the 1D and 2D histograms, while the second method built the 1D histogram from the data contained in the 2D histogram that is computed first. Unlike Shams, they used a higher bin count and did not down-sample any of the histograms; 256 bins for each image, resulting in 65536 bins for the joint distribution. However, image quality was compromised due to the use of nearest neighbour interpolation. The authors mention that this was done so that the interpolation technique matched the one used for their CPU reference implementation. In doing so, they had comparable results for both implementations. Registration tests were conducted on a

limited set of RIRE image sets (only four CT to MR registrations) and did not make a distinction between the two methods of histogram calculation mentioned. The average registration obtained was 10.8 seconds for the GPU implementation versus 66.7 seconds for the CPU version. This represented an average six-fold speedup of using the graphics processor.

The Cell processor is a specialized type of MIMD processor used in the Playstation 3 video gaming system. It shares some architectural similarity to contemporary CPU but has a very fast parallel processing pipeline similar to GPUs. Clusters of this processor have been used by the U.S. Department of Defence for high speed processing. Ohara et al. [61] used mutual information based on algorithms in the ITK software package [2] to implement a linear image registration using the Cell processor. Their test data included 97 image sets of clinical MRI and CT volumes obtained from the Mayo Clinic. The image dimensions were 256x256 or 512x512 pixels. The number of slices contained in those volumes ranged from 12 to 379. The worst case execution times are presented; 1.6 seconds for registration with a fixed volume size of 256x256x56 and 20 seconds for registration with a fixed volume size of 512x512x379. Those results were obtained by executing the registration on two Cell processors, which were be linked at the hardware level; this allowed the 16 internal cores (eight per processor) to work in parallel.

The OpenCL registration tool implemented as part of this thesis builds upon the various aspects of the accelerated image registration methods presented above. In the next chapter, the methodology used to construct the OpenCL image registration tool is discussed.

Chapter Five: **Methodology**

5.1 Existing Image Registration Software

The review of the image registration on parallel computing architectures (presented in Chapter 4) has led to the:

- Identification of the algorithms that could be converted to an OpenCL based implementation.
- Determination of the image types that have to be supported such that test results of different implementations can be compared against each other.
- Documentation of published failure modes of different registration methods or parameters.
- Determination of a baseline for the initial state of the project based on S. Chan's [20] and D. Adler's [7] CPU and GPU registration tools.
- Evaluation of the image visualization methods available in the Imaging Informatics Lab and determination of improvements that can be made.
- Identification the set of components necessary to construct a complete affine medical image registration tool.
- Identification of published registration methods that can be used for testing and/or validation of OpenCL based registration.

An OpenCL medical image registration tool was constructed and tested based on the findings of the review process conducted. A modular approach was undertaken such that the components constructed could be individually tested or re-used as part of other systems.

5.2 Implementation of Algorithms using OpenCL

5.2.1 Choice of Memory Access Methods

The choice of memory access pattern can affect the computational speeds achieved on a GPU. For the implementation of image registration using OpenCL, three different memory access methods were available:

- Use of global GPU memory pool.
- Use of read-only image memory.
- Use of read-write image memory.

As an initial test of the usability of OpenCL, registration was implemented with SSD and SAD metrics using only global GPU memory. At this stage, basics tests using synthetic image data were conducted to validate the correctness of the registration procedure. Subsequently, both methods of using texture memory (read-only and read-write) were utilized to perform registration with the SSD and SAD metrics. The difference in processing speeds using the three types of memory access methods was then investigated.

Using the results from the SSD and SAD registration tests as a guideline of possible computational performance, the cross correlation and mutual information metrics were implemented using a combination of all the three methods of memory access. As discussed later in this chapter, the combined use of different memory access techniques was done as a compromise to achieve high computational performance due to current hardware limitations.

5.2.2 General Structure of OpenCL based Algorithms

The parallelized algorithms implemented using OpenCL were designed to perform the same operations as existing sequential CPU algorithms using the same input. The pseudocode below outlines how those two different processing methods can be used to apply a rotation to a 3D volume containing a total of N voxels.

Sequential algorithm for applying a rotation to a 3D volume (CPU version)

1. *Construct rotation matrix to be applied to each voxel*
2. *Loop: For each voxel in the 3D volume*
    - 2.1. *Retrieve the coordinates of current voxel from computer memory*
    - 2.2. *Retrieve rotation matrix from computer memory*
    - 2.3. *Apply rotation matrix to voxel coordinates*
    - 2.4. *Save new coordinates back to computer memory*

   End For loop after N voxel coordinates computed

OpenCL parallelized algorithm for applying a rotation to a 3D volume (GPU version)

1. *Construct rotation matrix to be applied to each voxel*
2. *Copy rotation matrix to GPU's local memory*
3. *Call OpenCL function to setup and run GPU compute units for N simultaneous operations*

   (Each streaming processor within the compute units executes the following steps)
    - 3.1. *Retrieve unique identification values corresponding to a pixel location*
    - 3.2. *Retrieve a voxel's coordinates from 3D image memory using unique*

identifier

3.3. Access rotation matrix stored in local compute unit's memory

3.4. Apply rotation matrix to retrieved voxel coordinates

3.5. *Save the new coordinates back to 3D image memory*

In the example above, the GPU algorithm requires two extra steps as compared to the CPU version. The first item to consider is the optimizer that runs on the CPU. After geometric transformations have been generated, those parameters have to be loaded onto the GPU. The copy operation used directly inputs the transformation matrix into the local memory of each of the GPU's compute units instead of the global memory pool. This enables rapid access to the matrix by the streaming processors.

In our affine registration method, the modules responsible for transformation, re-sampling and similarity metric calculation were implemented using the algorithm described in the pseudocode above for the GPU version using OpenCL. However, unlike illustrated in Figure 1.6, the transformation and re-sampling modules were consolidated into a single module. The output from the transformation stage for each voxel can be immediately used to re-calculate the output voxel intensity. By combining the transformation and re-sampling steps, a complete read-write cycle for the entire 3D volume is eliminated – this improves the overall speed of registration. For the re-sampling operation, we made use of the internal re-sampler objects and interface provided in the OpenCL API. The option to perform tri-linear re-sampling was selected – linear interpolations are conducted using eight surrounding voxels to obtain the average pixel intensity at a desired output location.

The calculation of a similarity metric value is slightly lengthier than the generalized pseudocode provided for applying a transformation. Computing the final value of the similarity metric includes one or more extra calculations. In the case of SSD, SAD or NCC, the first step involves calculating the difference or correlation value between two voxels. The subsequent calculations make use of the values generated in the first step to compute a sum or accumulate the values to determine the final metric value.

5.2.3 Synchronization of Parallel Computing on GPU

In the sequential algorithm, a loop is used to iterate through all the voxels. When using a loop on CPU, the previous state and current state of the iterations are retained. This results in the computations being done in sequence from the first voxel to the last. On a GPU, the computation tasks are assigned randomly and asynchronously. When the GPU kernel is setup, the total number of computations to be performed is specified using available OpenCL function calls. The first step to synchronization is to split the number of computations across the total number of compute units available. Each task (one computation per voxel) is assigned a unique identifier which is equivalent to the splits of the workload. The GPU computing architecture guarantees that each task is executed only once as long as the unique identifiers and the correct arguments are used when setting up the kernel. Each streaming processor retrieves one identifier and executes the task to completion. The processors can then be assigned to another task until all identifiers have been used up. The second part to synchronization is to ensure than memory operations do not result in partial memory writes or overwrites. The OpenCL API provides functions that implement barriers for memory access. This enables

synchronization points that operate like semaphores. While one processor is writing to memory, the other processors are required to wait for that process to finish before being able to execute their own write operation.

5.2.4 Calculation of Mutual Information

The calculation of MI and NMI involves two major steps; constructing a 2D histogram and computing entropy values. Due to current hardware limitations, this two-step process cannot be efficiently performed with full parallelization on the GPU. For each iteration of the registration cycle, a complete 2D histogram has to be constructed. To build this 2D map of intensity values, the number of identical intensity values from the two input images are counted. This count is stored at a corresponding location on the 2D map; the count has to be incremented every time an identical set of intensities is found. The most efficient method to store this map would be to use the local memory of each compute unit. However, this was not possible since local memory for the GPU used is limited to 32KB, while 256KB is required to store the full 2D histogram containing 256^2 bins. Current GPU architecture does not allow for data sharing between compute units. Any method of splitting the histogram on local memory would eventually require data to be copied back to global memory multiple times, depending on memory usage; this can significantly impact computational performance. To eliminate the issue of having to track memory usage, the joint histogram was created on the GPU's global memory. Atomic (uninterruptible) write operations are then executed to prevent partial memory updates, which would lead to incorrect statistics being calculated.

Implementation of the mutual information metric was done after obtaining timing results for the SSD and SAD implementations. The computation times obtained for those metrics showed that performing accumulations using GPU global memory is slow as compared to using texture memory to read and write the computed sums. Due to this limitation, the 2D histogram is copied out of GPU memory to regular computer memory. The last step of generating 1D histograms for both volumes and calculating entropy values is done on CPU.

5.2.5 OpenCL Implementation against Baseline OpenGL Implementation

The registration codebase from S. Chan [20] and D. Adler [7] was used as a baseline for the OpenCL implementation. Due to the use of OpenGL for their implementation, some functional components were intertwined with other visualization and graphical elements. The OpenCL implementation is distinctive in that visualization and graphical components are separated from the four main registration modules shown in Figure 1.2.

The most significant differences were present in the transformation module. Within the OpenGL API, built-in functions are provided to apply translations and rotations to image textures. In OpenCL, those functions are not available. For consistency with the homogeneous coordinate system used in computer graphics, OpenCL kernels were implemented to replicate the missing functionality. To apply a transformation, pixel coordinates and a transformation matrix are first input into the kernel. In the case of a translation, a simple vector addition of the coordinates to the translation vector is performed. The output coordinates are then saved in the specified output location (buffer

memory or image read-write memory). Similarly, for a rotation, the input pixel coordinates are multiplied with the rotation matrix to generate output coordinates.

Another difference can be found with the similarity verification module. For some of the metrics, a sum or accumulation of values is necessary. OpenGL incorporates a functionality called blending which can be used to compute sums of values present within units of texture memory. In general, this is used for image transparency and opacity calculations. The OpenGL implementation was able to use this function to accumulate values due to the special structuring of data done by S. Chan and D. Adler when inputting the images on texture memory. Since similar functions are not available in OpenCL, separate kernels were created for performing accumulations. The approach used involved split summations as described in section 4.4.1.

The implemented functions mentioned above were tested using synthetic brain images.

5.3 Functional Testing using Synthetic Images

Functional tests were performed to verify the correctness of the modules responsible for transformation and resampling. A synthetic MR image (T1 modality) from the Brainweb database [62] was used to perform those tests. The T1 image volume from Brainweb measures 181x217x181 voxels corresponding to an isotropic voxel dimension of 1mm in all planes. When applying transformations, to avoid volume clipping, each image slice was zero-padded to a size of 256x256 pixels. The first part involved applying known transformations (translation and rotation) to the original image and verifying that the corresponding output image was obtained. This was achieved by using the same

functions that the registration procedure uses to apply transformations to the moving image volume. The second part of functional testing involved the registration of those transformed images to the original volume using the three different OpenCL memory access methods. Due to memory constraints and the slow access path to image data using global memory, the tests were conducted using the first 30 slices of the volume. Reducing the number of slices used allowed for validating the functionality of the system in a short amount of time. As mentioned in section 5.2.1, this served as a means to validate the choice of using readable and writable image memory. Additionally, this testing phase also served as a preliminary validation of the registration process producing accurate and usable output images.

5.4 Testing and Validation using RIRE Datasets

The designed OpenCL based affine registration tool was tested for both speed and accuracy using images from the Retrospective Image Registration Evaluation (RIRE) [22] database. Accuracy testing is performed to validate the correctness of the registration implementation and also to provide comparable results to other published methods. The RIRE database is accessible through the World Wide Web and provides 18 patient cases containing CT, MR and PET images. An extrinsic method of registration was used to generate the gold standard transformation values for combinations of CT to MR and PET to MR registrations. Prior to image acquisition, fiducial markers were implanted in the patients' skulls. By physically re-aligning those markers, affine transformation parameters that can be computationally applied to the all the voxels within the volumes were computed. All traces of the markers were removed from the image sets provided

online [6]. In the case of MR images, some geometric distortions may have been introduced at the time of patient scanning. To accommodate for those, the RIRE database also provides corresponding sets of rectified MR images based on a technique developed by Chang et al [63]. The 18 cases are subdivided into two groups of images having a different range of pixel size and number of slices. Group A patients cases are numbered 1 to 9 and group B patients are numbered 101 to 109. The image modalities provided for the image sets are outlined in Table 5.1. The dimensions of the volumes used are provided in Table 5.2.

Image Modality	Patient Number
CT	1 - 7, 102 - 109
PET	1, 2, 5 - 9
PD	1 - 9, 101 - 104
T1	1 - 9, 101 - 109
T2	1 - 9, 101, 102, 104 - 109
PD Rectified	1 - 7
T1 Rectified	1 - 5, 7
T2 Rectified	1 - 7

Table 5.1 - RIRE Image Set Modalities

Group	Modality	Rows	Columns	Number of Slices	Pixel Size (mm)	Slice Thickness (mm)
A	**MR**	256	256	20-26	W: 1.25 H: 1.25	4.00
	MR (Rectified)	256	256	26	W: 1.25-1.28 H: 1.25-1.28	4.00-4.16
	CT	512	512	27 - 34	W: 0.65 H: 0.65	4.00
	PET	128	128	15	W: 2.59 H: 2.59	8.00
B	**MR**	256	256	52	W: 0.82 H: 0.82	3.00
	CT	512	512	40-49	W: 0.40-0.44 H: 0.40-0.44	3.00

Table 5.2 - Dimensions of RIRE Datasets

To validate a registration method using the RIRE test cases, CT or PET images are registered to the MR images. The original and transformed coordinates of eight selected points from the resampled CT or PET volume are submitted to the automated online system. Using the eight pairs of homologous coordinates, the rigid transformations mapping the original coordinates to the transformed coordinates are computed. This eliminates any bias or manipulation of transformations by the database user. Additionally, the automated system also validates that the back-calculated transformations do correspond to rigid transformations only. The computed transformations are then applied to ten voxel coordinates of the moving volume and the deviations of re-aligned voxels from the gold standard are calculated. The mean errors in millimeters from the correct positions are provided for those ten voxels. The exact re-alignment parameters are not available to the database users. However, the RIRE database provides access to and ranks submissions from multiple researchers. The overall

ranking is listed for all submissions in relation to computed mean errors regardless of algorithmic implementation of the registration procedure.

For testing of the OpenCL based registration method, the normalized mutual information (NMI) similarity metric was used. The similarity metric value calculated when using MI or NMI varies depending on the number of histogram bins used. For our experiments, we chose to construct 2D histograms with 256^2 bins and 512^2 bins since those match the original dimensions of the input image sets.

Comparisons are made against the closest-matching best available results on the RIRE website. It is important to note that the comparisons are made only against results that have a full range of test cases for patients of group A. This involved only CT to MR registration for the first seven patient cases (Patient 1 to 7). Group B present a harder set of registration cases due to the increased number of slices and smaller voxel dimensions. Results for this set were not available for the selected best performing methods. Algorithms using a single test case (one single patient) resulting in a high rank due to low mean error were not considered. Such results do not allow for an objective comparison of the overall registration accuracy. As of September 2011, the two highest ranking submissions for the RIRE database were from L. Joskowicz [53], [64] and N. Cahill [65]. Both of their registration methods made use of the normalized mutual information metric, making this metric the preferred choice for testing such that relevant comparisons could be made. L. Joskowicz makes use of a two-step process for registration. Similarity is first coarsely computed using mutual information and the registration process is afterwards refined using deterministic based sampling [53]. The optimizer used is a gradient-based one that is part of the Elastix software package. The RIRE image registrations were

performed on a dual-processor 2GHz computer with 2GB of RAM running the Linux operating system – details about the processor manufacturer and other hardware or software components were not specified. N. Cahill's implementation uses normalized mutual information as described in Section 4.4. Optimization was carried out using functions available within MATLAB's [66] optimization toolbox. More specific details about this implementation was not found.

5.5 Timing Method

Due to the unavailability of functional profilers and debuggers for OpenCL under the Mac operating system, a separate timing procedure had to be implemented to accurately measure computation and complete registration times. A timer was designed to measure the time spent in different modules of the registration system to the closest microsecond. The timings were recorded for all the registration iteration cycles performed on our chosen datasets.

Two different sets of runtimes were measured:

1. **Complete runtime** of the registration procedure including time to load the images on GPU and display the output. This timer is started after images are loaded on computer memory and the user requests to start the registration procedure. It is stopped when the registration output is displayed on screen. Time spent by the user to manually select input images can be highly variable and was not considered as part of the registration procedure. In a clinical system, this process could be automated, thus eliminating this variation. This is exemplified below:

i. Registration tool is started

ii. User selects desired input volumes

iii. Input images are loaded into computer memory

iv. User requests to start the registration procedure

v. Timer is started

vi. Registration objects are created and images loaded on GPU

vii. Computations are done within the registration modules

viii. Output volume is generated and displayed on screen

ix. Timer is stopped

x. User can view and analyze the rendered volume

2. **Computation runtime** spent doing operations within the transformation / re-sampling module and the similarity metric calculation module. The timer is started when the GPU is set up and started for desired operations. After the completion of computations for one iteration, the timer is stopped. The sum of the recorded times is computed for all the iterations performed for the complete registration procedure. This is exemplified below for one iteration:

   i. Operations are performed on CPU

   ii. GPU kernel is setup and started for computations

   iii. Timer is started

   iv. GPU operations are performed for the whole volume

   v. Results are copied out of GPU memory if required

   vi. Timer is stopped

vii. Other CPU operations are performed

5.6 Experimental Setup

5.6.1 Operating System and Hardware Specification

The operating system used was Apple Mac OS X 10.7. Three processors were used for development and testing, namely:

- AMD Radeon HD 5870 (desktop GPU)
- AMD Radeon HD 6750m (laptop GPU)
- Intel Core i7-2720QM CPU (laptop CPU)

The complete set of RIRE tests was conducted using the AMD Radeon HD 5870 GPU under Mac OS X 10.7. This processor has 20 compute units with a total of 1600 stream processing units. The maximum clock frequency is 850MHz and maximum GPU memory is 512MB. The peak memory bandwidth is 153 gigabytes per second (GB/s). The HD 6750m processor was mainly used for debugging and ensuring portability of the OpenCL implementation. It has 6 compute units with a total of 480 stream processing units. The maximum clock frequency is 600MHz and maximum GPU memory is 512MB. The peak memory bandwidth is 57 GB/s. The Intel Core i7 CPU has 4 processing cores. Due to Intel hyper-threading technology, the OpenCL API detects those cores as 8 compute units. Due to the differences of the CPU and GPU architectures, the CPU compute units can be regarded as the being equivalent to the GPU stream processing units. The maximum clock frequency of the CPU is 2.2GHz and 8GB of memory was available for data storage. This CPU is capable of handling a peak memory bandwith of

25 GB/s. The Core i7 CPU was used for test cases that provided results for objective comparison of performance of the image registration procedure.

5.6.2 Numerical Precision

The numerical precision of calculations across the different platforms was kept identical through the use of single precision floating point numbers. For all the processors, the number representation conforms to the IEEE-754 standard for floating-point arithmetic [67]. A discussion of the effect of numerical precision on the calculations performed is made in section 6.3.1.

5.7 Project Limitations

In addition to the memory access limitations discussed for the calculation of mutual information, the following items limited the scope of the work accomplished for this thesis project.

5.7.1 OpenCL 1.1 Support

The main limitation of this project has been availability and access to OpenCL 1.1 compliant hardware and software. Currently, CPU and GPU manufacturers have varying levels of support for the OpenCL standard. To enable the use of OpenCL 1.1, software drivers have to include the necessary interface for this particular version; software drivers handle the interaction between the operating system and the processing hardware. The primary development platform was the Mac operating system. On this OS, only very recent AMD Radeon GPUs and Intel CPUs have the appropriate software drivers.

5.7.2 Image Slice Contiguousness

Depending on acquisition settings, patient data can be collected as contiguous or non-contiguous image slices. This can be inferred from metadata usually associated with collected volumes through either a pixel spacing or slice thickness tag. The spacing between slices can affect the output of the registration process. For very large slice spacing values, it would be more prudent to perform a 2D registration procedure. In such cases, 3D interpolation of the transformed image coordinates could result in an inaccurate representation as compared the original image. In this project, the available tested sets of images only contained contiguous slices. In the case of non-contiguous slices, the slice spacing is considered to be equivalent to slice thickness and therefore the images are considered to be contiguous. If there is high anisotropy within the input volume, it is expected that the registration result might not be accurate. To work around this issue, extra control logic could be implemented to determine whether 2D or 3D registration would be more appropriate. This would involve determining the trade-off between voxel dimensions, slice spacing or slice thickness. Designing the latter algorithms falls out of the scope of this project and was therefore not investigated.

5.7.3 Choice of Optimizer

The optimizer used for registration is the Nelder-Mead downhill simplex method [68] running on the CPU. As mentioned in section 4.5, this non-linear optimization method has well documented use cases for medical image registration and was therefore chosen for the presented image registration implementation. By using a multi-resolution approach, the optimization is first performed on downsampled versions of the input

images. This minimizes the chance of converging to local minima. Additionally, this enables iterations at a higher resolution to be started with parameters closer to the final solution. The GPU implementation of this method would be highly complex. Due to the storage requirements of the simplex and its required parameters, specialized functions to synchronize GPU threads to efficiently calculate the optimizer output would be required. While the CUDA library allows lower-level access to the GPU hardware, making those tasks possible, the same functions are not currently available within OpenCL version1.1. Coupled with the lack of appropriate tools for OpenCL debugging and profiling, trying to implement the downhill simplex method or a re-designed similar method on GPU would fall well outside the scope of this project. In fact, optimization theory is a distinct mathematical field.

For the purpose of this project, the optimizer was considered as a black box. The available source code for this optimizer was obtained from D. Adler [7] and was assumed to be completely functional and reliable. Its implementation follows the algorithm described in [58] along with modifications formulated by Shekhar and Zagrodsky [69]. A direct relationship can be established between the deformations (reflection, expansion and contraction) applied to the simplex and the transformations applied to the moving image. The parameter space of the simplex is first normalized to match that of the input images. This results in a unit step along any parameter axis being roughly equivalent to physical displacement of the input volumes in the spatial domain. The convergence of the optimizer to a solution is determined by a tolerance value denoted by the smallest allowable relative height difference between the highest and lowest vertex of the simplex. In the downhill simplex optimizer implementation used, the convergence tolerance could

be varied by a factor of 10 with respect to the normalization performed. Prior to testing across the whole range of RIRE datasets, preliminary tests using a subset of available images were performed to determine the best possible tolerance value for registration. For the OpenCL GPU implementation, when using 512^2 bins, the tolerance had to be reduced by a factor of 10 as compared to using 256^2 bins. For the OpenCL CPU and regular CPU tests, the same tolerance value of OpenCL GPU registration with 512^2 bins was used. This was necessary to maximize the number of valid registrations. Even the refined choice of height tolerance, for a minimal number of cases, the resulting transformed image was visibly inaccurate or no output was obtained since convergence was not achieved. Such cases were not taken into account during analysis of results obtained.

Chapter Six: **Results and Discussion**

In this chapter, the results of the registration tests conducted using the RIRE datasets are presented. The first section covers the accuracy results obtained for all the cases and the second section presents the corresponding timing results. Using the OpenCL implementation on a single patient for CT to PD registration, a lowest mean error of 0.734mm was obtained. This resulted in a biased highest-ranking result over all the sets of CT to PD patient cases. The results presented in this chapter refer to group A cases unless otherwise specified. Additional results for group B patients can be found in the Appendix section. Extensive discussion for this set of test cases could not be made due to the lack of information for objective comparison.

6.1 Functional Testing Results

6.1.1 Translation and Rotation Results

Rotations were applied about the z-axis of the MR image volumes. Translations were applied to each image slice along the x and y axes. Figure 6.1 shows the one of the original input image slices along with resulting output slice after applying transformations. The values of rotation and translation are annotated to the corresponding images. The results from those tests confirmed that the translation and resampling modules were functioning as expected.

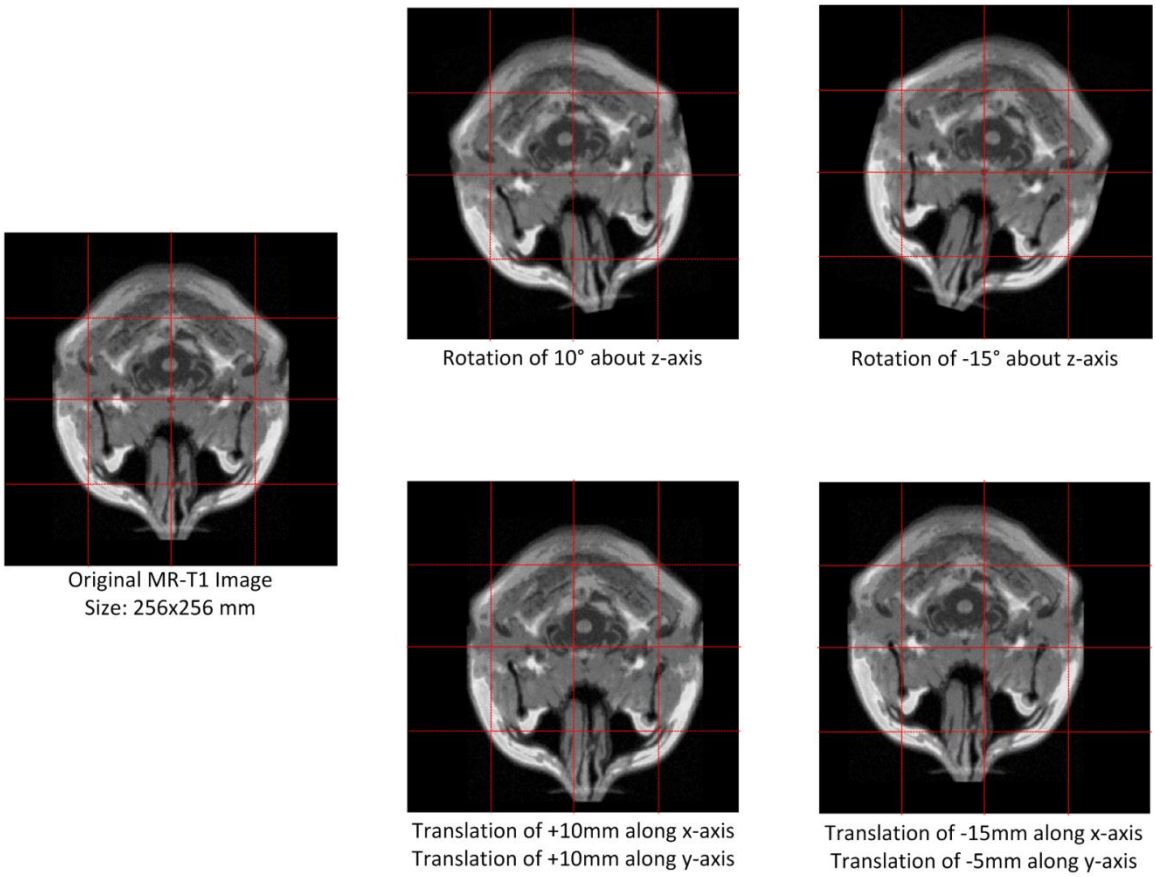

Figure 6.1 - Application of Transformations to a MR-T1 Image

6.1.2 Optimal Memory Configuration Results

Table 6.1 shows the results obtained for registration of the transformed image volume to the original volume. The timing results obtained using the different memory access methods are also presented. Tx, Ty and Tz denotes translations along the x, y and z axes respectively. Rx, Ry and Rz denote rotations about the x, y and z axes respectively. The values obtained show that the registration transformations result in a correct re-alignment of the transformed volume against the original volume. The computation times obtained show that the best performing memory access method involves making use of read-write

image memory. The computation time obtained using read-only memory is slightly higher than when using global memory. The reason for this is due to the need of copying and reconstructing a read-only intermediate image for every iteration of the registration cycle. Intermediate values have to be first written to global memory before being re-formatted as a read-only image. This introduces the noticeable overhead time, thus accounting for the larger computation time. With the use of read-write image memory, this overhead is eliminated since write operations are executed directly to the special image memory location.

Applied Transformations	Rz = 10°	Rz = -15°	Tx = 10mm Ty = 10mm	Tx = -15mm Ty = -5mm
Registration Transformations	Rx = 0.00° Ry = 0.00° **Rz = -10.00°** Tx = 0.50 mm Ty = 0.50 mm Tz = 0.50 mm	Rx = 0.00° Ry = 0.00° **Rz = 15.00°** Tx = -0.50 mm Ty = -0.50 mm Tz = -0.50 mm	Rx = 0.00° Ry = 0.00° Rz = 0.00° **Tx = -10.50 mm** **Ty = -10.50 mm** Tz = -0.50 mm	Rx = 0.00° Ry = 0.00° Rz = 0.00° **Tx = 14.50 mm** **Ty = 4.50 mm** Tz = 0.50 mm
Computation Time (seconds) using global memory	20.08	16.71	26.12	23.35
Computation Time (seconds) using read-only image memory	26.36	25.20	27.94	21.29
Computation Time (seconds) using read-write image memory	3.01	3.54	3.31	3.44

Table 6.1 - Functional Testing using Synthetic Images

6.2 Accuracy and Speed due to Optimizer

The accuracy and speed of image registration is driven by the chosen optimization method and the convergence criteria used to determine correct re-alignment. Components of the registration system, such as the calculation of the similarity metric, can limit the efficiency of the optimizer's algorithm. In the registration system presented in this thesis, the value of the similarity metric is one of the conditions used by the optimizer to generate transformation parameters.

The choice of the convergence tolerance used is a compromise between speed and accuracy. Generally, a small tolerance value will increase accuracy and decrease speed while a high tolerance will have the opposite effect. However, for small number of cases, a low tolerance resulted in an inaccurate solution being produced. This behaviour was observed when conducting unit tests on a sample of the RIRE datasets using the Radeon HD6750m GPU to determine the convergence value to be used for the entire set of images. Factors conferring to those solutions were non-deterministic. This led to the decision of choosing a larger tolerance being used for registration with 512^2 bins. This same value was used for CPU based registration as a compromise for the long registration time and to limit the number of potential non-converging cases. Table 6.2 shows the transformation parameters obtained for registration of CT to T1-weighted MR images for patient 5. When using a relative tolerance of 10^{-5} or 10^{-6} for OpenCL GPU registration using 256^2 bins, the transformation parameters obtained are equivalent. Differences of less than one degree for rotation or less than 1mm for translations represent transformations of less than one pixel deviations in image space. Therefore, those variations are not significant. Using 512^2 bins for OpenCL GPU registration, differences

as large as six degrees for rotation and 2.7mm for translation were observed – accuracy was worse when using the lower tolerance value. The effect of those transformation values can be seen in Figure 6.2. The most likely explanation for this result is that the optimizer detected a local minima and hence converged to a wrong solution due to the tighter constraints. The convergence metric was not modified or controlled over the entire set of RIRE registration test cases. The best relative tolerance was used and kept the same in relation to the results obtained for patient 5. Choice of patient 5 was done at random and used as a sample representative of all the data sets. Using the smaller tolerance of 10^{-6}, convergence was achieved in approximately 400 iterations for both CPU and GPU. Using a larger tolerance of 10^{-5} on GPU resulted in approximately an additional 100 iterations for convergence to a solution.

	256^2 bins		**512^2 bins**	
Relative Tolerance	**10^{-5}**	**10^{-6}**	**10^{-5}**	**10^{-6}**
Rx (degrees)	-1.834	-1.896	-1.187	6.212
Ry (degrees)	-0.474	-0.414	-0.090	3.982
Rz (degrees)	-0.450	-0.488	-0.514	3.743
Tx (mm)	-5.989	-5.989	-6.186	-6.269
Ty (mm)	-32.883	-32.920	-32.691	-29.029
Tz (mm)	-21.001	-20.938	-21.220	-21.151

Table 6.2 - Transformation Parameters for Varying Convergence Tolerance

(Rx, Ry and Rz are rotations about the coordinate axes.
Tx, Ty and Tz are translations along the coordinate axes)

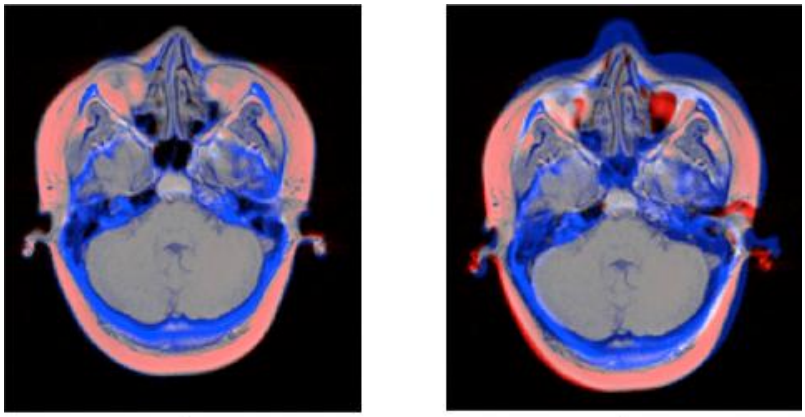

Figure 6.2 - Registration of CT to MR using 512^2 bins with different tolerance

Images A and B show the outputs obtained using a relative tolerance of 10^{-5} and 10^{-6} respectively. B has a visible misalignment of the moving image (blue coloured).

6.3 Affine Registration Accuracy

6.3.1 CT to MR

The mean errors obtained over the range of possible CT to MR registration combinations for group A using 256^2 and 512^2 bins are presented in Table 6.3.

Modality *(CT to)*	OpenCL (256^2 bins)			OpenCL (512^2 bins)	
	Mean Error (mm)	**Standard Deviation (mm)**		**Mean Error (mm)**	**Standard Deviation (mm)**
PD	1.829	0.843		2.085	0.912
T1	1.845	0.844		1.815	0.834
T2	1.970	0.926		2.058	1.119
PD Rectified	1.830	0.551		1.766	0.494
T1 Rectified	2.213	0.717		2.219	0.410
T2 Rectified	2.031	0.803		2.055	0.592

Table 6.3 - Accuracy of CT to MR Registration using OpenCL

The overall mean error obtained for CT to MR registration was 1.953 ± 0.131mm using 256^2 histogram bins. This value was 2.000 ± 0.273mm for registration using 512^2 bins. The results obtained indicate that using a larger number of bins did not produce more accurate CT to MR registration. Additionally, using the rectified images did not result in a visibly more accurate registration. The RIRE test results do not specify the spatial direction of the recorded errors from the correct position. If the largest errors are assumed to be in the planar slice direction, approximately 85% of the values obtained are well under 2.5mm. This represents an accuracy of less than two pixels. However, from visual inspection of the output images, it could be noted that the misalignments obtained in this plane are generally closer to an error of one pixel. For the test cases, the relatively large slice thickness as compared to the width or height of a pixel (4mm vs. 1.25mm)

affects the accuracy of registration. This is attributed to the interpolation method used to calculate output voxel values. The process of interpolation relies on the spatial location of the eight neighbouring voxels to apply weights to the contribution of each of those voxels. Therefore, the depth will have a larger contribution to the calculated intensity value than the width or height of a voxel. When slice thickness is used for accuracy comparison, all the recorded registration errors are well under one voxel of misalignment. One of the hypotheses of this project was that a registration error less or equal to the largest voxel dimension could be achieved. With a mean error of approximately of 2mm, this condition has been met since this value is half the largest voxel dimension of 4mm. As mentioned by Lee et al. [3], a target registration error (mean error) of less than the largest voxel dimension is considered as successful registration for CT to MR images.

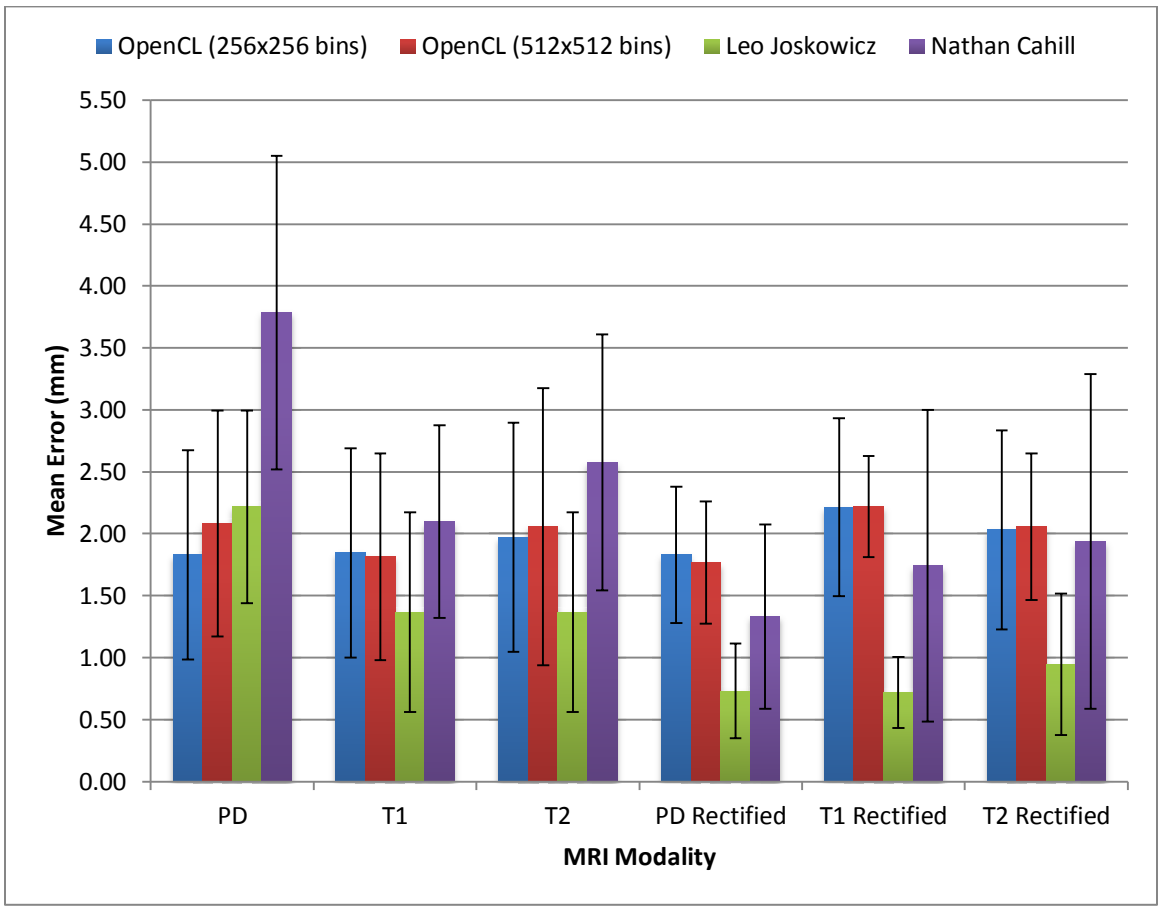

Figure 6.3 - Comparison of RIRE test results for CT to MR registration

Figure 6.3 provides a comparison of the mean alignment errors obtained using OpenCL against those published by L. Joskowicz and N. Cahill on the RIRE website. The error bars represent the standard deviations of the alignment errors across the set of voxels used to calculate the means. The results obtained for the OpenCL based registration compares favorably to those two implementations. For the rectified volumes, the spatial mean errors using the method presented in this thesis are higher than those obtained by the above named researchers. The deviations obtained for each of the image modalities are less than one millimetre, which is less than the dimensions of a pixel in one MR image slice. The only exception was for CT to T2 registration using 512^2 bins

for which a standard deviation of 1.119mm was obtained, but this is still less than the value of 1.25mm for a pixel's dimension. For CT to T1 rectified registration using 512^2 bins, the error bars are visibly smaller than other cases. This is due to the smaller data set available for this observation; only four of the patient cases produced usable registration results. The standard deviations obtained when using rectified volumes was lower than using the non-rectified volumes. Those results are similar to L. Joskowicz's. However, N. Cahill's method did not follow the same pattern. For CT to PD, T1 rectified and T2 rectified, the standard deviations were larger than one pixel. In addition, the rectified volume registration deviation was lower only for the PD case; this value increased for both T1 and T2 cases.

As mentioned in Chapter 5, L. Joskowicz and N. Cahill both make use of the normalized mutual information metric. However, differences in some of the algorithms used by those two researchers may account for some of the observed differences in mean errors obtained as compared to the OpenCL implementation. In the case of L. Joskowicz's method, an additional computational operation is performed after the calculation of NMI to increase the accuracy of registration. This involves calculation of image histograms for the NMI using curvelet-based masks or filters to improve the precision of estimating similarity between the input and transformed volumes. Overall, N. Cahill results show visible fluctuations not seen in the other presented results. Due to lack of direct information about his implementation, it is not possible to comment on the observed accuracy. The slight increase in performance of N. Cahill's method for the rectified volumes could be attributed to the use of a learned similarity measure for local calculation of NMI as described in [3]. In this implementation, machine learning

algorithms are used on a training set of images to improve the similarity function for the actual test image sets. For the OpenCL implementation, the NMI is calculated globally for the reference and transformed volume in one step. No refinements are made to limit the similarity measure to local zones showing varying levels of misalignment. Since the rectified volumes are geometrically corrected versions of the normal scans, the calculated entropy of the both sets of images will be very similar. This accounts for the closely identical mean registration errors obtained for normal and rectified image counterparts. As illustrated in Figure **6**.**4**, the histograms for one input set of images show that a nearly similar intensity distribution when using a normal PD image or its rectified version. Although there is a slight spatial shift of the distribution for the rectified image, the entropy calculation is virtually unaffected by this factor.

Another factor that could account for observed differences is the numerical precision used for the registration process. Numerical arithmetic inaccuracies compounded to modification of transformation values across many iterations can lead to a less precise final solution. As mentioned in Chapter 5, the OpenCL implementations (both GPU and CPU) conform to the standard IEEE-754 single precision floating point arithmetic standards. This ensures the recorded OpenCL accuracies are consistent across those platforms. In the past, not all GPUs conformed to the IEEE standard. Currently, most high-end consumer GPUs (including the AMD Radeon HD 5870 used for this project) abide by the standard for single precision calculations due to the increasing popularity of general purpose computing on GPUs. However, low-end GPUs (e.g. laptop processors) may not have that compliance. When very small convergence tolerances are used, double precision operations can enable the detection of small changes. This lead to

an improvement of accuracies obtained. However, the computational time of registration would increase, thus mitigating the benefit of using parallelism through the OpenCL API. Currently, only professional grade GPUs support the double precision standard. Hence, using double precision on the processors available for this project could lead to inconsistent results across platforms. Those graphics card cost over $4000 and therefore are not appropriate for a low cost and high computational benefit as demonstrated by using OpenCL on consumer level GPUs which cost under $500. Information about the numerical precision used by L. Joskowicz and N. Cahill was not available. If double precision arithmetic was used for their registration procedure, then the presented OpenCL using single precision arithmetic demonstrates very high accuracy of the observed mean errors.

As mentioned previously, accurate registration is limited by the optimizer and factors that affect the generation of valid transformation parameters. Even though the OpenCL based registration method is highly accurate, its precision could be improved by using a more sensitive similarity metric calculator along with a refined optimizer.

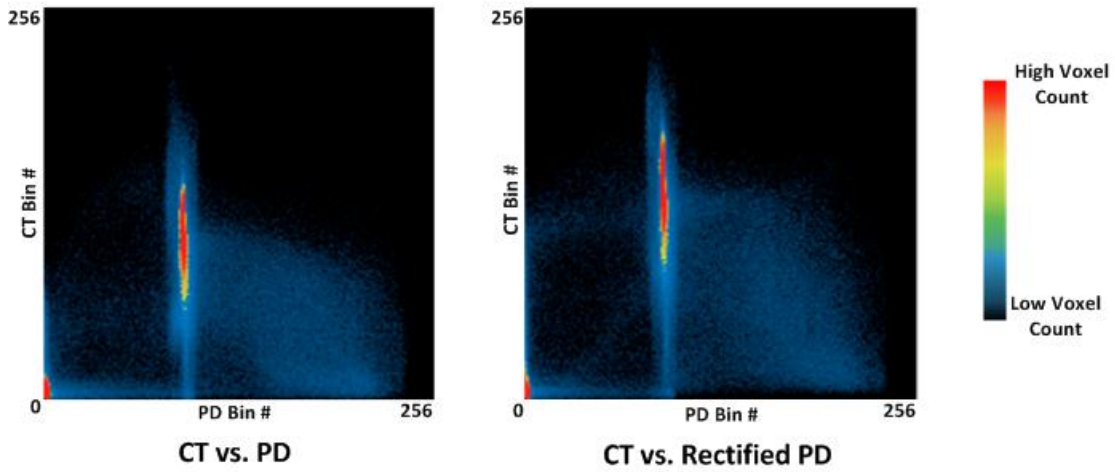

Figure 6.4 **- First computed histogram of CT vs. PD images for Patient 1**

6.3.2 PET to MR

Figure 6.5 shows the mean errors obtained for PET to MR registration for group A patient cases. PET images are not available for group B patients. The error bars represent the standard deviations of the alignment errors across the set of voxels used to calculate the means.

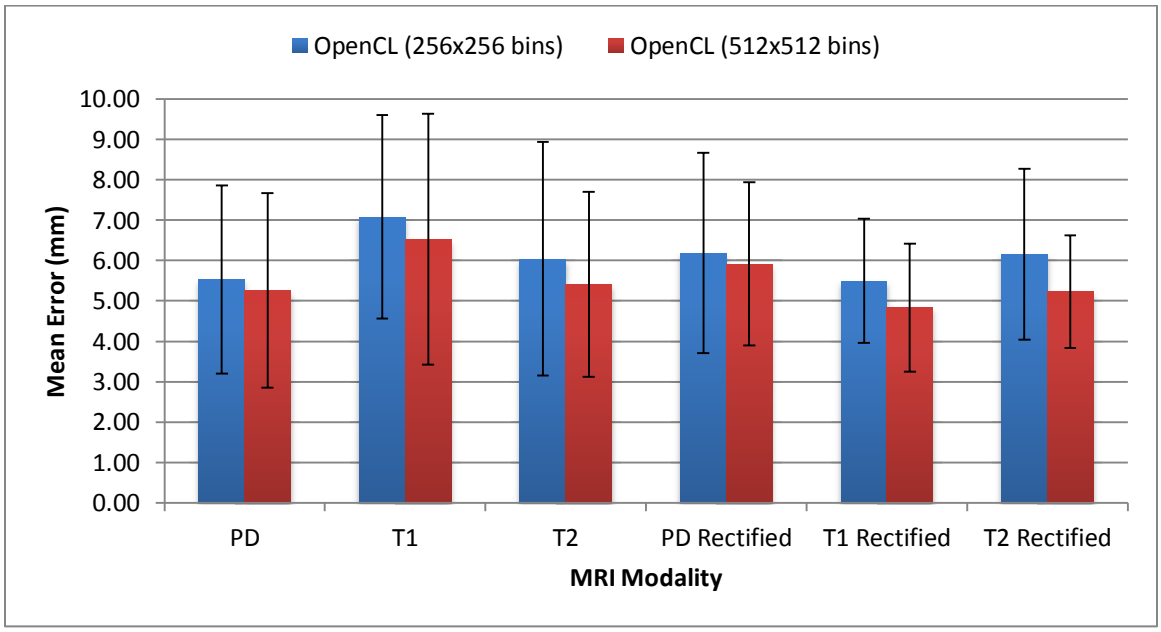

Figure 6.5 - RIRE test results for PET to MR registration

Using 256^2 bins, the overall registration error was 6.084 ± 0.578mm. With 512^2 bin, a mean registration error of 5.529 ± 0.600mm was obtained. The largest dimension for the PET volumes is 8mm for the slice thickness. For PET to MR registration, the OpenCL based registration method still performs to the desired accuracy of a mean error less than or equal to the largest voxel dimension. Similar to the results obtained for the CT cases, the standard deviations obtained for registration of rectified volumes was lower than the non-rectified volumes. No significant differences in deviations when using 256^2 versus 512^2 bins were observed. The exception was for PET to T2 rectified registration

where a smaller deviation was obtained since the results consisted of only 3 usable registration cases. However, as compared to CT registration, most of the alignment errors obtained are more than 4mm (half the slice thickness of the PET images). The decreased accuracy is mainly due to the quality of the PET images. A significant amount of noise can be observed outside the border of the skull region as shown in Figure 6.6. Since calculation of the similarity metric is not limited to using only the voxels of the region within the skull, the entropy values calculated will be affected by the quantity and intensity of the surrounding voxels. Therefore, maximization of MI can occur even if there is some misalignment of the reference MR volume and transformed PET volume.

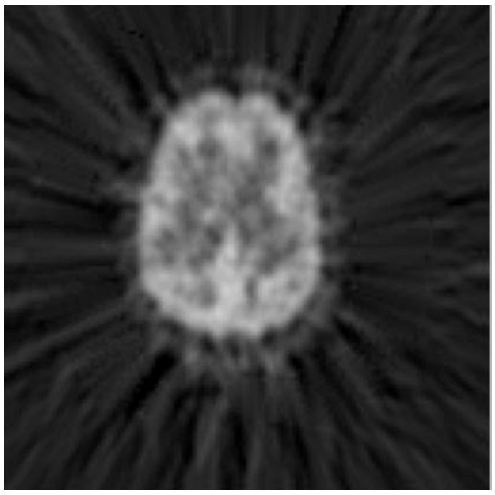

Figure 6.6 - PET image slice for Patient 1

Brightness and contrast adjusted to make border of skull more visible.

6.4 Timing Results

Table 6.4 and Table 6.5 show the average runtimes obtained for CT to MR and PET to MR registration for group A patients. The values provided represent the overall measured timings for all the MR image modalities available. Graphs showing the breakdown of GPU runtimes for individual MR modalities are included in the appendix section.

	Processing Mode			
	OpenCL GPU	**OpenCL GPU**	**OpenCL CPU**	**Regular CPU**
Number of Histogram Bins	256^2	512^2	256^2	256^2
Computation Time (seconds)	2.43 ± 0.80	6.77 ± 1.28	10.87 ± 3.51	149.92 ± 45.53
Overhead Time (seconds)	2.17 ± 0.16	2.20 ± 0.16	1.54 ± 0.12	32.47 ± 9.92
Total Runtime (seconds)	**4.61 ± 0.61**	**8.96 ± 0.88**	**12.41 ± 3.55**	**182.39 ± 55.12**

Table 6.4 - Runtimes of CT to MR Registration (Group A)

	Processing Mode			
	OpenCL GPU	**OpenCL GPU**	**OpenCL CPU**	**Regular CPU**
Number of Histogram Bins	256^2	512^2	256^2	256^2
Computation Time (seconds)	1.46 ± 0.45	6.42 ± 1.83	6.33 ± 2.27	87.14 ± 44.49
Overhead Time (seconds)	0.39 ± 0.08	0.38 ± 0.02	0.35 ± 0.01	18.55 ± 8.31
Total Runtime (seconds)	**1.85 ± 0.53**	**6.80 ± 1.85**	**6.68 ± 2.27**	**103.89 ± 43.22**

Table 6.5 - Runtimes of PET to MR Registration (Group A)

For CT to MR registration, the OpenCL GPU method using 256^2 bins has a 40-fold performance benefit against execution of the equivalent regular CPU method. A 2.5-

fold increase was observed when using the same OpenCL implementation on GPU against CPU. In the case of the OpenCL version running on CPU, the underlying drivers for OpenCL automatically make use of the multiple cores of the CPU. This results in the observed faster runtimes. The calculation of the mutual information metric accounted for 80% of the computation time for the OpenCL GPU method, 60% for the OpenCL CPU method and 9% for the regular CPU method. The difference in memory access patterns accounts for this variation. The OpenCL metric calculation timings are significantly higher than the regular CPU algorithm due to the use of the explicit atomic operations, which create stalls during execution to prevent partial memory. In contrast, the bottlenecks of the regular CPU algorithm are the computations involved in transformation and resampling of voxels. Regardless of those differences, the speedup obtained using OpenCL on GPU exceeded the initial expectation of ten-fold decrease in runtimes when compared to the single-threaded CPU algorithm.

For PET to MR registration, the OpenCL GPU method using 256^2 bins has a 56-fold performance increase over the regular CPU algorithm. Against the OpenCL implementation running on CPU, the GPU has 3.5-fold speedup. The calculation of the MI metric accounted for 60% of the computation time for the OpenCL GPU method, 50% for the OpenCL CPU method and 8.5% for the regular CPU method.

Using only the rectified volumes from group A for CT registration, L. Joskowicz et al. obtained runtimes of 87 ± 69secs using a dual-core CPU. This was achieved using a highly optimized registration algorithm as opposed to 512 ± 127secs for the non-optimized version [53]. The OpenCL implementation of NMI based registration running on the Radeon 5870 GPU has close to a 40-fold speedup against this optimized CPU

algorithm and a 200-fold speedup against the non-optimized version. The OpenCL implementation running on CPU achieves an 8-fold speedup against L.Joskowicz's optimized algorithm and a 42-fold speedup against the non-optimized version. Those results demonstrates the increased performance of using a parallel computing model such as OpenCL. Runtimes for N. Cahill's RIRE submission were not available. By comparing the recorded computation runtimes against those obtained by Shams et al.'s [18] CUDA-based GPU method, we can observe that the OpenCL implementation is approximately 2.5 times slower – Shams et al. obtained registration times of under one second for the same datasets. This difference is mainly attributed to the chosen optimizer and the algorithms used to calculate mutual information. Due to the use of NVIDIA hardware and CUDA, Shams was able to efficiently synchronize memory accesses when calculating MI. During the registration cycle, the optimizer may request application of a generated transformation and a re-evaluation of the similarity metric. A Powell optimizer was used for the CUDA implementation. A different set of conditions has to be met as compared to the Nelder-Mead simplex optimizer, thus resulting in different computation runtimes. Regardless of those differences, both the GPU OpenCL and CUDA implementations demonstrate the same magnitude of acceleration against CPU methods.

The runtimes for OpenCL registration demonstrate speedups obtained using a purely parallel processing environment. All the registration tasks were split across the compute units of the CPU or GPU. In the case of the CPU, the compute units are equivalent to the number of processor cores. Multi-threaded synchronization is not explicitly used on the CPU as a control mechanism for the execution of tasks for reduced time cost due to any possible data dependencies. Similarly, the regular CPU

implementation is single-threaded. A multi-threaded implementation on the regular CPU could potentially result in some speed improvements. Such an implementation is not expected to have gains that outweigh the benefit of the purely parallel processing platform available through OpenCL. The use of OpenCL is justified since the speedups were obtained without fine refinements to how tasks were being split and executed on the parallel architectures. Additional mechanisms could be introduced to take advantage of specific architecture differences of CPUs or GPUs to further enhance speeds. In comparison, within a multi-threaded environment, the fine control has to be introduced from the start. This increases the complexity of scheduling and distributing the tasks.

Overall runtimes obtained for CT to MR and PET to MR registration were under ten seconds. Thus, the speed for both CT and PET test cases are commensurate with the desired goals of using the OpenCL platform for accelerated medical image registration.

6.5 Group B Results Summary

Group B comprised of nine patient cases having different voxel dimensions and twice the number of images slices than group A. The test cases involved only CT to MR registration. The mean error obtained using 256x256 histogram bins was 2.195 ± 0.440mm. Using 512x512 bins, a mean error of 2.203 ± 0.448mm was obtained. The average runtimes obtained for those tests are shown in Table 6.6.

	Processing Mode			
	OpenCL GPU	**OpenCL GPU**	**OpenCL CPU**	**Regular CPU**
Number of Histogram Bins	256^2	512^2	256^2	256^2
Computation Time (seconds)	2.26 ± 0.47	8.15 ± 1.04	11.57 ± 2.91	212.19 ± 62.55
Overhead Time (seconds)	3.70 ± 0.31	4.04 ± 0.76	2.75 ± 0.20	47.59 ± 13.03
Total Runtime (seconds)	**5.96 ± 0.77**	**12.19 ± 1.80**	**14.43 ± 2.92**	**259.77 ± 75.49**

Table 6.6 - Runtimes of CT to MR Registration (Group B)

Calculation of the mutual information metric accounted for 75% of the computation time for the OpenCL GPU method, 45% for OpenCL CPU and 8.5% for regular CPU. Similar to group A, the accuracy for group B conforms to the hypothesis of achieving a mean error less than the largest voxel dimension. For this dataset, the largest dimension is still the slice thickness of 3mm. However, the runtimes obtained using 512^2 bins was slightly higher than the desired ten seconds. This is due to the CT and MR volumes of group B cases containing twice as many images slices than group A patients.

6.6 Ideal Calculation of Similarity Metric

The registration speeds obtained using OpenCL on GPU were indicative of high performance against other methods. From the results obtained, it can be observed that the largest bottleneck of this registration cycle was calculation of mutual information. Those runtimes could possibly be lowered by a significant amount if calculation of MI on GPU could be done efficiently. An improved method would involve splitting histogram calculation into smaller operations in a way that partial results can be merged at the end

of the procedure. The smaller operations could be localized to specific regions of the volumes instead of looking at the entire volume.

As described by the formulae in Chapter 4, the ideal calculation of similarity metrics involves consideration of only the overlapping regions of the input and transformed volumes. In a real situation, this is a difficult condition to meet due to issues such as different image sizes, different patient skull size or modality of images. The segmentation of the local portion of relevant tissue to be used is a different computational problem from registration and was out of the scope of this project. Segmentation, akin to registration, is also a computationally intensive task that can take minutes or hours to perform. However, current research in this field has shown the possibility of using GPUs to reduce the runtime of this procedure in a similar fashion to the OpenCL based registration method. The combination of both accelerated methods could lead to highly accurate analysis of medical images.

Chapter Seven: **Future Work and Conclusion**

7.1 Future Work

OpenCL is a platform that is supported by different types of processing hardware and operating systems. However, the experiments conducted within this project were limited to only a few representative processors (AMD Radeon HD 5870, AMD Radeon HD6750m and Intel Core i7). Future work would involve testing the correctness of our image registration technique using a larger set of OpenCL compliant computer architectures and operating systems. This would include new AMD Radeon GPUs, NVIDIA GeForce GPUs and desktop versions of Intel CPUs running on Microsoft Windows or Linux systems. One part of implementing OpenCL based registration that was overlooked is optimization of the algorithms used. The same code used could be modified to enhance memory access, thus leading to additional performance gains. Since the processing speeds obtained on GPUs may not be scalable to other architectures, optimization techniques that could reduce the computation runtime, regardless of the chosen hardware, would be investigated. In November 2011, version 1.2 of the OpenCL API was ratified. This standard includes the ability to subdivide and partition a compute device into smaller devices allowing refined control over how calculations are performed. Additionally, this provides functionality allowing task parallelism to be more easily implemented. Such changes could allow for efficient calculation of mutual information using GPUs.

7.2 Conclusion

7.2.1 Biomedical Contributions

Research previously conducted has demonstrated the possibility of performing image registration on graphics processors at a very fast speed. The presented method of accelerated image registration using OpenCL demonstrates the possibility of using this computing platform to process huge volumes of medical data simultaneously and accurately. This can be done while maintaining a low cost since regular consumer-level graphics processors can be used. By making the registration modules platform independent, clinical deployment is made easy since re-adaptation of the methods to account for differences in hardware or software would not be necessary.

7.2.2 General Engineering and Scientific Contributions

New computer technologies are introduced very frequently. However, the adoption rate of those technologies tends to be low, especially in research and development fields. This is due to factors such as difficulty to integrate new methods into existing software or very few tested and proven use cases. OpenCL is a platform that has the potential of enabling heavily computational tasks to be performed on various types of hardware. Medical image registration is a novel way of demonstrating the capability of the API and generating more interest into the adoption of a technology that can significantly improve the pace of scientific research. Additionally, OpenCL is a free platform that relies on the contribution of programmers around the world to evolve. The use of OpenCL for scientific computing can lead to the identification of areas of improvement for the

platform which can in turn be addressed in future revisions of the API for the benefit of potential commercial applications.

7.2.3 Final Words

A portable rigid image registration method using the OpenCL framework was created. The speed and accuracy of this registration technique were measured. Rapid computation times, averaging at 4.6 secs, were achieved. This represents a 40-fold speedup over conventional CPU based algorithms. The accuracy results obtained show that the OpenCL based medical image registration algorithms produce output volumes that are close to the gold standard for the RIRE test cases. The mean errors obtained corresponded to deviations less than or equal to one voxel from what would have been the perfect location after re-alignment. Those results showed that the OpenCL image registration method is highly accurate and commensurate with other published equivalent methods.

APPENDIX A: EXTENDED RESULTS

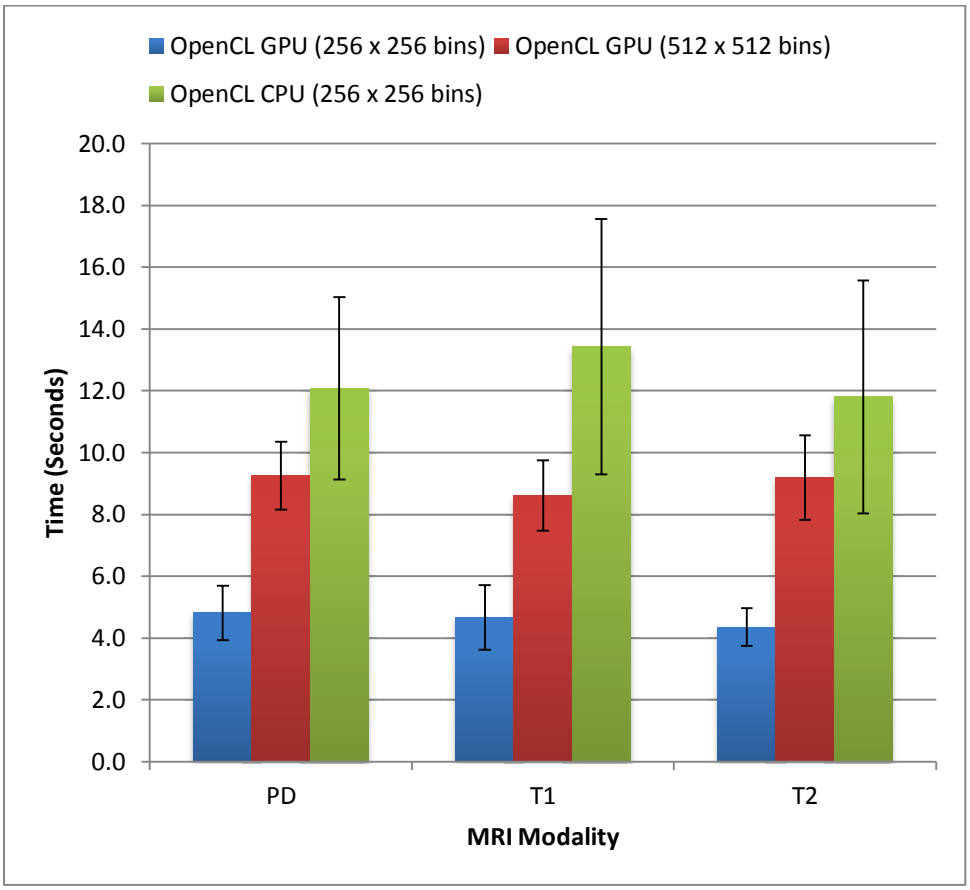

Figure A.1 - CT to MR registration runtimes for Group A

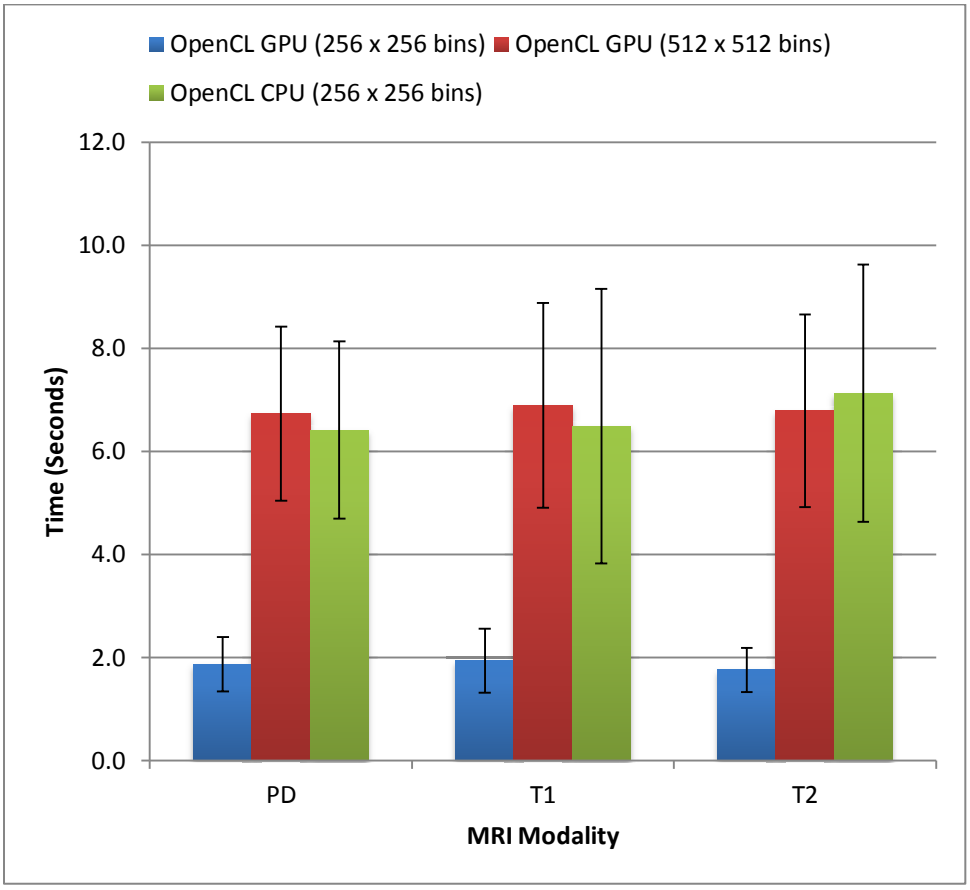

Figure A.2 - PET to MR registration runtimes for Group A

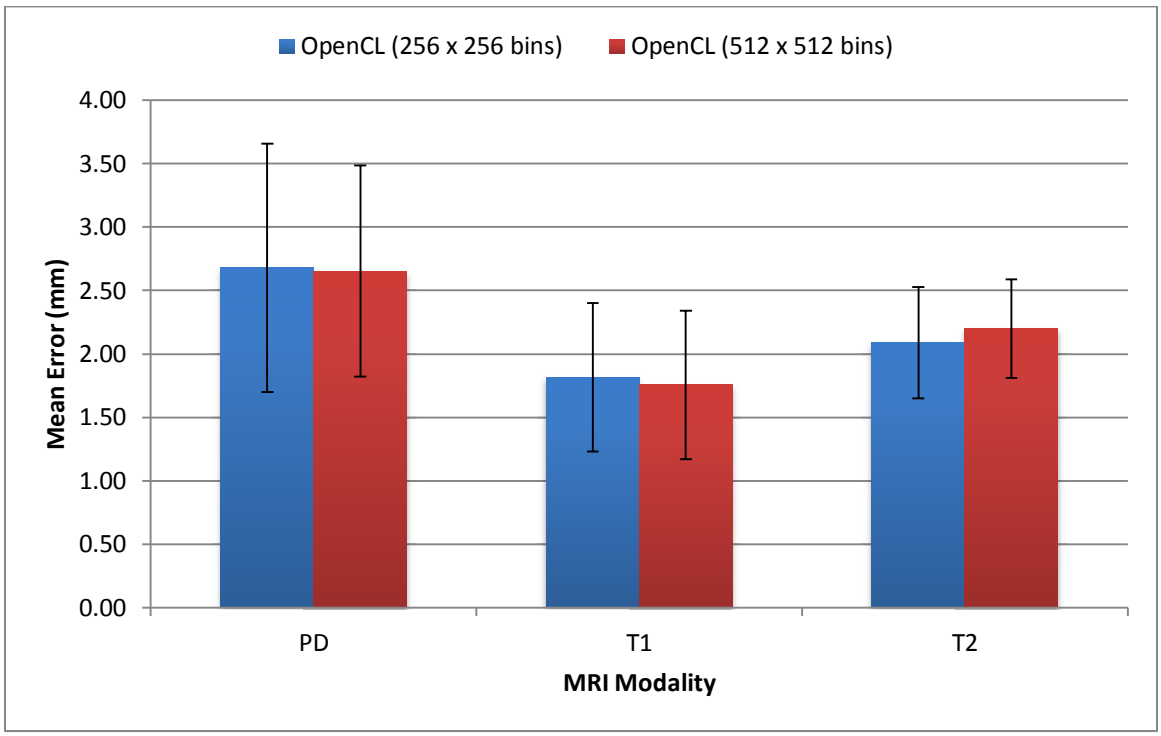

Figure A.3 - CT to MR registration for RIRE Group B cases

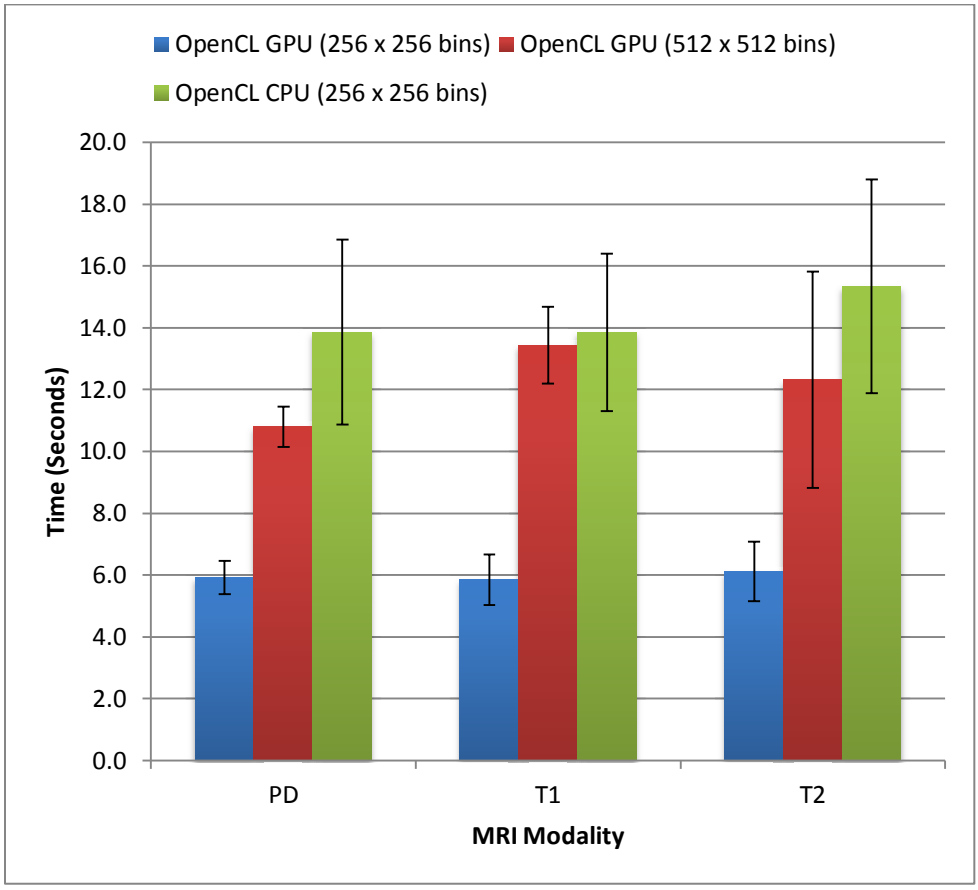

Figure A.4 - CT to MR registration runtimes for Group B

References

[1] S. Klein, M. Staring, K. Murphy, M. A. Viergever, and J. Pluim, "Elastix: a toolbox for intensity-based medical image registration," *Medical Imaging, IEEE Transactions on*, vol. 29, no. 1, pp. 196–205, 2010.

[2] "ITK - Insight Toolkit." [Online]. Available: http://www.itk.org/. [Accessed: 23-Nov-2011].

[3] D. Lee, M. Hofmann, F. Steinke, Y. Altun, N. D. Cahill, and B. Scholkopf, "Learning similarity measure for multi-modal 3D image registration," in *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009*, 2009, pp. 186–193.

[4] P. A. van den Elsen, E.-J. . Pol, and M. A. Viergever, "Medical image matching-a review with classification," *IEEE Engineering in Medicine and Biology Magazine*, vol. 12, no. 1, pp. 26–39, Mar. 1993.

[5] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," *Medical Image Analysis*, vol. 2, no. 1, pp. 1–36, Mar. 1998.

[6] J. M. Fitzpatrick, "Description of the original RIRE project." [Online]. Available: http://www.insight-journal.org/rire/download/description.pdf. [Accessed: 08-Nov-2011].

[7] Daniel Adler, "Accelerated Medical Image Registration using the Graphics Processing Unit," MSc, University of Calgary, Calgary, Alberta, Canada, 2011.

[8] J. West, J. M. Fitzpatrick, M. Y. Wang, B. M. Dawant, C. R. Maurer Jr, R. M. Kessler, R. J. Maciunas, C. Barillot, D. Lemoine, A. Collignon, and others, "Comparison and evaluation of retrospective intermodality brain image registration techniques," *Journal of computer assisted tomography*, vol. 21, no. 4, p. 554, 1997.

[9] N. Bertrouni, "Le recalage en imagerie médicale : de la conception à la validation," *IRBM*, vol. 30, no. 2, pp. 60–71, Apr. 2009.

[10] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kr\üger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," in *Computer graphics forum*, 2007, vol. 26, pp. 80–113.

[11] "NVIDIA CUDA C Programming Guide Version 3.1." NVIDIA Corporation, 28-May-2010.

[12] "Compute Unified Device Architecture (CUDA)." [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed: 18-Nov-2011].

[13] "OpenCL - The open standard for parallel programming of heterogeneous systems." [Online]. Available: http://www.khronos.org/opencl/. [Accessed: 18-Nov-2011].

[14] S. P. Dimaio, N. Archip, N. Hata, I.-F. Talos, S. K. Warfield, A. Majumdar, N. McDannold, K. Hynynen, P. R. Morrison, W. M. I. I. I. Wells, D. F. Kacher, R. E. Ellis, A. J. Golby, P. M. Black, F. A. Jolesz, and R. Kikinis, "Image-guided neurosurgery at Brigham and Women's Hospital," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 25, no. 5, pp. 67 –73, Oct. 2006.

[15] D. Rueckert and P. Aljabar, "Nonrigid Registration of Medical Images: Theory, Methods, and Applications [Applications Corner]," *Signal Processing Magazine, IEEE*, vol. 27, no. 4, pp. 113 –119, Jul. 2010.

[16] D. L. . Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in medicine and biology*, vol. 46, p. R1, 2001.

[17] P. Viola and W. M. Wells III, "Alignment by maximization of mutual information," *International journal of computer vision*, vol. 24, no. 2, pp. 137–154, 1997.

[18] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," *Computer methods and programs in biomedicine*, vol. 99, no. 2, pp. 133–146, 2010.

[19] C. Ewertsen, "Image fusion between ultrasonography and CT, MRI or PET/CT for image guidance and intervention-a theoretical and clinical study," *Danish medical bulletin*, vol. 57, no. 9, pp. 1–16, 2010.

[20] Sonny Chan, "Three-Dimensional Medical Image Registration on Modern Graphics Processors," MSc, University of Calgary, Calgary, Alberta, Canada, 2007.

[21] J.-P. Thirion, "Image matching as a diffusion process: an analogy with Maxwell's demons," *Medical Image Analysis*, vol. 2, no. 3, pp. 243–260, Sep. 1998.

[22] "Retrospective Image Registration Evaluation." [Online]. Available: http://www.insight-journal.org/rire/. [Accessed: 08-Nov-2011].

[23] W. R. Hendee and E. R. Ritenour, *Medical Imaging Physics*, 4th ed. Wiley-Liss, 2002.

[24] R. C. Smith and R. C. Lange, *Understanding Magnetic Resonance Imaging*. CRC Press, 1997.

[25] R. Damadian, "Tumor detection by nuclear magnetic resonance," *Science*, vol. 171, no. 3976, p. 1151, 1971.

[26] R. Damadian, K. Zaner, D. Hor, T. DiMaio, L. Minkoff, and M. Goldsmith, "Nuclear magnetic resonance as a new tool in cancer research: human tumors by NMR," *Annals of the New York Academy of Sciences*, vol. 222, no. 1, pp. 1048–1076, 1973.

[27] P. C. Lauterbur and others, "Image formation by induced local interactions: examples employing nuclear magnetic resonance," *Nature*, vol. 242, no. 5394, pp. 190–191, 1973.

[28] J. Mallard, J. M. S. Hutchison, W. Edelstein, R. Ling, and M. Foster, "Imaging by nuclear magnetic resonance and its bio-medical implications," *Journal of Biomedical Engineering*, vol. 1, no. 3, pp. 153–160, Jul. 1979.

[29] N. R. Mollet, F. Cademartiri, and P. J. de Feyter, "Non-invasive multislice CT coronary imaging," *Heart*, vol. 91, no. 3, pp. 401 –407, Mar. 2005.

[30] M. Goyal, "Understanding Alberta Stroke Program Early CT Score (ASPECTS)." [Online]. Available: http://www.aspectsinstroke.com/imaging-in-acute-stroke/role-of-ncct/. [Accessed: 25-Jan-2012].

[31] R. Brinks and T. M. Buzug, "Image reconstruction in positron emission tomography (PET): the 90th anniversary of Radon's solution/Bildrekonstruktion in der Positronen-Emissions-Tomographie (PET): zum 90. Jahrestag von Radons Lösung," *Biomedizinische Technik*, vol. 52, no. 6, pp. 361–364, 2007.

[32] "Positron Emission Tomography (PET)." [Online]. Available: http://www.webmd.com/a-to-z-guides/positron-emission-tomography. [Accessed: 26-Jan-2012].

[33] “Texas Instruments,” *Texas Instruments*. [Online]. Available: http://www.ti.com/.

[34] “TMS34010 Graphics systems processor,” *Texas Instruments*. [Online]. Available: http://www.ti.com/product/tms34010.

[35] K. M. Guttag, T. M. Albers, M. D. Asal, and K. G. Rose, “The TMS34010: an embedded microprocessor,” *IEEE Micro*, vol. 8, no. 3, pp. 39–52, Jun. 1988.

[36] J. P. Farrugia, P. Horain, E. Guehenneux, and Y. Alusse, “GPUCV: A framework for image processing acceleration with graphics processors,” in *Multimedia and Expo, 2006 IEEE International Conference on*, 2006, pp. 585–588.

[37] “OpenGL - The Industry’s Foundation for High Performance Graphics.” [Online]. Available: http://www.khronos.org/opengl. [Accessed: 23-Nov-2011].

[38] “DirectX Developer Center.” [Online]. Available: http://msdn.microsoft.com/en-us/directx/default. [Accessed: 23-Nov-2011].

[39] “OpenGL Shading Language.” [Online]. Available: http://www.opengl.org/documentation/glsl/. [Accessed: 27-Dec-2011].

[40] “Cg Toolkit.” [Online]. Available: http://developer.nvidia.com/cg-toolkit. [Accessed: 27-Dec-2011].

[41] “High Level Shading Language (HLSL).” [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/bb509561%28v=VS.85%29.aspx. [Accessed: 27-Dec-2011].

[42] “BrookGPU.” [Online]. Available: http://graphics.stanford.edu/projects/brookgpu/. [Accessed: 27-Dec-2011].

[43] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan, “GpuCV: A GPU-accelerated framework for image processing and computer vision,” *Advances in Visual Computing*, pp. 430–439, 2008.

[44] “Acceleware - Processing Superpower.” [Online]. Available: http://www.acceleware.com/products. [Accessed: 23-Nov-2011].

[45] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, “A performance study of general-purpose applications on graphics processors using

CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, Oct. 2008.

[46] Waage, "Accelerated Filtering using OpenCL," 2009.

[47] M. J. Flynn, "Some computer organizations and their effectiveness," *Computers, IEEE Transactions on*, vol. 100, no. 9, pp. 948–960, 1972.

[48] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[49] R. Cypher and J. L. . Sanz, "SIMD architectures and algorithms for image processing and computer vision," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 12, pp. 2158–2174, Dec. 1989.

[50] "Big Oh in the parallel world." [Online]. Available: http://igoro.com/archive/big-oh-in-the-parallel-world/. [Accessed: 30-Dec-2011].

[51] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," in *ACM Transactions on Graphics (TOG)*, 2004, vol. 23, pp. 777–786.

[52] J. Nickolls and W. J. Dally, "The GPU Computing Era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Apr. 2010.

[53] M. Freiman, M. Werman, and L. Joskowicz, "A curvelet-based patient-specific prior for accurate multi-modal brain image rigid registration," *Medical Image Analysis*, vol. 15, no. 1, pp. 125–132, Feb. 2011.

[54] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in Medicine and Biology*, vol. 46, p. R1–R45, Mar. 2001.

[55] C. E. Shannon and W. Weaver, *The mathematical theory of communication*, vol. 19. University of Illinois Press Urbana, 1962.

[56] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, "Automated multi-modality image registration based on information theory," in *Information processing in medical imaging*, 1995, vol. 3, pp. 264–274.

[57] Studholme, D. Hill, and D. J. Hawkes, "An overlap invariant entropy measure of 3D medical image alignment .," *Pattern Recognition*, vol. 32, no. 1, pp. 71–86, 1999.

[58] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and Flannery, Brian P., *Numerical recipes in C*. Cambrigde: Cambridge University Press, 1992.

[59] R. Shams and N. Barnes, "Speeding up mutual information computation using NVIDIA CUDA hardware," in *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*, 2007, pp. 555–560.

[60] F. Jung and S. Wesarg, "3D Registration Based on Normalized Mutual Information: Performance of CPU vs. GPU Implementation," *Bildverarbeitung für die Medizin], Deserno, T. and Hoffmann, J., eds., Informatik aktuell*, vol. 5, 2010.

[61] M. Ohara, H. Yeo, F. Savino, G. Iyengar, L. Gong, H. Inoue, H. Komatsu, V. Sheinin, and S. Daijavad, "Accelerating mutual-information-based linear registration on the cell broadband engine processor," in *Multimedia and Expo, 2007 IEEE International Conference on*, 2007, pp. 272–275.

[62] "Brainweb: Simulated Brain Database." [Online]. Available: http://mouldy.bic.mni.mcgill.ca/brainweb/. [Accessed: 09-Apr-2012].

[63] H. Chang and J. M. Fitzpatrick, "A technique for accurate magnetic resonance imaging in the presence of field inhomogeneities," *IEEE Transactions on Medical Imaging*, vol. 11, no. 3, pp. 319–329, Sep. 1992.

[64] M. N. Safrana, M. Freimanb, M. Wermanb, and L. Joskowiczb, "Curvelet-based sampling for accurate and efficient multi-modal image registration," in *Proc. of SPIE Vol*, 2009, vol. 7259, p. 72590M–1.

[65] N. D. Cahill, "Normalized Measures of Mutual Information with General Definitions of Entropy for Multimodal Image Registration," in *Biomedical Image Registration*, vol. 6204, Springer Berlin Heidelberg, 2010, pp. 258–268.

[66] "Matlab Optimization Toolbox." [Online]. Available: http://www.mathworks.com/products/optimization/. [Accessed: 09-Apr-2012].

[67] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1 –58, 2008.

[68] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, p. 308, 1965.

[69] R. Shekhar and V. Zagrodsky, "Mutual information-based rigid and nonrigid registration of ultrasound volumes," *Medical Imaging, IEEE Transactions on*, vol. 21, no. 1, pp. 9–22, 2002.