PERFORMANCE-AWARE PROGRAMMING FOR INTRAOPERATIVE INTENSITY-BASED IMAGE REGISTRATION ON GRAPHICS PROCESSING UNITS

by

Leong Chun Wing Martin

A thesis submitted in partial fulfilment of the requirements of
the Degree of Master of Philosophy
at The University of Hong Kong

September 2018

Abstract of thesis entitled

"Performance-Aware Programming for Intraoperative Intensity-based Image Registration on Graphics Processing Units"

Submitted by

Leong Chun Wing Martin

for the degree of Master of Philosophy

at The University of Hong Kong

in September 2018

Recent advancement of intraoperative imaging technologies allows real-time view of the tissue morphologies to ensure safer and efficient interventions. Particularly, intraoperative imaging is extensively used in surgical scenarios that require accurate target localization and precise movement of surgical tools. Stereotactic neurosurgery and cardiac catheterization are typical examples. However, the intraoperatively acquired image may not be aligned with the preoperative image used for planning, due to motion, gravity, or interventions. Intensity-based non-rigid image registration is able to resolve such misalignment, but it suffers from prolonged registration time due to its high computation requirement. This extended registration time makes intensity-based registration inadmissible to the highly dynamic surgical scenarios.

To allow seamless application of intra-operative application without disrupting the surgical workflow, there is a constant demand for having a fast intensity-based registration. Graphics processing units (GPU) have attracted the most attention in the recent years due to its unmatched parallel computing power. However, many works of GPU-based image registration have overlooked the underlying memory transaction patterns, which can hamper computation efficacy if not appropriately managed. In view of achieving fast computation, *performance-aware programming* is a specialized practice that involves repeated profiling, micro-benchmarking, and code optimization to ensure full device utilization.

In this thesis, *performance-aware programming* techniques were employed on GPU to resolve for the high computation requirement in the *diffeomorphic log-demons* algorithm, which is one of the most popular intensity-based image registration algorithms. The GPU implementation of the algorithm was tested and analyzed extensively. By successfully pinpointing and optimizing for the blocking operations, significant (>200×) performance speed-up has been achieved as a promising result.

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

Signed _____ / Leong Chun Wing Martin

“A dream worth three years of dedication”

三年・一夢

ACKNOWLEDGEMENTS

"A dream worth three years of dedication" 三年・一夢

"A dream worth three years of dedication" was a saying since my undergraduate study, describing how university life goes by. I am privileged to be allowed spending six years at my alma mater, the University of Hong Kong, to dream twice. Approaching the conclusion of this wonderful M.Phil. journey, I would like to take this opportunity to express my gratitude to those who accompanied and supported me through these years.

This work could not have made possible without my supervisors, Dr. Ka-wai Kwok and Prof. James Lam's support in every aspect. They are the ones who supported me throughout the journey whilst allowing me the room to work in my own way. Their sense and enthusiasm in research are always motivating. Thank you for bringing me into this wonderful research journey. My research topic, *performance-aware programming*, not only makes me aware of computation performance, but also taught me to *self-aware* of my downsides.

Thank you to all current and past colleagues in the Group of Interventional Robotics and Imaging System for their help and support during both the good days and the bad days. Especially Brian, Marco, Jacky, Ziyan, Ziyang, Xiaomei, Justin, Fai, Peggy, and Alan. Thank you to Bowen Kwan, An Qu, and Gary Chow who worked with me together on this topic. The road of research is harsh, but it is a bit less bitter to have you here to walk together.

Finally, thank you to my good old friends from high school, Chung, Jeffrey, Fan, Jeni, Rachel, Loraine and Holly; as well as my classmates from my undergraduate study, Mannix, Rachel and Keo for their support at all times.

Last but not least, I am greatly blessed to have my parents' and my girlfriend, Dorothy's, unlimited support and consideration throughout this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ACRONYMS

AoS	Array-of-Structure
API	Application Programming Interface
CBCT	Cone-Beam CT
CP	Control Point
CPU	Central Processing Unit
CT	Computed Tomography
CUDA	Compute Unified Device Architecture
DARTEL	Diffeomorphic Anatomical Registration Using Exponentiated Lie Algebra
DoF	Degree-of-Freedom
EA	Electro-Anatomical
EP	Electrophysiotherapy
FFD	Free Form Deformation
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
iMRI	Intraoperative MRI
IGRT	Image-guided Radiotherapy
Intra-op	Intraoperative
IMRT	Intensity-modulated Radiotherapy
MRI	Magnetic Resonance Imaging
MSE	Mean Squared Error
nvvp	NVidia Visual Profiler
OP/s	Operation per Second
Pre-op	Preoperative
PSO	Particle Swarm Optimization

PTX	Parallel Thread Execution
SDK	Software Development Kit
SIMT	Single Instruction, Multiple Threads
SoA	Structure-of-Array
SVF	Stationary Vector Field
TPS	Thin Plate Spline

Chapter 1

INTRODUCTION

1.1 Motivation

Intraoperative (intra-op) imaging allows perioperative visualization of any tissue morphological changes. By providing timely visual guidance to the surgical instruments, intra-op imaging enables safer and efficient interventions. Particularly, intra-op imaging is very useful in surgical scenarios that require precise manipulation and control of surgical tools, such as stereotactic neurosurgeries and cardiac catheterization. However, anatomical disparity can present between the preoperative (pre-op) image and intra-op image, due to physiological motion, gravity, or tool-tissue interaction. Such disparity will have to be resolved using non-rigid image registration. Once the misalignment is resolved, accurate surgical guidance can be achieved by virtually augmenting the predefined treatment target/ critical regions onto the intra-op images.

There are two main streams of non-rigid registration approaches: feature-based and intensity-based registration. Feature-based registration relies on automatic feature detection to re-align the image. Intensity-based registration accesses the pixel/voxel intensity value for image co-registration. Considering image noise and artifacts can be prevalent in the intra-op images, intensity-based methods is a more preferable approach due to its higher tolerance to noise and artifacts. However, most intensity-based registration requires substantial computation to register the images, which can preclude smooth surgical workflow in image-guided surgeries.

The current trend of non-rigid image registration resides in the realization of these co-registration algorithms in the surgical scenario. In particular, despite a number of intensity-based image registration algorithms were proposed, these algorithms are generally set back by their high computation requirement. To this end, there is a need to accelerate the computation process of such registration. Computation enhancement can be achieved by utilizing application accelerators. Particularly, the graphical processing unit (GPU) is one of the most commonly used application accelerators with massive parallel computation power. However, many works on GPU image registration overlooked the importance of low-level optimizations in their implementations, which can lead to under-utilization of the device. In this regard, performance-aware programming is the key to ensure full device utilization. With the implementation being optimized, GPU can accelerate the image registration process by at least an order of magnitude for seamless integration with existing surgical navigation devices.

The purpose of this thesis is to address the technological gap in the actual realization of recent non-rigid image registration algorithms in clinical practice. Performance-aware programming techniques will be implemented on GPU to accelerate a popular intensity-based image registration algorithm, the *diffeomorphic log-demons*. The main objectives of the thesis include:

(1) To provide a comprehensive performance analysis of *diffeomorphic log-demons*, thus pinpointing the key computation process that bottlenecks the computation process;
(2) To devise proper optimization schemes to accelerate any bottlenecking computations pinpointed by **(1)** using a GPU; and
(3) To realize high-performance *diffeomorphic log-demons* on a GPU to enable rapid co-registration between the pre-op and the intra-op images.

1.2 Thesis organization and main technical contributions

In **Chapter 2**, an overview of current intra-op imaging technique, as well as the clinical demands of such image-guidance will be highlighted. Basic principles of non-rigid image registration, particularly intensity-based image registration, will also be included. The chapter concludes by introducing the programming model and hardware architecture of modern GPU, and a horizon scan of related work of using GPU to accelerate the computation of intensity-based image registration.

In **Chapter 3**, extended testing and profiling on the *diffeomorphic log-demons*, one of the most popular intensity-based image registration algorithm, will be conducted. Different memory access pattern on the GPU will be investigated. Moreover, time-critical computation steps of the *diffeomorphic log-demons* algorithm will also be looked into. Focusing on these time-critical operations, various *performance-aware programming* techniques are proposed to resolve the computation bottlenecks.

In **Chapter 4**, the implementation of an optimized GPU version of *diffeomorphic log-demons* using performance-aware programming techniques will be presented. Focusing on the previously identified computation bottlenecks, extensive experiments will be conducted to quantitatively investigate the computation enhancement brought by performance-aware programming. In-depth analysis of such optimized computation will also be performed. Finally, an optimal implementation of *diffeomorphic log-demons* will be presented.

With the working implementation of *diffeomorphic log-demons* on GPU, **Chapter 5** discusses the technical considerations for this optimized GPU image registration scheme to be employed for extensive applications. With the ability to swiftly perform the registration, potential impact of this work will also be discussed. This implementation of this high-performance *diffeomorphic log-demons* on GPU will also be open sourced in the near future.

1.3 Publications and exposure

The author of this thesis has participated and in the authorship of various peer-reviewed international journals and conferences proceedings within the period of this study:

Related publications:

1. M.C.W. Leong, P.Y. Kwan, K.H. Lee, G.C.T. Chow, W. Luk and K.W. Kwok,
   "Performance-aware Programming in Intensity-based Non-rigid Image Registration: Implementation with GPU and FPGA", **Medical Image Analysis**. *(in preparation)*

2. Z. Guo, M.C.W. Leong, H. Su, K.W. Kwok, D.T.M. Chan and W.S. Poon, *"Techniques for Stereotactic Neurosurgery: Beyond the 'Frame', Towards the Intraoperative MRI-guided and Robot-assisted Approaches",* **World Neurosurgery**, 2018. *(in press)*

3. K.H. Lee, D.K.C. Fu, Z. Guo, Z. Dong, M.C.W. Leong, C.L. Cheung, A.P.W. Lee, W. Luk and K.W. Kwok, *"MR Safe Robotic Manipulator for MRI-Guided Intracardiac Catheterization ",* **IEEE/ASME Transactions on Mechatronics**, vol.23, no.2, pp.586-595, 2018.

4. Z. Dong, Z. Guo, D.K.C. Fu, K.H. Lee, M.C.W. Leong, C.L. Cheung, A.P.W. Lee, W. Luk and K.W. Kwok, *"A Robotic Catheter System for MRI-guided Cardiac Electrophysiological Intervention",* **Workshop in IEEE International Conference on Robotics and Automation (ICRA)**, 2017.

Other publications published in the study period:

1. S.Fan, A.Chan, S.Au, M.C.W. Leong, M. Chow, Y.T. Fan, R. Wong, S. Chan, S.K. Ng, A.P.W. Lee and K.W. Kwok, *"Personalized anaesthesia: three-dimensional printing of facial prosthetic for facial deformity with difficult airway",* **British Journal of Anaesthesia**, 2018. *(in press)*

2. M.C.W Leong, K.W. Kwok, Y.T. Fan and A.P.W Lee, *" Using 3D Printed Models For Planning Transcatheter Aortic Valve Implantation in Patients With Bicuspid Aortic Valve",* **Journal of the American College of Cardiology**, vol. 71, no. 11, p. A1130, 2018.

3. M.C.W Leong, K.W. Kwok and A.P.W Lee, *"Prediction of Paravalvular Leak after Transcatheter Aortic Valve Implantation using Patient-specific 3D-printed Models - a case with Bicuspid Aortic Valve Stenosis",* **Proceedings of 43rd Annual Scientific Meeting of Korean Society of Echocardiography**, 2017.

4. K.H. Lee, D.K.C. Fu, M.C.W. Leong, M. Chow, H.C. Fu, K. Althoefer, K.Y. Sze, C.K. Yeung and K.W. Kwok, *"Nonparametric Local Learning for Hyper-Elastic Robotic Control: An Enabling Technique for Effective Endoscopic Navigation",* **Soft Robotics**, vol. 4, no. 4, pp. 324-337, 2017.

5. K.H. Lee, M.C.W. Leong, M. Chow, H.C. Fu, W. Luk, K.Y. Sze, C.K. Yeung, K.W. Kwok, *"FEM-based Soft Robotic Control Framework for Intracavitary Navigation",* **Proceedings of IEEE International Conference on Real-time Computing and Robotics (RCAR)**, Okinawa, 2017, pp.11-16.

6. C.L. Cheung, K.H. Lee, Z. Guo, Z. Dong, M.C.W. Leong, Y. Chen, A.P.W. Lee and K.W. Kwok, *"Kinematic-Model-Free Positional Control for Robot-Assisted Cardiac Catheterization",* **Proceedings of 9th Hamlyn Symposium on Medical Robotics**, 2016.

Chapter 2

INTRAOPERATIVE NON-RIGID IMAGE REGISTRATION

2.1 Introduction

Image-guided intervention has caught constant research attention since 1986 when the first image-guided surgery is performed [1]. It satisfies the much-needed accuracy requirement by providing essential visual guidance to the surgeons. Particularly, having thorough knowledge on the locations of the critical/target tissue through image guidance is imperative to perform precise instrument manipulation. Damage to the surrounding healthy tissues, and subsequently invasiveness dealt on the patient can, therefore, be reduced. The advancement of image-guided surgeries also enables minimally invasive interventions by visualizing the patient's anatomy in real-time.

In this chapter, a brief survey on intra-op imaging techniques will be conducted, followed by the clinical demands of intra-op imaging in some surgical applications. Potential problems caused by misalignment between the pre-op image and the intra-op image will also be discussed. Non-rigid registration schemes are the clue to resolving the image misalignment, but they will have to be robust and fast enough to cope with the dynamic intra-op scenario. A balance between accuracy, reliability and computation speed will have to be made. GPU is one of the promising application accelerators which can be used to accelerate the registration. To pinpoint the current research demand to enable intra-op registration,

a survey of image registration techniques and related work to accelerate the registration process is also presented in the latter part of this chapter.

2.2 Intraoperative imaging

Image-guided intervention is a general term describing any surgical or interventional procedures where the procedure is done in conjunction with intra-op imaging to guide the interventional procedure. It is often used to perform operations that associate with high risk, for example, when the treatment target (e.g. tumor) is very close to critical regions or organs (e.g. brain stem, major arteries, etc.). The surgical instrument used in such intervention is often tracked. A wide variety of intra-op imaging modalities are developed for accurate tracking of the surgical apparatus and tissue margins, such as ultrasound-based [2], optical-based [3-5], X-ray-based [6], and MR-based [7]. Particularly, the latter two imaging modalities are able to provide timely intra-op images with a fixed frame of reference, which can be useful to realign the image with a pre-op roadmap for accurate guidance [8, 9].

In this section, the general surgical workflow of image-guided intervention will be introduced, which depicts the current demand for frequent intra-op scans throughout the procedure. The basic principle, application, advantages, and disadvantages of some mainstream intra-op imaging modalities are also discussed.

2.2.1 Surgical workflow in image-guided intervention

Prior to any surgery, surgical planning is essential for the surgeons to have an overview. Pre-op scanning in different imaging modalities allows the construction of a knowledge-based model that allows surgeons to plan the operation procedures. For example, in image-guided radiation therapy, the planned target volume and other critical regions/organs are segmented during the treatment planning process [10]. This knowledge-based model provides invaluable guidance for the surgeons to select appropriate strategies to achieve the expected clinical outcomes. However, such pre-op model will not account for any possible anatomical/pathological changes that occur after the pre-op scan. Any misalignment between the pre-op models and the actual surgical scenario introduces uncertainties to the surgeons.

Intra-op imaging allows perioperative localization of the treatment target as well as the surgical instrument. Prior to any critical procedures (e.g. insertion of the biopsy needle, surgical resection, etc.), an intra-op image is usually acquired for the surgeons to make any necessary adjustments to the instruments. This image acquisition – instrument reposition process may iterate for several times in order to get the surgical tool to align with the treatment target [11]. Furthermore, treatment effectiveness of surgical procedure can be instantly validated using intra-op scans after any critical procedures (e.g. placement of electrode, removal of tumor, etc.) are performed. Malposition of instruments or implants can also be promptly visualized peri-operatively to make any necessary adjustments.

In all, intra-op guidance mitigates the risks of unintentional damage to surrounding healthy tissue near the treatment area. With accurate target localization and precise instrument manipulation, image-guided intervention can undoubtedly improve surgical safety and accuracy.

2.2.2 Medical imaging modalities for image-guided intervention

2.2.2.1 Intra-op computed tomography

Intra-op computed tomography (CT), alongside with other X-ray-based scans (e.g. radiography and fluoroscopy), are well adopted in image-guided interventions [12, 13]. The principle of these X-ray-based scans is based on distinct radiodensity between different body parts: body parts with a high radiodensity (e.g. bones) can attenuate incident X-ray from the detector on the other side. Such difference in radiodensity is measured by attenuation units that lie on the Hounsfield scale. Hence, the brightness of any spot on an X-ray detector represents the summation of the radiodensity of all materials along the line of X-ray projection.

Intra-op CT makes use of back-projection algorithms [14] to reconstruct the 3D tomography from multiple of radiographs imaged at different incident angles. To-date, intra-op Cone-beam CT (CBCT) scanners are usually adopted as they tend to have lower scan time. The average scan and reconstruction time required for a CBCT scan is typically 1 minute [15, 16]. The image resolution of CBCT scans is also usually quite high (typ. resolution = $0.2mm^3$) [17]. Intra-op CT is widely

adopted in numerous operations that require target localization, such as cardiac catheterization, neurosurgery, and radiotherapy.

However, the major drawback of the intra-op CT is the involvement of potentially harmful ionizing radiation. Also, as most soft tissues (e.g. brain and myocardium) have similar attenuation coefficients, X-ray-based imaging techniques produce a high contrast images of these tissues [18]. This inability hinders the sensitivity of the intra-op scanners, making accurate localization of soft tissue margins very difficult. Although increasing X-ray intensity can increase image contrast, it will also inevitably increase the radiation exposure to the patient [15]. As a result, to avoid excessive radiation dose acumination, frequent intra-op scans using CT is also not desirable. Thus, intra-op CT a less useful in applications that require constant updates of the position of surgical instruments and tissue margins by frequent imaging.

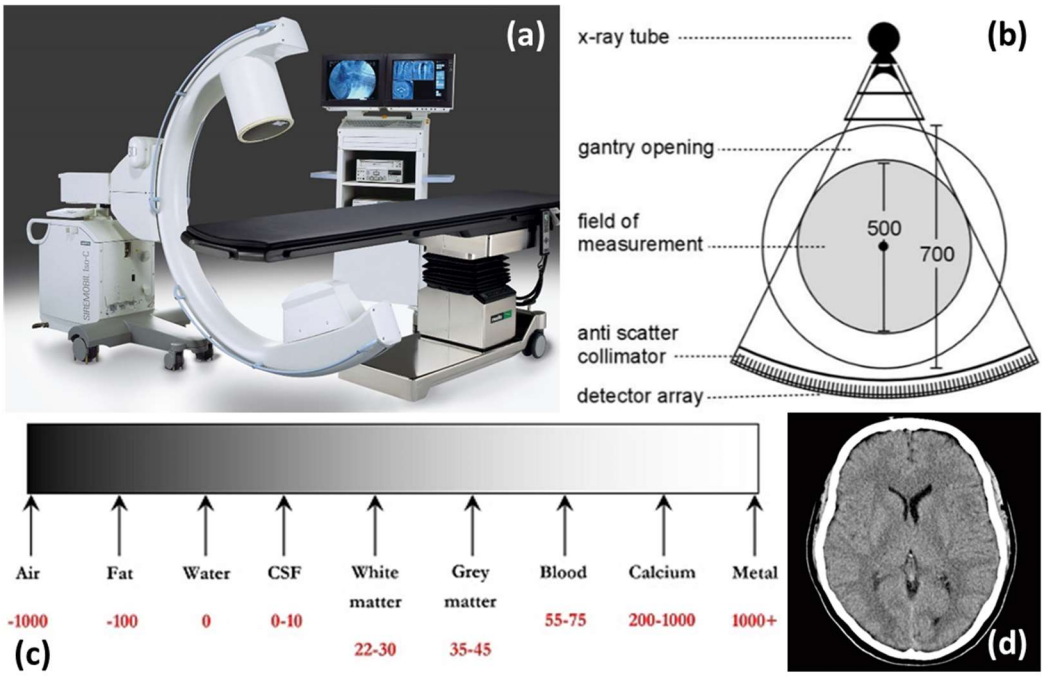

Figure 2.1 **(a)** Intra-op CBCT scanner. **(b)** Working principle of CBCT scanner. **(c)** Hounsfield scale used to quantify radiodensity. **(d)** Axial CT image of the brain showing the lateral ventricles.

2.2.2.2 Intraoperative MRI

Magnetic resonance imaging (MRI) is one of the more important imaging modalities. It is a non-invasive imaging technique due to the lack of involvement of ionizing radiation. Being first proposed in 1971, physicists had already found nuclear magnetic resonance useful in body tomography scans, as well as tumor diagnosis [19]. The basis of MRI is to excite and detect change in rotational direction of protons in water, which make up of most soft tissue. As a result, MRI is able to produce clear images of soft tissues with excellent contrast. Moreover, MRI can also reveal any physiological changes inside the body, including the formation of scars, edema, and fluid flow [20]. As such, MRI poses significant contributions to the actual clinical usage. For example, surgeons are able to directly visualize the effect of the ablation delivered by observing the scar/edema formed using MRI.

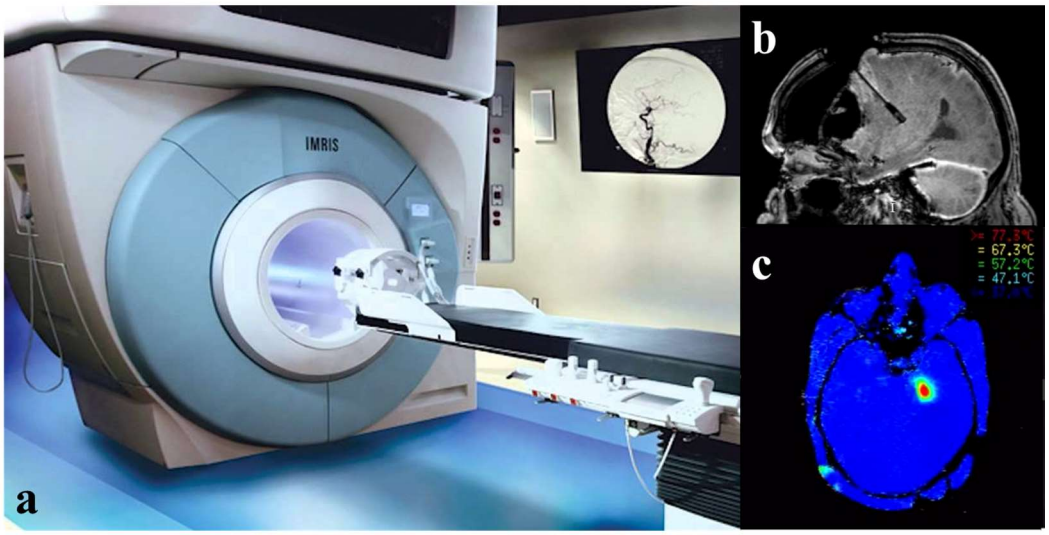

Figure 2.2 **(a)** Wide-bore iMRI scanner. **(b)** iMRI enables localization of surgical tool inserted into the deep brain region. **(c)** Intra-op MR thermometry allows visualization of the deep brain ablation progress.

There is a large clinical demand to employ MRI intraoperatively given the ability of imaging soft tissues in high contrast. However, most MRI scanners have to be close-bored to maintain a strong magnetic field, which precludes direct access to the patient. The long MRI scanning and reconstruction time required is also prohibiting. Therefore, the demand for intra-op MRI (iMRI) has motivated the development of open- or wide-bore MRI imagers to allow easier patient acces. The

first iMRI-guided neurosurgery performed in Brigham and Woman's Hospital with a double-donut shaped intra-op imager [21]. This double donut design of the MRI provides a spherical imaging volume with 30cm in diameter and a relatively wide (50cm) area for direct patient access [22]. However, the long reconstruction time can be limiting; the small area-of-access is also inconvenient.

As the technology advances, the scanning time for MRI can be drastically reduced. To-date, rapid MRI can be achieved under-sampling the *k*-space [23]; real-time MRI with a frame rate of 3.6 frame/s is also reported [24]. Fast low angle shot MR imaging (also known as "FLASH" MRI) is usually used for real-time MRI with a typical frame rate of 5 frames per second [25]. However, as the scanning time for MRI is directly proportional to the number of slice acquisition, these real-time 3D MRI images usually have a low resolution in terms of slice thickness. Despite having a low resolution, real-time MR image guidance can be achieved with the assistance of MR tracking coils [25-27]. Neurosurgery is one of the most important applications of real-time iMRI. Apart from neurosurgical intervention, iMRI has the potential to be applied in a wide range of stereotactic procedures on highly deformable body parts such as the breast, prostate and liver. In all, not only MRI can provide a continuous and clear image of soft tissue, but it is also non-invasive in nature. With the recent advancement of iMRI, these clear advantages have put it become one of the more promising intra-op imaging technique for various type of intervention procedures.

2.2.3 Clinical applications and potential problems

The emergent of intra-op imaging techniques enables safer interventional procedures. These intra-op images can, not only present the much-needed guidance of anatomical structures within the regions of interest, but also provide information that cannot be captured preoperatively. In this section, several surgical applications will be reviewed to illustrate the importance of intra-op image guidance and their potential problems.

2.2.3.1 Stereotactic neurosurgery

Stereotactic neurosurgeries involve placing objects or removing tissues to/from the brain. Typical examples are stereotactic biopsy [28], stereo-electroencephalography [29] and deep brain stimulation [30]. As the brain is extremely important and delicate, these interventions demand the highest accuracy and precision. Intra-op imaging is actively utilized to avoid any unintended damage, particularly to the brain's critical/functional regions. For example, imprecise positioning of the instrument can result in a deviated trajectory, which can significantly increase the risk of intracranial hemorrhage. In order to avoid damaging any critical tissues, an acceptable error of a mere 1-2mm in any neurosurgery is generally established [31].

However, deformation of the brain can cause significant issues peri-operatively. Since the brain is a soft, fluid-filled organ, it is not unusual for the brain to deform throughout the surgical process. This deformation, known as "brain shift", can be significant: intra-op brain deformation of 10mm after craniotomy is not uncommon (**Figure 2.3**) [33]. Such deformation would undoubtedly affect the accuracy of the preoperatively-gathered data, as well as the associated surgical plan. As such, surgeons will have to be conservative in order not to damage any healthy tissue near the treatment target.

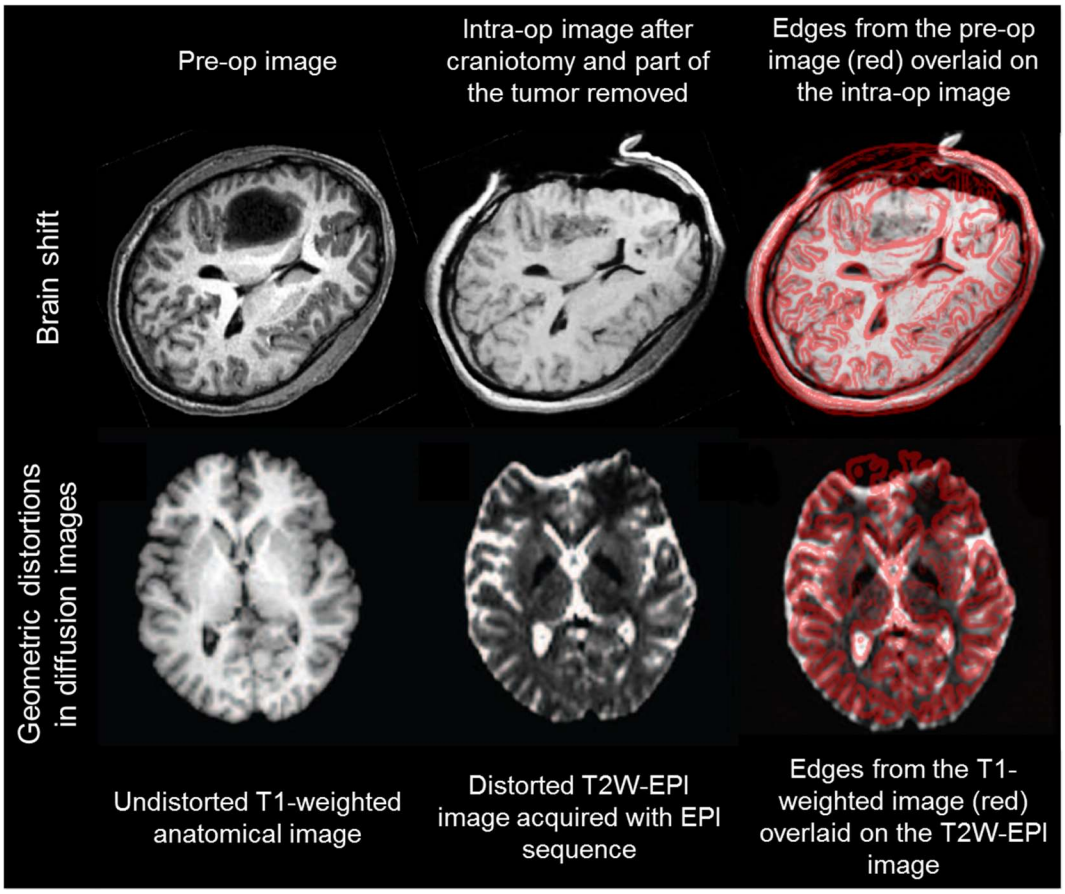

Figure 2.3 Brain shift after craniotomy and tumor removal [32] *(upper row)*; image distortion in diffusion MRI *(lower row)*. Misalignments between the images are visualized in the image overlay at the last column.

In light of the uncertainty introduced by brain deformation, image guidance is a fundamental part of any neurosurgical procedures. The intraoperatively acquired images enable accurate localization of the instrument and the target. Hence, surgeons can respond accordingly to compensate the misalignment due to brain shift. The first reported image-guided neurosurgical procedure is supported by intra-op CT [34]. Yet, as intra-op CT cannot provide high contrast images to visualize the soft tissue, it is insufficient to meet the supreme accuracy requirement in neurosurgeries. Other intra-op guidance approaches using ultrasound-based or optical-based guidance are also undesirable, due to the increased invasiveness to fulfill the requirement of having direct line-of-sight or contact of the dura [35].

To-date, iMRI-guided neurosurgeries are more prevalent as the technology advances. The iMRI images can reveal the anatomical structure in a detailed manner, thus allowing accurate localization of target tissue. Furthermore, iMRI also allows the surgeons to visualize the surgical effectiveness by performing scans peri-operatively. For example, in brain tumor resection, iMRI images can be used to guide the surgeons to clean up any remaining tumor with a stereotactic tool, thus, maximizing the effectiveness of the surgery. However, iMRI is susceptible to distortion. Let aside static field inhomogeneity, chemical shift, and susceptibility artifacts, the nonlinearity of the B_1 gradient field contributes most to such distortion. It has been reported that the spatial distortion can be as much as 25mm at the perimeter of an uncorrected 1.5T MRI. Even after standard gradient calibration, the error can still remain within the 1% range (typ. ~4mm). This error can be significant due to the straight accuracy requirement in stereotactic neurosurgeries.

In all, there exist a high demand for compensating the brain deformation and MRI distortion for neurosurgery. In order to provide a visual reference for neuronavigation, one can overlay/augment the predefined critical regions onto the intra-op image, provided that the deformation of the brain due to brain shift is known. Although the general deformation of "brain shift" can be visualized using iMRI, detailed localization of the critical tissue that requires advanced imaging techniques cannot be performed intraoperatively. The error brought by MRI distortion will also need to be resolved. This can be achieved by co-registering the intra-op images and the pre-op images. As rapid and frequent scans are required to ensure navigation accuracy, this co-registration have to be performed in a speedy manner to cope with the highly dynamic surgical workflow.

2.2.3.2 Cardiac catheterization

Cardiac catheterization is another example to showcase the importance of intra-op imaging. Particularly, cardiac electrophysiotherapy (EP), an effective treatment to cardiac fibrillation, require frequent intra-op imaging. Cardiac EP is performed by inserting a long (1.5m), thin (Ø2.67mm) catheter from the femoral vein to the left atrium perform ablation on the tissue. This ablation, known as pulmonary vein isolation, is guided by a pre-op electro-anatomical (EA) roadmap that is collected prior to the intervention. The pre-op EA roadmap is essential as it act as a fundamental visual reference to pinpoint the ablation targets.

Image guidance is the key to achieve a successful cardiac catheterization. Constant use of intra-op imaging techniques, including fluoroscopy and cardiac ultrasound, are required throughout the interventional procedure [37]. However, fluoroscopy images lack essential image contrast; cardiac ultrasound images can also be blurry and misleading. Furthermore, fluoroscopy and ultrasound images are unable to visually any physiological changes in response to the interventional procedure. For example, while scar formation around the pulmonary vein can guarantee complete isolation, edemas formed due incomplete ablation can allow recurrent atrial fibrillation even after the EP procedure. However, both fluoroscopy and cardiac ultrasound are unable to visualize such scars or edema.

The emergence of iMRI has opened up a new approach to achieving image guidance in the cardiac EP procedure. Not only MRI possesses the ability to visualize soft tissue clearly, but such guidance can also be used to stably steer the catheter to the lesion targets within the confined and rapidly deforming cardiac chamber. MR tracking coils can also be used to provide reliable localization of the tip of EP catheter [25-27]. Moreover, T2-weighted MRI can also be used to precisely and responsively monitor the physiological changes in cardiac tissues (**Figure 2.4**) [36, 38]. Such visual guidance can readily tell the surgeons whether the ablation has been successful by visualizing the scars or edema that arise after the ablation peri-operatively.

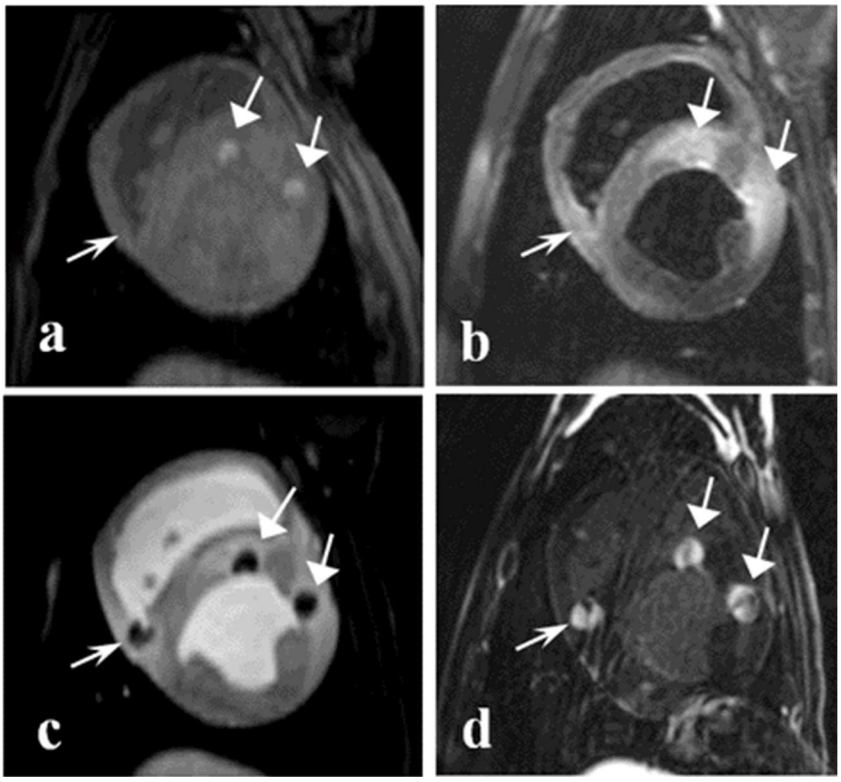

Figure 2.4 iMRI capable of visualizing scars and edema distinctively with different imaging sequences (e.g. T2-MRI, DT-MRI). Image retrieved from [36].

Numerous research groups have already conducted many clinical trials to demonstrate the value of iMRI guidance for cardiac EP in the clinical routine. However, despite the ability to visualize the tissue morphologies perioperatively, image-guided cardiac EP lacks real-time update to the roadmap according to the intra-op image. A reliable visual reference can only be established to guide the catheter if the EA map is able to be mapped and overlaid on the intra-op image. However, to overlay the EA map, the pre-op and the intra-op images will have to be co-registered. Such registration is essential to restore the deformation of the tissues due to rapid beating motion of the myocardium. Nonetheless, this co-registration between the pre-op and intra-op images is very challenging due to the possibility of having a high image mismatch. Furthermore, although non-rigid image registration algorithms are available, the major bottleneck resides in the computation time for the registration, in which a long computation time can be considered as clinically impractical for MRI-guided EP.

2.2.3.3 Intensity-modulated radiotherapy

The aim of intensity-modulated radiotherapy (IMRT) is to deliver an effective radiation dose to the tumor while minimizing the dose to the surrounding tissues. Current practice of IMRT acquires a detailed pre-op plan composed by high-resolution CT/MRI images to define the clinical target volume. During the treatment planning process, the radiation delivery sequences are optimized to make sure the clinical target volume receive sufficient radiation dose. However, this clinical target volume will inevitably morph over time due to tissue deformation, tumor shrinkage, as well as weight loss. Such misalignment can be significant, especially in radiotherapy treatments that span multiple weeks [39]. Treatment re-planning is therefore usually required. This morphing can often lead to a mismatch between the pre-op image and the actual tumor location, particularly the boundary of the planned tumor volume can be affected (**Figure 2.5**).

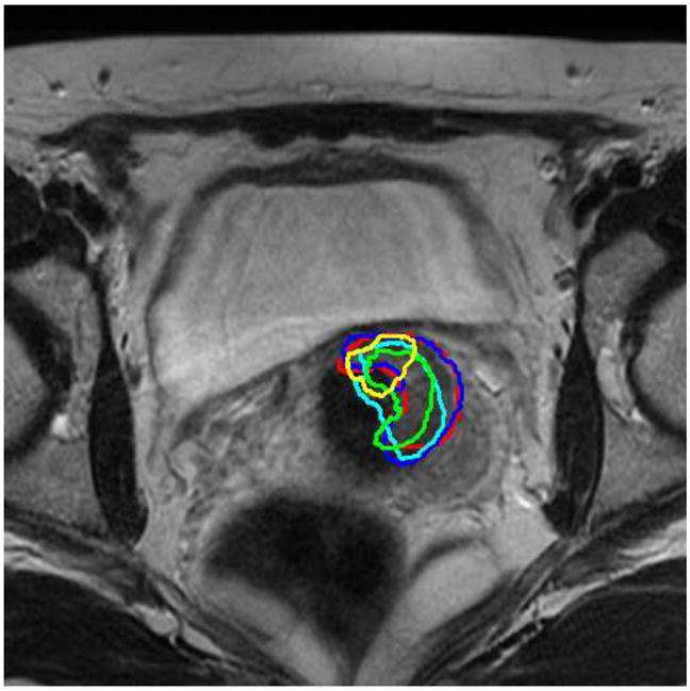

Figure 2.5 Tumor contours from 5 distinct treatment days overlaid on pre-op MRI image. Noticeable shrinkage of the tumor can lead to radiation overdose of surrounding normal tissue, as well as radiation under-dose to the target volume. Image retrieved from [39].

The change in clinical target volume due to extensive tissue deformation is undesirable. As uncompensated misalignment between the treatment plan and real-time anatomy can lead to radiation under-dosage to the treatment target, or over-dosage to surrounding normal tissue or organ. With the advancement of intra-op imaging techniques, image-guided radiotherapy (IGRT) adds another layer of protection by allowing CBCT images to be acquired immediately prior to the treatment [40]. Once the intra-op image is obtained, necessary adjustments to the radiation delivery plan as well as a re-evaluation of the dosimetric deposition can be performed accordingly [41].

However, there is a constant demand for co-registering the pre-op treatment plan with the intra-op CBCT images for effective radiation dose delivery. Despite IGRT can visual the deformation of the clinical target volume, the computation required to non-rigidly aligning the two images can be time-consuming. Therefore, IGRT is not extensively used for every patient receiving radiotherapy, but is only adopted in a few high-risk patients.

2.3 Demands for intra-op non-rigid image registration

Advances in image-guided techniques, especially with the recently-emerged iMRI, enable visual guidance to identify the anatomical target of interest during the procedure. To date, recent intra-op navigation systems are also capable to virtually augment the preoperatively segmented critical/target tissues on the intra-op image [42, 43]. However, surgical interventions can induce large-scale tissue deformation. Also, most images acquired by intra-op imagers are also prone to distortion. For example, CBCT images can be susceptible to blooming effect at regions with high radiodensity gradient [44]. iMRI images are also susceptible to nonlinear distortion due to B-field gradient inhomogeneity (**Figure 2.3**) [45]. The combined effect of both surgical interventions and intra-op image distortion can induce large misalignment between the pre-op and intra-op images. Such misalignment can make the surgical plan very inconsistent to the actual anatomy during the intervention.

Many intra-op navigation systems employ *rigid registration* to align the pre- and intra-op images. However, rigid registration cannot compensate for any non-

linear image discrepancies due to the aforementioned reasons. Besides, *non-rigid registration* possesses the ability to recover the potential misalignment between the pre-op and intra-op images. However, non-rigid registration schemes are generally set back by their high computation requirement. As frequent intra-op scans will be performed during operation, non-rigid registration will have to be performed in a fast and frequent manner. In this section, a survey on different non-rigid image registration strategies is presented. The performance of these registration schemes on registering intra-op MR images will also be discussed based on three fundamental aspects: accuracy, robustness and computation requirement.

2.3.1 Overview

The goal of image registration is to determine an optimal transformation (T) between the fixed image (F) and the moving image (M). Such transformation represents the mapping of image features between two corresponding, misaligned images. In geometry, the equation below depicts the transformation T [46]:

$$T\colon\ M \mapsto F \Leftrightarrow T(M) = F \qquad (1)$$

Of which one can perceive that, given an accurate transformation T, the transformed moving image $T(M)$ yields an exact image as the fixed image F. This transformation is determined by image registration. Besides, the transformation can be represented in different formats. For example, *rigid* image registration yields a matrix that defines translation, rotation, scaling or sometimes affine transformation. In contrast, *non-rigid* image registration yields a set of transformation parameters that describe the deformation between two images. In general, rigid image registration can be performed very quickly, despite having a poor registration accuracy. Non-rigid image registration possesses the potential to achieve very high registration accuracy, however, is much more computationally demanding.

This thesis focuses on non-rigid image registration, as rigid registration is inimical to the high accuracy demand for intra-op use. In all, there are two major approaches to non-rigidly register the images: feature-based and intensity based. Both approaches rely on pixel intensities to register the fixed and moving images, but the former approach performs the registration on a higher level by evaluating

image features using clues points, lines or even area. In contrast, intensity-based image registration approaches make uses of the native information provided by the image intensity, for example intensity gradient and pixel-wise intensity difference, to register the images. Details of these approaches will be discussed in 2.3.2 and 2.3.3 .

2.3.2 Feature-based non-rigid image registration

Feature-based image registration is a common approach to non-rigidly register two images. The basis of feature-based non-rigid image registration relies on the identification and matching of distinctive features. There are four major steps involved in feature-based image registration, namely feature detection, feature matching, transformation model optimization, and image resampling [47].

Image features detected on images to be registered are often called as control points (CP). Feature-based registration framework matches and evaluates the correspondences between two sets of CPs extracted from the fixed and moving images. Such image features can be points, lines, or even enclosed regions (**Figure 2.6**). For example, a bright/dark spot/line/area in an image can be eligible for CP detection. The detection of CP on the image shall preferably be performed in an automated manner. In this light, many methods are developed to automatically detect the control points for image registration. For example, point features can be detected by intersections [48] or local curvature discontinuities using Gabor wavelets [49]; line features can be detected by Canny edge filters [50]; region features can be detected using automatic segmentation methods [51].

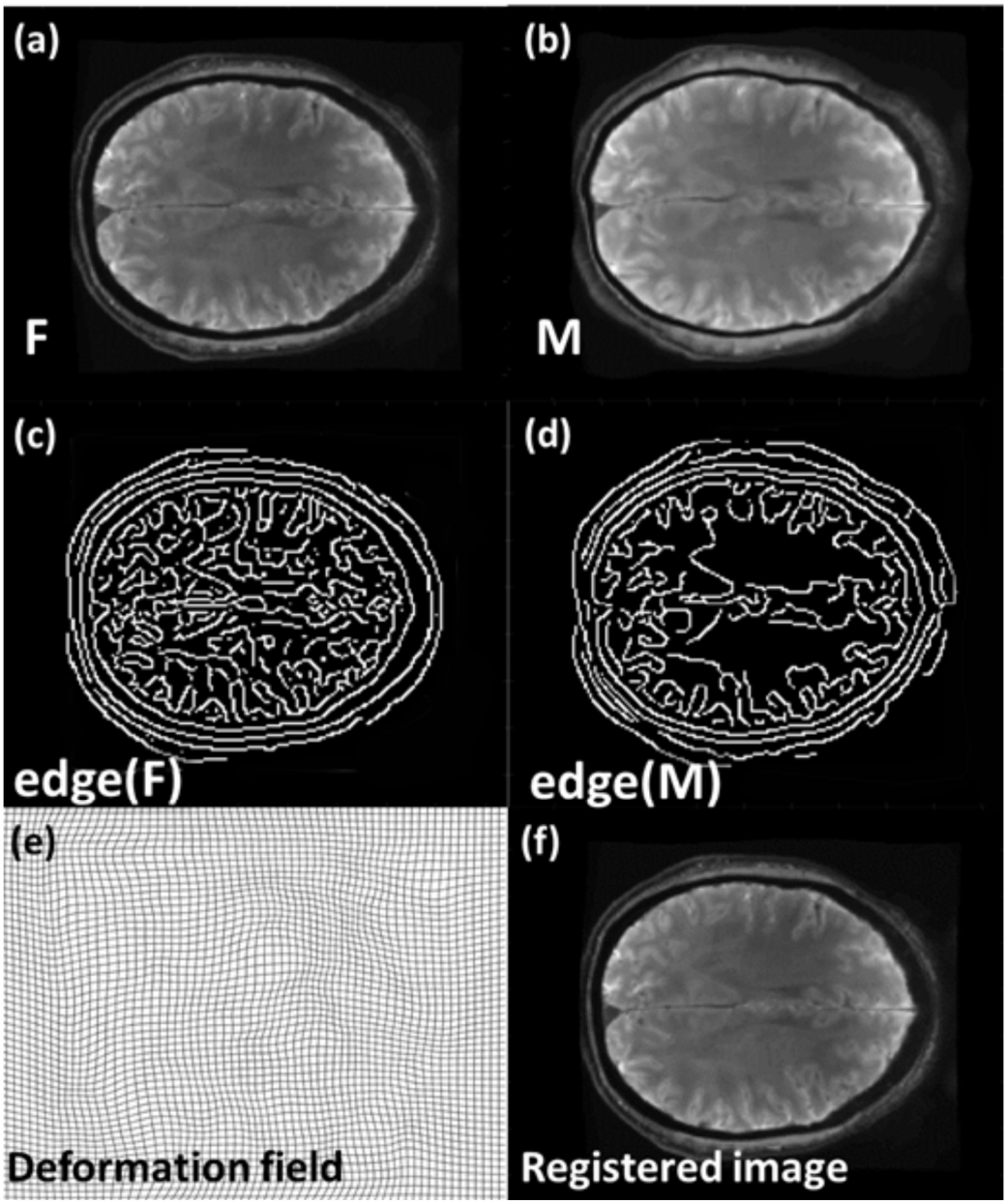

Figure 2.6 Feature-based image registration framework using CPs. CPs from the edge features **(d-e)** of the **(a)** fixed and the **(b)** moving image are extracted using a Canny edge filter. **(e-f)** The resultant deformation field can be generated accordingly to realign the images.

The detected CP on the fixed and moving images will have to be matched in order to calculate the transformation between the images. Such matching can be performed, not only using the intensity values at the corresponding pixels of CPs but also the spatial distribution of the feature. Clustering techniques are often employed to match the CP being connected by an abstract edge or a line [52]. Once the features are matched, a transformation parameter can be computed to determine the spatial transformation between the two images.

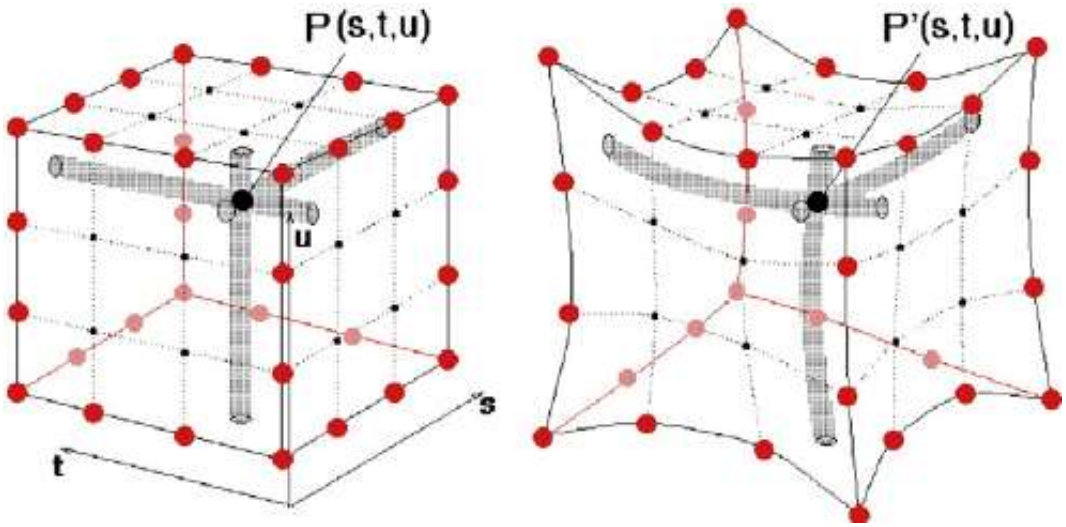

Figure 2.7 Free-form deformation transforming the point *P* to *P'* defined by re-alignment of control points (red dots). Image retrieved from [53].

The transformation model used in feature-based image registration is parametric in nature. Free-form deformations (FFD) are often used to parameterize the deformation. FFD define a mesh of passive grid points that govern the deformation of the image, with the edges of each mesh cell defined using spline lines (**Figure 2.7**). Such spline lines can be B-spline, thin-plate spline (TPS) or others. In feature-based image registration using FFD, the detected CP pair on the fixed and moving images will be aligned altering/deforming the passive deformation grids. The iterative closest point algorithm is one of the well-known and robust approaches to register the control points in 3D [54]. However, the techniques to perform such re-alignment remains to be one of the most studied topic to-date.

To facilitate the search of optimal deformation parameters, most feature-based image registrations are usually conducted in an iterative manner. The resultant mesh grid describing the deformation can be also used as a parametric mapping function that depicts the transformation of the moving image and thus registering/re-aligning two images. In the last step of feature-based image registration, the moving image is deformed (or "warped") by interpolating the image according to the mapping function. This interpolation is an essential step to transform the moving image by the mapping parameters. A wide variety of interpolants are used in the warping process, including trilinear/tricubic functions,

spline functions, Gaussian functions, truncated sinc functions, along with many others [47]. Such interpolation strategy is deterministic to the accuracy of the transformation that describes the misalignment. Despite having a wide variety of choice in terms of interpolants, trilinear interpolation is generally considered as a trade-off between accuracy and computation efficiency.

2.3.3 Intensity-based non-rigid registration

Intensity-based image registration, in contrary to its feature-based counterpart, does not require automated detection of image features to register the image. Instead, it directly exploits the pixel/voxel intensity values to perform the co-registration. Most intensity-based image registration schemes carry the assumption that the pixel values contain enough information for the image registration process, which happens to be true in most cases. Particularly, the *"demons algorithm"* is one of the most renowned intensity-based image registration schemes. In this sub-section, a comprehensive review of the *demons* algorithm and its variants will be presented.

2.3.3.1 Demons algorithm

Thirion [55] first proposed the *demons algorithm* in 1998. The name of this algorithm is inspired by its analogy to *Maxwell's demons* in the field of thermodynamics. The main concept of the *demons algorithm* resides in the optical flow of the pixels/voxels. This concept is similar to the particle diffusion process: concentration gradient across a membrane drives particle movement. Similarly, intensity gradient between the mismatched images drives the intensity to "diffuse" across the pixels/voxels boundary. Therefore, the deformation in the *demons* algorithm is driven by the intensity gradient. In theory, the algorithm can register any datasets as long as the intensities between the fixed and moving images are conserved. The original *demons* algorithm is often referred to "*additive demons*" as the addition operations is used to manipulate the vector fields. The pseudocode in **Algorithm 2.1** describes the general framework used in the *demons* algorithm [55].

Algorithm 2.1 Pseudocode showing the general framework used in the *additive demons* algorithm.

Pseudocode: General framework for the additive *demons* algorithm

Input: Fixed image F and moving image M

Do until Harmonic energy (E) is minimized**:**

    Compute transformation update u_i based on F and $s_i(M)$

    Update transformation for next iteration $s_{i+1} \leftarrow s_i + u_i$

    Evaluate Harmonic energy (E)

Output: Transformation $s = s_i$ from M to F

The *demons* algorithm is an iterative framework which updates the deformation field s in a step-wise manner. The deformation field is updated by the adding the update vector field u in each iteration. Such update is driven by the "forces" that depends on the both image gradient and pixel/voxel intensity difference of the fixed and transformed moving image. The resultant transformation that depicts the pixel/voxel displacement is represented by a deformation field s. Note that this registration scheme is non-parametric in nature, as the transformation/displacement of every pixel/voxel can be independent to each other. As the deformation update is driven by the pixel/voxel information of the input images, the algorithm is also considered to be stable, and it will converge unconditionally over iterations.

To date, a couple of "*demons force*" variants were developed to improve the accuracy and robustness of the algorithm [56-60]. Different "*demons force*" formulation, including passive force, active force, symmetric forces, among others, attempts to improve the registration by accelerating the convergence rate of the registration. Despite these variants of "demon's force" are distinguishable to each other, the main concept of the whole registration algorithm remains unchanged. **Table 2.1** below also provides a comprehensive summary of different updating schemes developed.

Table 2.1 Summary of different update rules under the *additive demons* framework.

Name	Update rule	Ref.

Passive Force	$u_i = \frac{(s_i(M) - F)\nabla F}{(s_i(M) - F)^2 + \lvert\nabla F\rvert^2}$	[55]
Evolved Passive Force	$u_i = \frac{4(s_i(M) - F)\nabla F\lvert\nabla F\rvert\lvert\nabla M\rvert}{[2(s_i(M) - F)^2 + \lvert\nabla F\rvert^2 + \lvert\nabla M\rvert^2](\lvert\nabla F\rvert^2 + \lvert\nabla M\rvert^2)}$	[56]
Active Force	$u_i = \frac{(s_i(M) - F)\nabla(s_i(M))}{(s_i(M) - F)^2 + \nabla(s_i(M))}$	[57]
Double Force	$u_i = \frac{(s_i(M) - F)\nabla F}{(s_i(M) - F)^2 + \lvert\nabla F\rvert^2} + \frac{(s_i(M) - F)\nabla(s_i(M))}{(s_i(M) - F)^2 + (\nabla s_i(M))}$	[57, 58]
Adjusted double force	$u_i = \frac{(s_i(M) - F)\nabla F}{\alpha^2(s_i(M) - F)^2 + \lvert\nabla F\rvert^2} + \frac{(s_i(M) - F)\nabla(s_i(M))}{\alpha^2(s_i(M) - F)^2 + (\nabla s_i(M))}$	[57, 59]
Inverse Consistent	$u_i = \frac{\left(s_i(M) - s_i^{-1}(F)\right)\left(\nabla(s_i(\mathrm{M})) + \nabla\left(s_i^{-1}(F)\right)\right)}{\left(s_i(M) - s_i^{-1}(F)\right)^2 + \left\lvert\nabla\left(s_i^{-1}(F)\right) + \nabla(s_i(M))\right\rvert^2}$	[60]

* $\nabla(\cdot)$ denotes the pixel gradient of an image.

As the *demons* algorithm is iterative in nature, the whole registration process can be considered as an optimization problem. Pennic et al [59] first proposed an optimization framework for the *demons* algorithm. Vercauteren et al. also showed that the inverse consistent method [60] can be cast to the efficient second-order minimization framework [61][62, 63], which is able to solve for the target transformation effectively solved using gradient descent and/or variational schemes [59, 64].

To compute the best transformation field, these optimization schemes evaluate and minimize the cost function, known as the Harmonic Energy (E). This harmonic energy not only compare the similarity (Sim) between the fixed image and the transformed moving images, but also considered the likelihood (Reg) for such deformation to occur:

$$E = \frac{1}{\sigma_i^2} Sim\big(F, s_i(M)\big) + \frac{1}{\sigma_T^2} Reg(s) \qquad (2)$$

Where σ_i and σ_T are the regulation terms for the optimization. Minimization of this harmonic energy is achieved by optimizing the similarity term and the likelihood term in an alternative manner [61]. Particularly, the similarity term is usually evaluated in terms of mean squared error (MSE). The likelihood of having such a deformation field is usually determined by the field Jacobean, which indicates the local stretch, shear and, rotation of the field. Cachier et al. also proposed an auxiliary correspondence term regarding to iconic features during the evaluation of the harmonic energy [65]. This auxiliary term can stabilize the optimization process of the regularization term throughout the registration process.

The *additive demons* framework has been well established and widely adopted. However, the additive nature of the deformation update disregards the fact that the deformation update vector field is representing a spatial transformation. Therefore, addition or subtraction of a vector field does not necessarily preserve the topology of the field [66, 67]. As a result, one-to-one mapping between the pre- and post-transformed image cannot be guaranteed. The absence of one-to-one mapping implies that any deformation fields generated under the addictive framework are not invertible. This incapability of inverting the deformation field is illustrated in **Figure 2.8**. As such, the additive update scheme can only provide an approximation of the field after transformation. This approximation will not hold valid upon large deformation, or when the field is being updated in multiple instances.

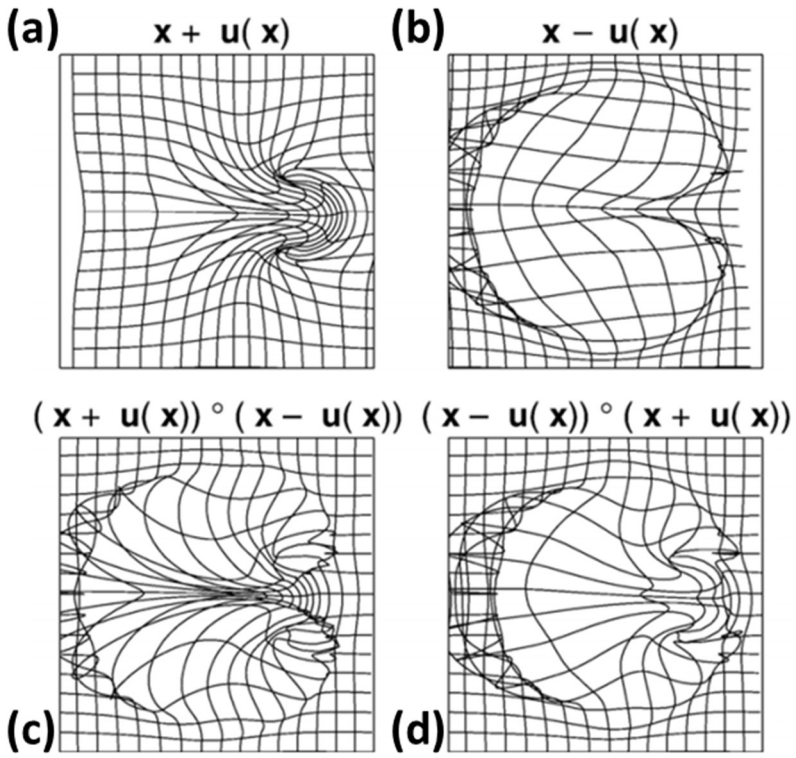

Figure 2.8 Test of invertibility for additive spatial transformation. **(a-b)** Opposite transformation yielded by vector field addition/subtraction. **(c-d)** Composite of these opposite transformation cannot cancel out each other, indicating the fields are not invertible. Image retrieved from [66].

2.3.3.2 Diffeomorphic log-demons algorithm

Ashburner et al. [66] introduced the Diffeomorphic Anatomical Registration using Exponentiated Lie Algebra (DARTEL) algorithm, which utilizes diffeomorphism to allow large deformation for image registration. Diffeomorphism is the transformation of a differentiable manifold that belongs to the Lie Group [68]. It processes a lot of beneficial mathematical properties which are desirable in image registration. The most remarkable property that makes diffeomorphism useful is that the topology can be preserved even when there is a large deformation. Folding of a diffeomorphic manifold is also not possible. As the diffeomorphic transformation preserves image topology, global one-to-one mapping is guaranteed. Thus, given any diffeomorphic transformation field, it is possible to "undo" such transformation by providing a complementary backward transformation.

In all spatial transformation under the diffeomorphic framework, mapping an image with the transformation s is equivalent to composing the image with s. Similarly, consecutive update to the mapping with multiple transformations $\{s_1, s_2, \dots, s_n\}$ is equivalent to consecutive compositions on the image. Such composition is achieved by resampling and interpolating one field by another. Therefore:

$$s(x) = s \circ x \tag{3}$$

$$s_n\{\dots s_2[s_1(x)]\} = s_n \circ \dots \circ s_2 \circ s_1 \circ x \tag{4}$$

Contrast to additive updates which provide only an approximation, this compositive update scheme provides an accurate representation of the mapping even after multiple transformations.

In the DARTEL algorithm, the diffeomorphic deformation Φ is no longer defined by the deformation field. Instead, the deformation is defined by a stationary vector field (SVF), v. Such model yields the differential equation [66]:

$$\left.\frac{d\Phi}{dt}\right|_{t=t'} = v(\Phi|_{t=t'}) \tag{5}$$

which describes the evolution of the deformation field starting with an identity transform $\Phi|_{t=0} = 0$ to the final transformation $\Phi|_{t=1}$. Traditionally, simple integration methods can be used to compute the solution. However, considering integration will take place on every voxel, the algorithm can be very computationally expensive. As the velocity vector field is the time divertive of the diffeomorphic deformation field that belongs to the Lie group, the vector field itself can be considered as a member of Lie group structure that resides within the log-domain [68]. Therefore, Lie algebra can be applied, and the resultant deformation field can be obtained by computing exponentiating the SVF:

$$\Phi^{(1)} = \exp(v) \tag{6}$$

This relationship between the velocity vector field v and the deformation field Φ is of utmost importance in the development of *diffeomorphic log-demons* [69]. Similar work also exploit SVF to deal with diffeomorphic transformations [70].

With the relationship between the velocity vector field and diffeomorphic deformation field being established, Vercauteren extended the *demons* algorithm and incorporated the diffeomorphic framework into it [69]. By introducing the SVF used in DARTEL [66], the *diffeomorphic log-demons* algorithm now enforces diffeomorphism under the *demons* framework. Similar to the *additive demons*, an update field $\boldsymbol{u}$ is also computed according to the image intensity gradient as well as pixel/voxel-wise intensity difference. However, *diffeomorphic log-demons* apply this update on the velocity field $\boldsymbol{v}$, which indirectly update the deformation transformation field under the relationship $\boldsymbol{s} = \exp(\boldsymbol{v})$. By frequently updating the velocity vector field in the algorithm, the transformation field can be ensured to be diffeomorphic at all times. The algorithm also performs vector field regularization to discourage excessive update, as well as prevent unrealistic deformation field. This regularization is commonly done by applying Gaussian smoothing.

Given the *diffeomorphic log-demons* algorithm is under the umbrella of the *demons* registration framework, it can also be considered as an optimization problem [72]. Therefore, Newton's method can also be used to resolve this optimization problem [61]. The cost function for optimization used is also unchanged. In all, the pseudocode presented in **Algorithm 2.2** illustrates *diffeomorphic log-demons* presented in [69, 71]:

Algorithm 2.2 Pseudocode showing the iterative registration process in the *diffeomorphic log-demons* algorithm.

Pseudocode: *diffeomorphic log-demons* algorithm

Input: Fixed image $\boldsymbol{F}$ and moving image $\boldsymbol{M}$

In each iteration i **Do**:

  Compute update field update $\boldsymbol{u}_i$ based on $\boldsymbol{F}$ and $\boldsymbol{M}_i$

  Apply fluid-like regularization: $\boldsymbol{u}_i \leftarrow K_f \star \boldsymbol{u}_i$

  Update velocity field: $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} \circ \boldsymbol{u}_i$

  Apply diffusion-like regularization: $\boldsymbol{v}_i \leftarrow K_d \star \boldsymbol{v}_i$

  Compute deformation field $\boldsymbol{s} \equiv \exp(\boldsymbol{v})$:

  Update the moving image $\boldsymbol{M}_{i+1} = \boldsymbol{M} \circ \boldsymbol{s}_i$

  Evaluate Harmonic energy $\boldsymbol{E}$

Until Harmonic energy (E) is minimized

Output: Deformation field s_i from $\boldsymbol{M}$ to $\boldsymbol{F}$

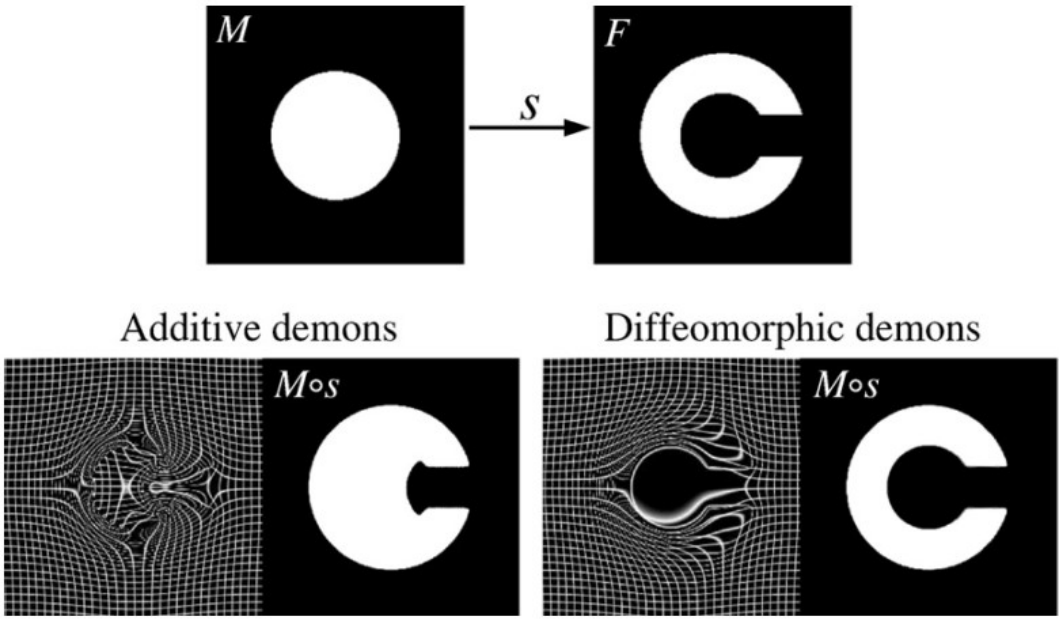

Figure 2.9 "Circle to C" registration for *additive demons* and *diffeomorphic log-demons*. *Additive demons* failed to converge. In contrast, *diffeomorphic log-demons* is able to register the images with a smooth, invertible field. Image retrieved from [66].

A practical and fast approximation of the field exponentials algorithm is developed in [72], which utilizes the "scaling-and-squaring" method to compute the vector field exponentiation. This approximation, which will be introduced in the chapters afterwards, enables fast computation of the field exponential. With the trait of diffeomorphic deformation, *diffeomorphic log-demons* is well known for its capability of registering large deformations. The classical "Circle-to-C" registration demonstrated that *diffeomorphic log-demons* is much more reliable than traditional *additive demons* registration approach (**Figure 2.9**).

Klein et al. also provided a comprehensive survey and comparison of *diffeomorphic log-demons* with other non-rigid registration schemes [73]. The *diffeomorphic log-demons* algorithm is one of the faster registration schemes with good accuracy, compared to its feature-based counterpart. However, despite the algorithm can produce a smooth and invertible deformation field, the update of the velocity field in *diffeomorphic log-demons* relies on the local intensity gradient of the images [74]. This "localized" update discourages correspondence search of highly deformed or low-contrast images. These potential problems can be tackled by employing multiresolution schemes to register a largely deformed image. A more lenient regularization scheme can also improve the convergence of the algorithm.

2.3.4 Application on registering intraoperative images

Registering intra-op images is a very challenging task. As most intra-op imaging techniques sacrifice image quality for temporal resolution, intra-op images can be noisy and rich in artifacts [12, 75]. For example, low-dose CBCT images, in general, have lower signal-to-noise ratio [12]. The B_0 field in iMRI being substantially lower than conventional high-field MRI can also induce considerable amount of noise and artifacts [21]. Besides, reconstruction of intra-op images may also employ empirical approximation to accelerate the process, further decreasing the image quality [76]. Let aside all problems brought by excessive noise and artifact in intra-op scans, the non-rigid registration process have to be performed in a rapid and automated manner to avoid interference to the surgical workflow. In such regard, three major qualities have to be considered in choosing the best registration strategy: computation speed, accuracy, and reliability.

2.3.4.1 Feature-based registration

In feature-based registration methods utilizing FFD, the degree-of-freedom (DoF) defines the number of parameters that the algorithm will have to optimize for. As feature-based registration algorithms tessellate the image into a mesh of deformation grids that define the global transformation, employing fine-grained deformation mesh increases the DoF of the registration. With higher DoF, the registration can become more accurate. However, the computation required for high DoF will also increase significantly. Therefore, trade-offs between registration accuracy and computation time will have to be made. As the co-registration have to be computed in a timely manner in the intra-op scenario, the number of DoF used in the feature-based image registration scheme will be limited, leading to concern of accuracy. Further, increasing the DoF can make the optimization to become ill-posed due to the curse of dimensionality [77]. Furthermore, the existence of local minima upon high DoF is also unavoidable, which can be one of the robustness concern in feature-based image registration.

Another concern of feature-based registration on the reliability of CP detection. Although the feature-based registration schemes can utilize automated CP detection schemes, the detection of CP is heavily influenced by the image noises and artifacts. Most intra-op images suffer from lower image contrast, lower signal-to-noise ratio, and lower resolution. Therefore, it is often challenging for any algorithms to extract the CPs on the intra-op images automatically. Further, the pre-op image may exist unmatchable "outlier" features that do not exist in the intra-op image (e.g. tumor being extracted during surgery will not present in the intra-op image) [78]. This outlier problem is aggregated as delicate feature landmarks detectable on pre-op images may not be able to be easily detectable in the intra-op images. As a result, extensive preprocessing work is often required for optimal registration. Manual actions are therefore mandatory and critical to ensure the CPs are detected are in good conditions. In all, the registration reliability of feature-based image registration prone to the accuracy of the detected features.

2.3.4.2 Intensity-based registration

Contrast to its feature-based counterpart, the intensity-based image registration is non-parametric in nature. The non-parametric deformation field used in intensity-based image registration possesses a very high DoF and carry the potential of having a very accurate registration result. The registration can also converge unconditionally given that the deformation is driven continuously by the image difference and intensity gradient. In particular, the *diffeomorphic-log demons* algorithm provided a good framework to readily optimize the deformation field. Despite the registration is also susceptible to local minima due to high DoF, such local minima can be worked around by employing multi-resolution approaches and appropriate regularization [55, 79, 80]. In multi-resolution approaches, the algorithm registers the image in a pyramidal manner. Larger-scale deformations can be registered with relative ease using a subsampled image, while accurate deformation representation will then be registered subsequently after the larger-scale deformation is found.

In general, the intensity-based image registration algorithm is more reliable than feature-based image registration, due to the non-necessity of employing automatic control point detections. Convergence is also guaranteed by the algorithm. Manual tuning of the regularization parameters used in the optimization cost function may be required, but these fine-tuning operations do not pose critical influence on the registration process.

However, intensity-based image registration suffers from the heavy computation required to register the images. The overall registration process is in general slow, due to the iterative optimization nature of intensity-based image registration. As reported in [69], the *diffeomorphic log-demons* require 2 minutes and 30 seconds to briefly register a single image; more than 8 minutes is required to completely register a high-resolution image [73] with 2.8M voxels.

2.3.4.3 Summary

Previous sub-sections have summarized the pros and cons of the feature-based and intensity-based image registration. In the application for registering intra-op images, apart from registration accuracy, other factors such as reliability and computation speed are also concerned.

As discussed, the feature-based image registration schemes rely on automated detection of the image features. Therefore, the robustness of the registration depends on the intra-op image quality, feature extraction methods, and DoF of the registration. However, as most intra-op images sacrifice image quality in an exchange for reduced scanning and reconstruction time, such automated feature extraction schemes may not be reliable due to the presence of considerable noise and artifacts. This inherent disadvantage had put feature-based registration unreliable in registering intra-op images.

In contrast, intensity-based image registration schemes do not require image preprocessing, as the deformation is solely driven by the underlying image difference and pixel gradient. Such advantage can make intensity-based registration superior to its feature-based counterpart in terms of reliability. However, the computation demand of the algorithm hinders its ability to in registering intra-op image quickly. Nonetheless, such downsides are not inherently related to the algorithm, but on the computation perspective. Such registration schemes may be eligible to be accelerated using advanced application accelerators.

2.4 Current trends of high-performance intra-op registration

The importance of intra-op surgical navigation and non-rigid image registration has been discussed in previous sections. Despite the co-registered images can provide invaluable image guidance to the surgeons, these registered images will have to be provided in a timely manner to avoid disrupting the surgical workflow. Furthermore, the tissue margins updated by such co-registration can be provided in an asynchronous manner [81, 82]. **Figure 2.10** provides a general flowchart illustrating such asynchronous loop for intra-op surgical navigation with visual guidance using intra-op imagers.

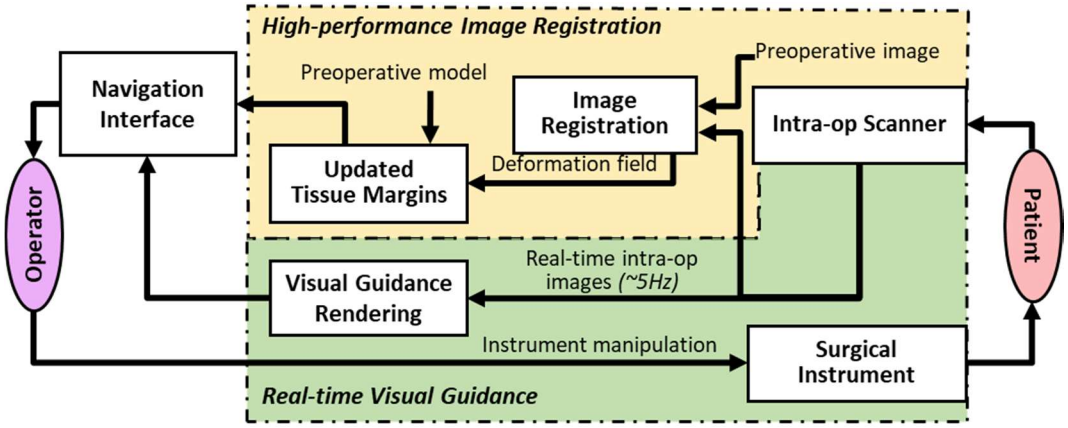

Figure 2.10 Image registration and visual guidance working asynchronously in the navigation interface. Pre-op model and the intra-op images can be swiftly aligned by high-performance image registration *(top)*. Intra-op scanning also enables continual position tracking of the surgical instruments *(bottom)*.

Intensity-based image registration schemes are more preferably adapted to register intra-op images during surgery due to its potential high registration accuracy and robustness. However, the intensity registration approach is set back by its own computation requirement. Substantial pixel-wise computation is required to retrieve the correspondence between the pre-op and intra-op images. Therefore, these computations will need to be accelerated and optimized to enable intensity-based registration to be used in the clinical practice. In this light, application accelerators and/or coprocessors such as GPU can be used to optimize the computation. In this section, the general properties of GPU will be introduced.A horizon scan of computational acceleration for intensity-based image registration will also be conducted.

2.4.1 Graphics processing units as application accelerator

Previous sub-sections have illustrated the underlying clinical demand of having a fast non-rigid image registration. However, there is a clear gap between the formulation and the application of these intensity-based image registration algorithm. Their extensive computation time required is the main concern. With the advancement of application accelerators such as GPU in the recent decade, there were high hopes in leveraging the accelerator's parallel processing power for fast registration.

Amongst a variety of application accelerators, GPU is a specialized hardware originally being developed for rendering images for display output. As the pixel intensity values encoded in the output display signal are independent, these output values are computed separately inside the GPU. Contrast to the central processing unit (CPU) which optimizes for single-threaded computation latency, the GPU is designed to have numerous computation cores for highly parallelized computation. This architectural advantage can be exploited to perform complex scientific calculations, such as particle simulation, image reconstruction, and optimization. As intensity-based image registration also involves a lot of pixel-wise operations, the GPU is a good choice to accelerate the registration.

2.4.1.1 Compute Unified Device Architecture

Even with an advantage of parallel processing a large amount of structured data, the potential of GPU to perform general purpose computing had been overlooked until recently. In the earlier developmental stages of GPU, these exist no tools for scientists to exploit GPU's capability of parallel computing. Returning data from the GPU back to the CPU was impossible in many of the earlier GPU models. The data paths of these earlier models are often hard-wired, thus limiting their parallel computation potential.

In 2006, NVidia released Compute Unified Device Architecture (CUDA), an application programming interface (API) for general purpose computing using its CUDA-enabled GPU. Such API allow bi-directional communication between the GPU and the host CPU. Therefore, CUDA enables the GPU to act as a coprocessor to run customized subroutines for general applications. Since then, the term "General Purpose GPU" is coined, and nearly all newer released GPUs are customizable for accelerating specific computation tasks.

It should be noted that AMD also released a stream computing software development kit (SDK) based on Brook for general purpose computing with their GPU. In this thesis, we will only focus on NVidia's CUDA devices, as it has been reported that CUDA devices outperform their AMD counterparts [83]. However, the concerned performance-aware computation enhancement techniques presented in this thesis are largely translatable between two mainstream GPUs.

2.4.1.2 CUDA programming model

General-purpose computing can be achieved by programming the CUDA GPU using the C++-alike CUDA language, which enables customized computation kernels to be launched on the GPU by allowing access to the GPU's memory, instruction sets, and computation elements. Subsequently, the computation kernels written in CUDA language will be compiled into parallel thread execution (PTX) instruction sets, which are low-level instruction sets that are optimized for CUDA devices.

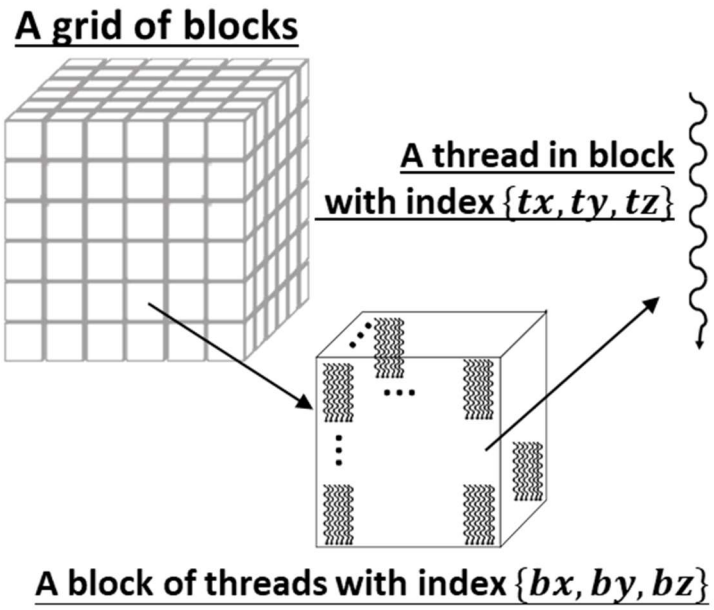

Figure 2.11 Hierarchical illustration of the CUDA programming model. Upon kernel execution, a grid of thread blocks which consist of numerous threads are instantiated. Therefore, the total number of threads launched is the product of number of blocks launched and number of threads in a block.

The CUDA programming framework is highly hierarchical. It divides the computation *kernels* into a grid of *thread blocks* that consist of numerous *threads* (**Figure 2.11**). Upon kernel execution, the launched thread blocks are assigned to be executed by one of the streaming multiprocessors, which utilizes its underlying array of CUDA cores to process the computation in parallel. All threads are able to access the GPU's *global memory*; communication and data exchange between the threads in a block can be achieved using the on-chip *shared memory* space. Further,

most GPUs also include read-only memories known as the c*onstant memory* and *texture memory*.

In the CUDA language, the GPU kernels are written as function and called with a triple chevron (<<<…>>>) style. The two arguments inside the triple chevron correspond to the number of thread blocks, and the number of threads per thread block being called. Thus, the total number of threads being called by the kernel is therefore (# of thread blocks) × (# of threads per block). Upon kernel execution, each thread can access its unique thread ID and can then access the unique data to perform the computation. The computation results from each thread can then be saved and fetched back to the host memory after computation. Example of such kernel and kernel call is given below:

Code snippet: CUDA Kernel Example

```
1  __global__ void gpuKernel(int *c, const int *a,. const int *b)
2  {
3      int globalThreadId =
                (blockIdx.x * blockDim.x)+ threadIdx.x;
4      c[globalThreadId] =
                a[globalThreadId] + b[globalThreadId];
5  }
```

Line 3 of *gpuKernel* exemplifies how the GPU threads query their unique global thread ID. Thus, the GPU threads can access a unique member of a, b and c as specified in line 4.

Code snippet: CUDA Kernel Call

```
1  unsigned int blocksPerGrid=2, threadsPerBlock = 1024;
2  gpuKernel <<<blocksPerGrid, threadsPerBlock>>> (c, a, b)
```

Line 1 of the kernel call example defines that 2 thread blocks containing 1024 threads will be called by *gpuKernel*. Therefore, a total of 2048 threads is called in the above example. Thus, the first 2048 members of *a* and *b* will be added and saved into the first 2048 members of *c* respectively.

It should be noted that all memory and data required for computation should be allocated and/or initialized to the GPU's memory prior to kernel execution. Such action can be done using the *cudaMalloc()*, *cudaMemset()* and *cudaMemcpy()* functions provided by CUDA API. This memory fetching process is slow due to limited bandwidth between the GPU and the host device (typically through the PCI-e slots). As such, it is more preferable to have all computation being completed in GPU before fetching them back to the host memory to reduce the overheads. Despite the internal memory bandwidth of GPU is much larger than the CPU-GPU bandwidth, it is still one of the factors limiting the performance of GPU. Particularly, the global memory one of the hot spots that can bottleneck the computation operations.

2.4.1.3 Hardware architecture of a CUDA GPU

A CUDA GPU is designed to process any computation in parallel under the *Single Instruction, Multiple Threads (SIMT)* architecture. The base computation units of a CUDA device are the streaming multiprocessors, which possess an array of processing elements (as known as "CUDA cores") for parallel computation. Multiple streaming multiprocessors are present in one GPU. Apart from the CUDA cores, each streaming multiprocessor also possesses their own instruction fetching unit, shared memory, as well as registers. Any memory transactions operations are accomplished by the multiprocessor's ability to access the graphics memory chip through a heavily cached memory bus.

Figure 2.12 illustrates a schematic diagram of a CUDA GPU. The GPU (device) is connected to the host PC via PCI-e channels, which allows communication between the GPU and the PC (host). The device usually consists of 2-8GB of graphics memory, of which the memory is physically located off-chip at close proximity to the streaming multiprocessors. The memory controller is responsible to load/store data to/from the streaming multiprocessor through the hardware-managed L2 and L1 caches. Both caches excel at fetching multiple data at once. However, one have to note that The L2 and L1 cache have different transaction characteristics. Memory transactions between the L2 cache and L1 cache are in batches of 32 bytes. Similarly, the transactions between the L1 cache and CUDA cores are in batches of 128 bytes. These cache behavior forms an

integral part in later optimization and discussions concerning about coalesced memory transaction.

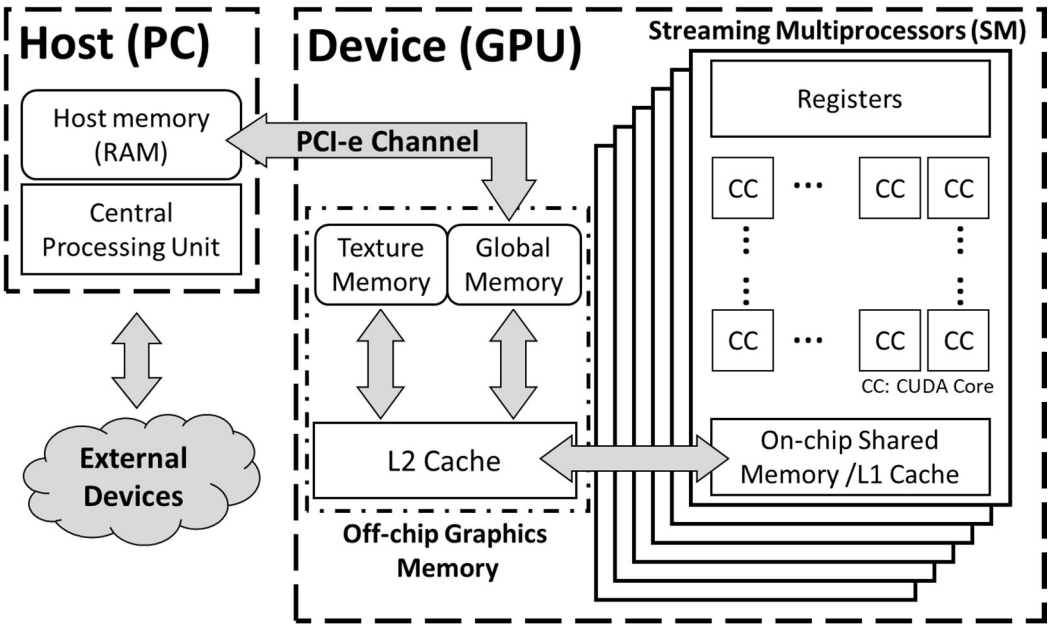

Figure 2.12 Simplified schematic diagram showing the hardware microarchitecture of a CUDA GPU. The GPU possesses numerous streaming multiprocessors as its basic computation units. The streaming multiprocessors can access the off-chip graphics memory via a heavily cached data bus.

On top of global memory, the GPU also possesses other types of specialized memory. For example, the on-chip shared memory is a fast, user-managed memory space that is capable to be utilized for effective data reuse which will be discussed in the later part of this thesis. This shared memory is much faster than the global memory in terms of both bandwidth and latency [84]. Besides, the texture memory that is also located off-chip is specialized to fetch data from its 2D or 3D locality which is not normally supported by the global memory. The texture memory forms another integral part in later optimization strategies concerning about interpolation. Finally, the constant memory of the GPU is responsible to pre-cache any needed data that is needed to be broadcasted to all streaming multiprocessors globally.

2.4.2 Horizon scan

Despite having much potential to speed-up the intensity-based image registration process, GPU was often overlooked by the field of image registration. However, GPU has been used in many related applications such as tomography reconstruction [85], Monte-Carlo simulation [86] and digital radiograph reconstruction [87]. Besides, the current research focus of non-rigid image registration mainly resides in the algorithmic development. There were many research groups focusing on developing a more advanced registration method to improve registration accuracy and robustness [88]. However, research on enhancing the computation process of image registration, particularly intensity-based image registration, has often been overlooked. To-date, there are a very few open-sourced implementations of GPU *demons*-based registration being available. The only available package is the GPU implementation of Thirion's *demons* registration included within the ITK package [89]. However, this implementation does not include any kind of GPU-based optimizations. Besides, the available deformation field update schemes provided for the GPU demons implementation in ITK [90] are very limited, with only passive force, active force and double force [57] currently being available.

Several research groups have attempted to implement the GPU version of Thirion's *demons* registration. Sharp et al. [91] first presented the GPU implementation of the native *demons* algorithm on a NVidia GPU using the Brook environment [92], yielding an 80× performance speedup compared to CPU. Courty et al. [93] transferred the native *demons* algorithm into GPU, and re-implemented the Gaussian smoothing step by reclusive mapping and filtering the 3D volume on a 2D texture [94]. Muyan-Ozcelik et al. [95, 96] implemented the same demon-based registration using CUDA, yielding additional 10% speedup compared to the work presented by Sharp et al. [91]. Gu et al. [97] later presented a quantitative comparison of 5 implementations of *Demons* variants on GPU, including passive force, evolved passive force, active force, double force and inverse consistent methods.

However, little work has been done on the accelerating any types of advanced *demons* variants. For example, the *diffeomorphic log-demons* [69] and

spectral log-*demons* [74] are considered to be much more robust than Thirion's native *demons* algorithm. The only reported work is by Huang et al. [98], who presented an implementation of 2D *diffeomorphic log-demons* using CUDA. However, the computation speed-up reported is also not as satisfactory as expected, possibly due to the fact that their implementation is not fully optimized.

2.5 Conclusion

In this chapter, an overview of the intra-op imaging technique has been reviewed, and the clinical demands of having image-guided interventions have also been discussed. I have also introduced the basic principle of image registration. Particularly, intensity-based image registration is a more reliable approach in the intra-op scenario. However, the immense computation demand of intensity-based image registration precludes registration to be applied in the clinical practice. GPU possesses the ability to parallelize the computation for fast registration. In this regard, the hardware microarchitecture of CUDA GPU, which is a mainstream general-purpose GPU, has been introduced. However, in a brief horizon scan, we found that GPUs are often overlooked in the field intensity-based image registration.

Aiming to achieve fast intensity-based image registration, the considerations in the implementation process of GPU-accelerated *diffeomorphic log-demons* will be presented in the following chapters. The computation demand of *diffeomorphic log-demons* will be analyzed in **Chapter 3**. The implementation details will be presented in **Chapter 4**. And the consideration of the limits of GPU-accelerated *diffeomorphic log-demons*, future directions, and potential impacts will be finally outlined in **Chapter 5**.

Chapter 3

ALGORITHMIC ANALYSIS AND PERFORMANCE-AWARE OPTIMIZATION

3.1 Introduction

As depicted in the previous section, the overwhelming amount of computation can result in long processing time for intensity-based non-rigid registration. Despite having satisfactory registration robustness, this prolonged running time is the major bottleneck of putting the algorithms into clinical use. The implementation of native *demons* algorithm (Thirion's *demons* algorithm) on the GPU has already been widely studied. However, there is a lack of GPU implementations for any advanced *demons* algorithm, such as the *diffeomorphic log-demons*. Implementing algorithms on a GPU requires a thorough understanding of the computation process involved. To achieve the best performance gain through the best use of GPU, identification of the limiting steps that bottleneck the computation is crucial. Particularly, as GPU speeds up the computation by parallelizing the workload on a large scale, the memory bandwidth requirement that comes with this largely parallelized computation will also need to be addressed.

Given there is a large computation demand for the intensity-based registration algorithm, performance-aware computation is the key technique to leverage the full power of a CUDA GPU for efficient computation. In this chapter, a brief highlight of different features on a GPU is first presented, followed by a survey of GPU memory access patterns. A complete algorithmic analysis of

diffeomorphic log-demons, aiming to pinpoint the computation bottleneck, will also be presented. Once the computation bottleneck is identified, the computation requirement, as well as the underlying memory access patterns involved in those operations will be investigated. In light of the computation and memory requirement, an optimal implementation for those bottlenecking will be devised using various performance-aware programming techniques.

3.2 GPU performance-aware programming for image registration

As presented in previous sub-sections, the GPU has an unmatched computation throughput for processing large computation workload in parallel. However, in order to harness the full computation power of a GPU, having *performance awareness* in the programmer's mind is the key. Performance-aware programming is a method that involves repeated optimization and profiling of the program. In order to achieve high-performance GPU non-rigid image registration, one has to effectively utilize the GPU's strength for high-performance computation. Under-utilization of the GPU device has to be avoided. In all, the CUDA GPU possess 3 features that are essential for high-performance computing, namely: *a)* highly-parallelized computation via SIMT, *b)* efficient caching, and *c)* rapid interpolation.

a) Parallelized SIMT computation

Highly parallelized computation is achieved by the GPU's streaming multiprocessors, which are designed to execute numerous threads in parallel. Under the SIMT architecture, the CUDA cores of each streaming multiprocessor execute warps of 32 threads simultaneously under a single instruction fetch-decode cycle. However, execution of the warps can be bottlenecked ("stalled") due to memory dependency, branching, or synchronization barriers. In this regard, the instruction scheduling unit on the streaming multiprocessor can mitigate this latency by executing multiple warps concurrently, similar to simultaneous multithreading (SMT) strategies on CPUs. Once a warp is stalled upon high latency operations, the scheduler can switch to another warp for continual execution. Such warp-switching strategies, combined with warp-level SIMT, provides streaming multiprocessors essential computation throughput to handle the enormous thread launched under a kernel.

b) Efficient Caching

Efficient caching can be achieved by the efficient usage of the on-chip shared memory on each streaming multiprocessor manner (**Figure 2.12**). As memory dependency is one of the most common reasons of bottlenecking, the much faster (>80x) shared memory can act as an efficient, user managed cache to mitigate any latency brought by redundant global memory accesses [84]. With the shared memory being used as a user-managed cache, data can be initialized on the shared memory in a highly efficient manner. For example, memory coalescence during the global-shared memory transaction can be enforced. Besides, as threads in the same block can access the shared memory, essential data can be exchanged between threads through such shared memory space. Any temporary results can be also stored and accessed with little overheads. Once the computation is completed, the results can be efficiently written back to the global memory in a coalesced manner.

c) Rapid Interpolation

The ability of rapid interpolation in GPU is brought by its dedicated texture hardware. Texture filtering is one of the most commonly used processes in graphics rendering, which is considered an expensive operation due to its high arithmetic and memory demand. To this end, the GPU possess specific hardware to perform the underlying fetching and arithmetic computation of interpolation in an optimized manner. By binding specific memory segment onto the texture memory/cache for read-only access, the GPU's texture hardware is able to automatically resolve the potentially complicated memory access patterns. Particularly, the hardware-managed texture cached is not only fast, but also provides unmatched spatial locality in 2/3D for efficient data fetching. The subsequent computation process is also optimized by the hardwired interpolation in GPU, which utilize fixed-point interpolants to speed up the potentially expensive multiplication process.

3.3 *Algorithmic breakdown*

Diffeomorphic log-demons is an algorithm that involves an iterative optimization process to find out the optimal transformation between two images. Such optimization is performed by minimizing the global harmonic energy. As introduced in equation (2) in page 26, such harmonic energy can be broken down the "similarity term" (Sim) and the "likelihood term" (Reg). Although the computation is already simplified by the *demons* framework which break down the workflow into a much simpler iterating routine, the required computation throughput is still very demanding. In this sub-section, I will first identify the related memory access patterns required in *diffeomorphic log-demons*. After that, the computation process required in *diffeomorphic log-demons* will be analyzed.

These code analyses are based on of the open-sourced MATLAB code[1] composed by Lombaert, the author of spectral-log *demons* [74] which is an improved version of the *diffeomorphic log-demons*.

3.3.1 GPU memory access patterns

A thorough understanding of memory access patterns in GPU is for implementing high-performance applications. Memory contention is one of the major factors that contribute to computation bottleneck. To this end, identifying of memory access patterns involved in the computation is crucial for tailoring specific strategies to resolve any bottleneck. In this sub-section, a brief survey will be presented to identify the key memory access patterns involved in *diffeomorphic log-demons*, including map, gather/scatter, reduction, and stencil access patterns. These memory access patterns can be identified in the operations when I start to break down the algorithm in the next sub-section.

[1] MATLAB code available on https://www.mathworks.com/matlabcentral/fileexchange/39194-diffeomorphic-log-*demons*-image-registration. Accessed 8 March 2017.

3.3.1.1 Map

The map operation is one of the simplest memory operations in GPU. The map access pattern depicts each thread to operate independently without any conflicts. Therefore, the map operation producing a one-to-one mapping on the input-output datasets. A common example of such map operation is to perform a constant offset to the value stored in an array. As there is no external dependency, the read/write access incurred should be made sequential if possible in order to take advantage of the coalesced memory access by the global memory. Given the threads are totally independent, there is also minimal communication between each running thread inside the GPU. A schematic of the map operation is shown in **Figure 3.1**.

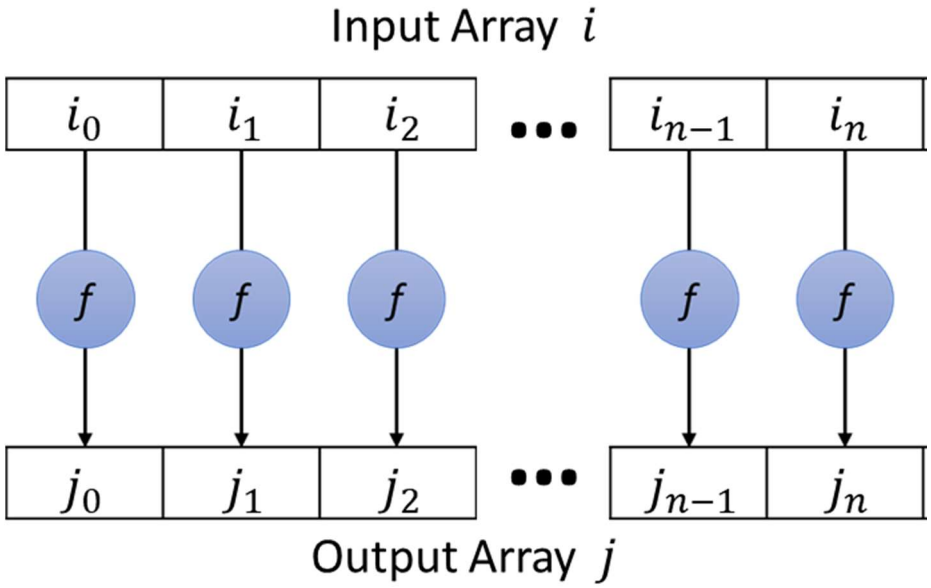

Figure 3.1 Schematic diagram showing map operation of a function f on an input array with a parallel processing architecture. Each computation is independent of each other.

Every thread launched by a kernel function involving a map access shall be independent in nature. Therefore, operations involving the map access pattern is inherently parallelizable. The hardware design of GPU is optimized for parallel processing of such independent computation. Thus, it is favorable for the GPU to enhance the computation for this kind of memory operation. As the read/write access of map operations are to be made sequential, they can also be cached in an orderly manner, thus, enabling full utilization of the memory bandwidth on the GPU.

3.3.1.2 Gather/Scatter

Gather and scatter are special forms of map operations [99]. Gather operations performs an indexed read from an array; scatter operations perform indexed write to an array. These operations do not require communication between the running threads within the same block, but the read/write access are indexed but may not be coalesced. **Figure 3.2** illustrates the memory access pattern of gather/scatter operations inside a GPU.

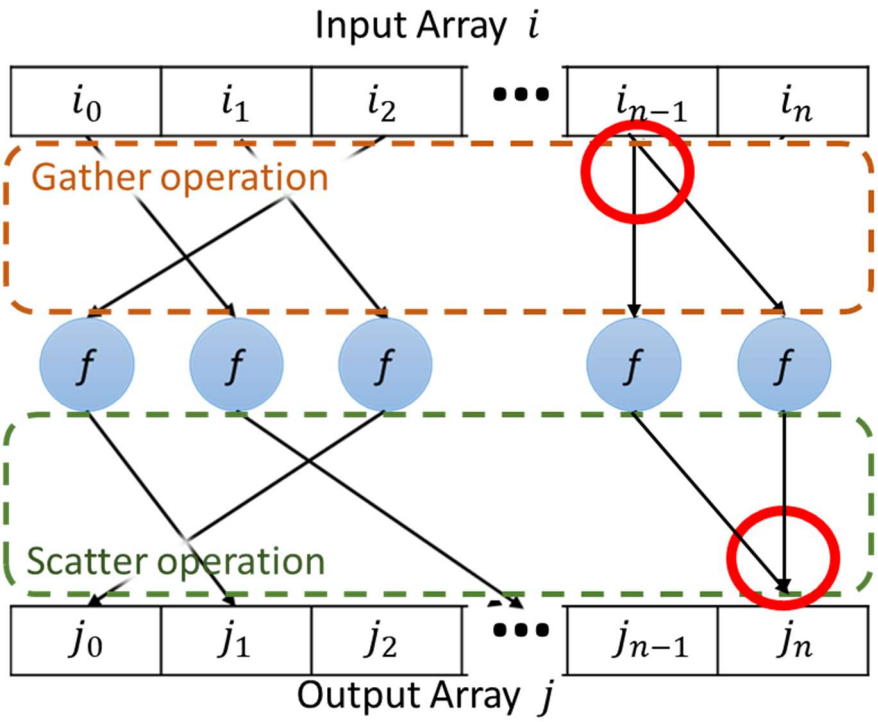

Figure 3.2 Schematic diagram showing gather-scatter operation of a function f with a parallel processing architecture. The threads are still independent, but the indexed reads/writes may induce conflicts among threads (red circles).

The memory access in gather/scatter operations is random in nature. In fact, a random memory access pattern is defined as a combination of gather and scatter operations. Such non-coalesced memory access patterns hinder effective memory caches. Data read/write conflicts may also occur when multiple threads attempt to access the same memory location. In many parallel architectures, the processor and memory controllers usually have specific hardware to handle such gather-scatter access patterns and resolve any potential read/write conflicts.

3.3.1.3 Reduction

The reduction operation combines all elements in the input array to generate a single output. As shown in **Figure 3.3**, the computation is performed in a tree-like pattern. Operations admissible to the parallel reduction are those which are binary, associative and commutation. For example, addition, multiplication, or Boolean operations such as AND, OR, and XOR operations. The order performed on the inputs are unimportant [100].

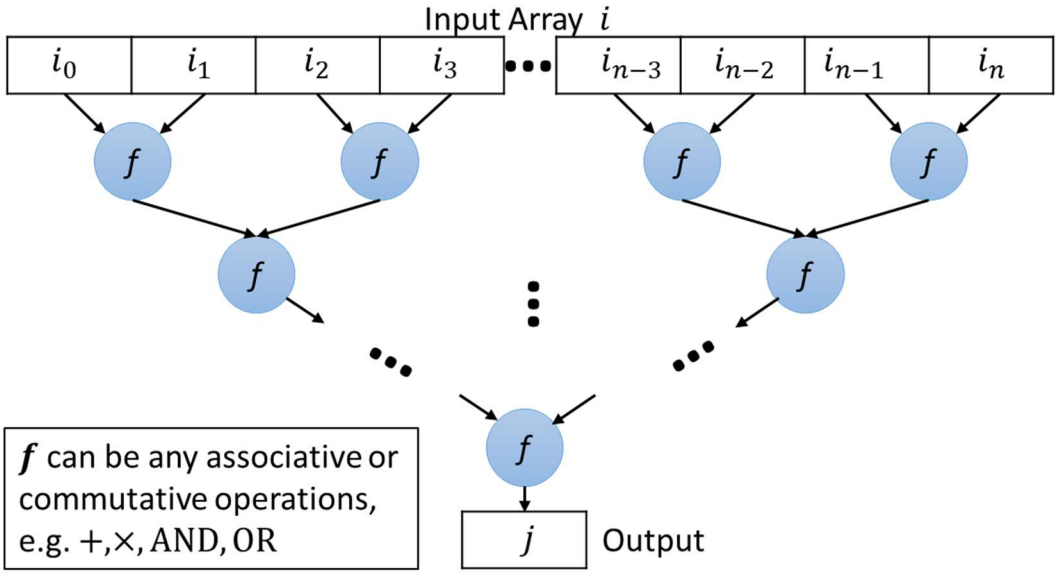

Figure 3.3 Schematic diagram showing reduction operation of a function f with a parallel processing architecture. To maximize parallelism, the dataflow is organized in a tree-like pattern.

Implementing a parallel computation to perform reduction possess a significant advantage over serial implementation. A reduction algorithm on a serial processor requires the big-O complexity of $O(n) = n$ to perform reduction of n elements. However, based on a tree-like parallel reduction pattern, theoretically a big-O complexity of $O(n) = \log_2 n$ can be achieved using an idealized parallel processor with infinite processing cores. However, as the number of processing core in the GPU is limited, the actual time required can be depicted by the Brent's Law [101] stated as follow:

$$T_p \leq T_N + \frac{T_1 - T_N}{p} \tag{7}$$

Where $T_{(\cdot)}$ depicts the time required for a parallel architecture with the specified number of processors in the $(\cdot)$ subscript to solve the problem; N is the maximum number of processors possible for solving the problem in parallel and p is the number of processor used in the practical situation.

3.3.1.4 Stencil

The stencil operation computes the output values using a set of fixed neighboring elements (called stencil) at the corresponding position of the input array. Typical stencil patterns are von Neumann pattern and Moore pattern. Theoretically, this operation is parallelizable due to having no inter-dependency between different threads in a kernel. However, the stencil access pattern is memory exhaustive when implemented in a parallel computing architecture, due to multiple read operations required for computation. As indicated in **Figure 3.4**, data members in the input array will have to be accessed multiple times throughout the computation process. Such recurrent access to the input array known as data redundancy must be resolved.

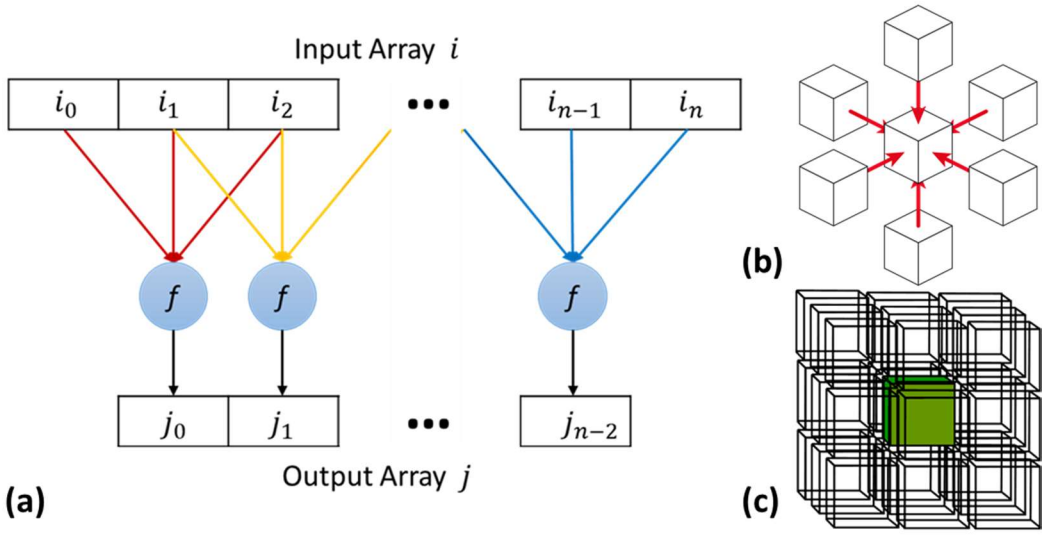

Figure 3.4 **(a)** Schematic diagram showing stencil operation of a function f with a parallel processing architecture. Overlapping of the stencil pattern induces redundant read operations. The stencil access pattern can either follow **(b)** 3D von Neumann stencil pattern, or **(c)** 3D Moore stencil pattern.

In addition to data redundancy, stencil operations in higher array dimensions will also induce non-coalesced memory access when reading over across rows and slices. As the memory is cached in a row-major manner, a cache miss is unavoidable when the memory pointer attempts to stride through the memory lab across columns or slices. Such memory access striding across column or slices of the memory slab is known as non-coalesced, and the cache miss induced by non-coalesced memory access can drastically bottleneck the computation.

3.3.2 Computation bottleneck of *diffeomorphic log-demons*

A MATLAB profiler is configured to monitor the execution time required for 100 *diffeomorphic log-demons* iterations to run on an image with a resolution of 0.2M voxels (64×64×64). Although the Java-based MATLAB codes are inherently not optimized for computation performance, the results obtained by the profiler can give reliable clue on the computation bottleneck that hinders the computation. The profiling results are abstracted and presented in **Table 3.1**. Also, the pseudocode of *diffeomorphic log-demons*, **Algorithm 2.2** first presented on page 23, is once again presented to facilitate the discussion in the subsequent sub-sections.

Table 3.1 Abstract results of the MATLAB profiler report after running 100 *diffeomorphic log-demons* iterations on a small (60×60×60) image set.

Function	Corresponding line in pseudocode	Number of times called	% of time consumed
Compute velocity field update	3	100	5.077%
Gaussian smoothing	4&6	600	40.434%
Field exponentiation	5	100	45.385%
Compute rergistration energy	7	100	6.145%

*Note that the total time consumed does not add up to 100% since this abstract report does not account for other minor operations and overheads.

Algorithm 2.2 Pseudocode showing the iterative registration process in the *demons*.

Pseudocode: *diffeomorphic log-demons* algorithm

1 **Input:** Fixed image $\boldsymbol{F}$ and moving image $\boldsymbol{M}$

2 In each iteration i **Do**:

3 Compute update field update $\boldsymbol{u}_i$ based on $\boldsymbol{F}$ and $\boldsymbol{M}_i$

4 Apply fluid-like regularization: $\boldsymbol{u}_i \leftarrow K_f \star \boldsymbol{u}_i$

5 Update velocity field: $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} \circ \boldsymbol{u}_i$

6 Apply diffusion-like regularization: $\boldsymbol{v}_i \leftarrow K_d \star \boldsymbol{v}_i$

7 Compute deformation field $\boldsymbol{s} \equiv \exp(\boldsymbol{v})$:

8 Update the moving image $\boldsymbol{M}_{i+1} = \boldsymbol{M} \circ \boldsymbol{s}_i$

9 Evaluate Harmonic energy $\boldsymbol{E}$

10 **Until** Harmonic energy (E) is minimized

11 **Output:** Deformation field s_i from $\boldsymbol{M}$ to $\boldsymbol{F}$

It is found that the major computation bottlenecks reside on the field exponentiation to retrieve the diffeomorphic deformation field, and the Gaussian regularization of the update/velocity field. Field exponentiation is the essential step to get the updated diffeomorphic deformation field from the velocity field. Hence, one field exponentiation computation is required per iteration. Furthermore, Gaussian smoothing is the common strategy for the regularization of the update field (line 4) and velocity field (line 6). As smoothing need to be done in all x, y, and z directions, the Gaussian smoothing function is called 6 times in each iteration, being called three times in each regularization step. In the remaining parts of this sub-section, I will breakdown the bottlenecking operations, namely Gaussian smoothing and field exponentiation, and propose appropriate optimization strategies for the best computation speed-up.

3.3.2.1 Gaussian smoothing

One computation bottleneck that hinders fast computation of *diffeomorphic log-demons* is the Gaussian smoothing step. Gaussian smoothing (also known as Gaussian blur) is the result of blurring an image/vector by a Gaussian function:

$$G(x,y,z) = \frac{1}{\left(\sqrt{2\pi\sigma^2}\right)^3}\exp\left(\frac{{x_d}^2 + {y_d}^2 + {z_d}^2}{2\sigma^2}\right) \quad (8)$$

Mathematically, such smoothing is equivalent to convolving the image/field with a Gaussian function. Applying a Gaussian blur to a field can reduce any high-frequency signal. Thus, discouraging sudden, sharp changes in the vector field.

Gaussian filtering is extensively used on vector fields as a simplified model of deformation propagation [7] and regularization [13]. The vector field is convoluted by a 3D array of pre-computed Gaussian values ("Gaussian kernel") that represents the Gaussian function. However, as the variance of the Gaussian function (σ) increases, this array can become large. A naïve implementation of 3D Gaussian filtering requires access to nearby $(6\sigma)^3$ elements for each voxel. However, such an operation involving numerous data transaction can impose heavy memory demand, especially when σ is large. To this end, multi-pass Gaussian filtering is commonly used to cut down the computation. Considering the 3D Gaussian kernel is symmetrical, this memory-demanding convolution process can be decomposed into multiple 1D convolutions. The formulation of a 1D Gaussian kernel ($k[d]$) is shown in Dquation (9) below. The workflow of performing multi-pass filtering by a symmetrical kernel on a vector field is presented in **Algorithm 3.1**.

$$k[d] = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{d^2}{\sigma^2}\right), d = \{0,1,\dots,\mathrm{nint}(3\sigma)\} \quad (9)$$

Algorithm 3.1 Pseudocode showing the computation procedure of multi-pass 3D Gaussian smoothing.

Pseudocode: 3D Gaussian smoothing

Input: vector field $\boldsymbol{u}$, kernel size $\boldsymbol{a}$, kernel values $\boldsymbol{k}[\boldsymbol{a}]$

// Perform x-pass convolution on $\boldsymbol{u}$ *as* $\boldsymbol{u}^x$*:*

For each vector $\boldsymbol{u}_i^x$ in $\boldsymbol{u}^x$:

    Perform discrete convolution on $\boldsymbol{u}$ along the x-direction:

    $u_i^x = \sum_{n=-a}^{a} \boldsymbol{k}[|n|] \times \boldsymbol{u}_{(x+n,y,z)}$

// Perform y-pass convolution on $\boldsymbol{u}^x$ *as* $\boldsymbol{u}^{xy}$*:*

For each vector $\boldsymbol{u}_i^{xy}$ in $\boldsymbol{u}^{xy}$:

    Perform discrete convolution on $\boldsymbol{u}^x$ along the y-direction:

    $\boldsymbol{u}_i^{xy} = \sum_{n=-a}^{a} \boldsymbol{k}[|n|] \times \boldsymbol{u}^x_{(x,y+n,z)}$

// Perform z-pass convolution on $\boldsymbol{u}^{xy}$ *as* $\boldsymbol{u}^{xyz}$*:*

For each vector $\boldsymbol{u}_i^{xyz}$ in $\boldsymbol{u}^{xyz}$:

    Perform discrete convolution on $\boldsymbol{u}$ along the z-direction:

    $\boldsymbol{u}_i^{xyz} = \sum_{n=-a}^{a} \boldsymbol{k}[|n|] \times \boldsymbol{u}^{xy}_{(x,y,z+n)}$

Output: Smoothed vector field $\boldsymbol{v} = \boldsymbol{u}^{xyz}$

In the context of *diffeomorphic log-demons*, Gaussian smoothing is used to both induce uncertainty to the feature correspondence [65], as well as to regulate the likelihood of the deformation field [55]. To perform Gaussian smoothing, every output pixel/voxel are required to read the intensity value of nearby pixel/voxels. This intense memory fetching is followed by multiplication of such intensity value with the corresponding Gaussian values. This memory access hence can be classified as a 3D stencil pattern as the processor needs to read and perform the weighted-sum computation of all elements near Moore's neighborhood.

3.3.2.2 Vector field exponentiation

Diffeomorphic log-demons adopts a "scaling and squaring" approach [66] which can effectively approximate the exponentiation of vector field. In the scaling and squaring approach, Newton's method is first used to evaluate the first step of the integration. The resultant integral is than self-composed several times to yield

the complete integration of the exponentiated vector field. In fact, the first step integration process is not computationally intensive, compared to the repeated composing operations. In each compositive operation, each element in the output vector is interpolated from the input vector field. As such, the nearest 8 vectors in 3D from the input vector field will have to be fetched before trilinear interpolation. Without a doubt, such interpolation will incur a massive memory bandwidth requirement due to extensive memory access. The access pattern will also appear to be random due to the non-parametric nature of the vector fields. The pseudo code for field exponential in a 3D vector field are provided:

Algorithm 3.2 Pseudocode showing the key computation procedure for fast approximation of vector field exponentials using the "*scaling and squaring*" method.

Pseudocode: Fast Vector Field Exponentials

Input: Velocity vector field $\boldsymbol{v}$

Choose N such that $2^{-N}\boldsymbol{v}$ is close to 0 (e.g. $2^{-N}\boldsymbol{v} \leq 0.5$)

Perform explicit first-order integration $\Phi \leftarrow 2^{-N}\boldsymbol{v}$

Repeat *N* times:

    Recursive scaling and squaring: $\Phi \leftarrow \Phi \circ \Phi$

Output: Diffeomorphic map $\Phi = \exp(\boldsymbol{v})$

As illustrated **Algorithm 3.2**, the vector field exponentiation algorithm requires N compose operations to compute the diffeomorphic mapping $\boldsymbol{\phi}$ from a velocity vector field $\boldsymbol{v}$. The number of compose operations depends on the magnitude of the velocity field. As the step of integration for needs to be small enough to achieve reasonable accuracy by the Newton's method, the velocity vector needed to be first scaled down by a factor of 2^{-N}. The ultimate value of N is to be determined by the magnitude of the largest velocity vector inside $\boldsymbol{v}$. Thus, more field composition iterations are required to estimate the field exponentials with a larger $\boldsymbol{v}$. The typical value of N resides around 3-5. In the context of the *diffeomorphic log-demons* algorithm, the computation demand will increase exponentially as the deformations acuminate over the iterations.

3.3.2.3 Vector field composition

Vector field composition is the most used operation in *diffeomorphic log-demons* (denoted by the operator "∘" in the pseudocode). It operates on Lie group structure and essentially shares the same concept of addition in linear algebra. Interpolation is extensively used in evaluating vector field composition. Aside from being extensively used in vector field exponential operations, the compositive operation is also used to warp the deformation on the moving image, as well as updating the velocity field using the update field. The pseudo code for vector field composition and interpolation is presented:

Algorithm 3.3 Pseudocode showing the key computation procedure for vector field composition, which is one of the essential computations inside the *diffeomorphic log-demons* algorithm.

Pseudocode: Vector Field Composition (Operator "∘")

Input: velocity vector field $\boldsymbol{v}$, update vector field $\boldsymbol{u}$

For each vector $\boldsymbol{v}_i$ in $\boldsymbol{v}$:

Add the coordinates of $\boldsymbol{v_i}$ onto the components of $\boldsymbol{v}_i$ respectively as $\boldsymbol{v}'_i$: $\boldsymbol{v}'_i = \boldsymbol{v}_i + \text{coord}(\boldsymbol{v}_i)$

Treat each component of $\boldsymbol{v}'$ as a single image and wrap with update field $\boldsymbol{u}$ *(See Algorithm 3)*

$\boldsymbol{v}'_{px} = \boldsymbol{v}'_z \circ \boldsymbol{u}$ $\quad$ $\boldsymbol{v}'_{py} = \boldsymbol{v}'_z \circ \boldsymbol{u}$ $\quad$ $\boldsymbol{v}'_{pz} = \boldsymbol{v}'_z \circ \boldsymbol{u}$

For each vector $\boldsymbol{v}'_{pi}$ in $\boldsymbol{v}'_p$:

Subtract the coordinates of $\boldsymbol{v_i}$ onto the components of $\boldsymbol{v}'_{pi}$respectively as $\boldsymbol{v}_{pi}$: $\boldsymbol{v}_{pi} = \boldsymbol{v}'_{pi} + \text{coord}(\boldsymbol{v}'_{pi})$

Output: composed vector field $\boldsymbol{v_p} = \boldsymbol{u} \circ \boldsymbol{v}$

The composition of two fields $\boldsymbol{u}$ and $\boldsymbol{v}$ combines two smooth, continuous manifolds under Lie Algebra. However, the data stored in the computer memory are discrete in nature. Thus, interpolation is required to find out the vector value at an arbitrary point inside the field. As illustrated in **Algorithm 3.3**, field composition first resolve for the corresponding query point depicted by $\boldsymbol{u}$ on the vector field $\boldsymbol{v}$. After that, interpolation is performed at the query point depicted. This procedure is repeated over every discrete vector that represent the field $\boldsymbol{v}$.

The computation of field composition requires numerous of interpolation. Due to the non-parametric nature of the vector field, the number of interpolation required is equal to the resolution of the vector field. Different interpolation approaches can be used. For example, nearest neighbor interpolation is the faster, but it is the most inaccurate interpolation approach. Trilinear interpolation assumes a linear correlation between the interpolants. Tri-cubic or b-spline approaches tend to be more accurate, but require more computation. Trilinear interpolation (**Figure 3.5**) is well-known to be an acceptable trade-off between computation efficiency and accuracy. The pseudocode for trilinear interpolation is presented below in **Algorithm 3.4**:

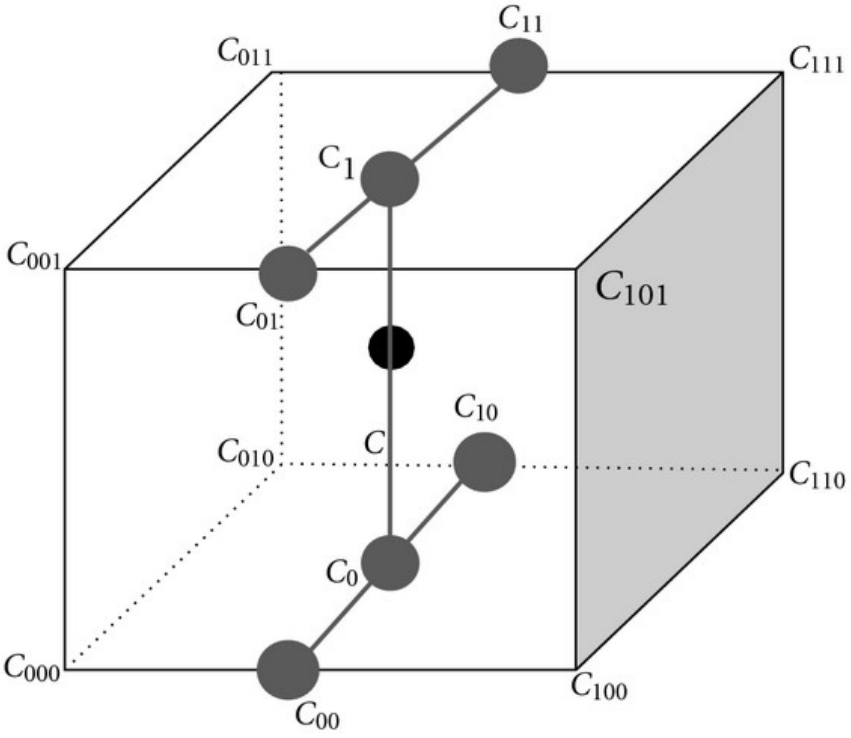

Figure 3.5 Illustration of trilinear interpolation. The value of point C is interpolated by the eight closest interpolants ($C_{000} - C_{111}$). Images retrieved from [102].

Algorithm 3.4 Pseudocode showing the key computation procedure for trilinear interpolation, which is performed numerous times in a single compositive operation.

Pseudocode: Trilinear interpolation (interp3)

1 **Input**: vector field $\boldsymbol{u}$, query point $\boldsymbol{q}$

2 Compute x_d, y_d, z_d which are the differences between query point $\boldsymbol{q}$ and the closest grid point with the least coordinate

$$x_d = \boldsymbol{q}_x - floor(\boldsymbol{q}_x) \qquad y_d = \boldsymbol{q}_y - floor(\boldsymbol{q}_y) \qquad z_d = \boldsymbol{q}_z - floor(\boldsymbol{q}_z)$$

3 Get 8 closest vectors $\{\boldsymbol{w}_{000} \dots \boldsymbol{w}_{111}\}$ from $\boldsymbol{q}$ in field $\boldsymbol{u}$ for interpolation:

$$\boldsymbol{w}_{000} = \boldsymbol{u}\begin{Bmatrix} floor(\boldsymbol{q}_x) \\ floor(\boldsymbol{q}_y) \\ floor(\boldsymbol{q}_z) \end{Bmatrix} \quad \boldsymbol{w}_{001} = \boldsymbol{u}\begin{Bmatrix} ceil(\boldsymbol{q}_x) \\ floor(\boldsymbol{q}_y) \\ floor(\boldsymbol{q}_z) \end{Bmatrix} \quad \boldsymbol{w}_{010} = \boldsymbol{u}\begin{Bmatrix} floor(\boldsymbol{q}_x) \\ ceil(\boldsymbol{q}_y) \\ floor(\boldsymbol{q}_z) \end{Bmatrix}$$

$$\boldsymbol{w}_{011} = \boldsymbol{u}\begin{Bmatrix} ceil(\boldsymbol{q}_x) \\ ceil(\boldsymbol{q}_y) \\ floor(\boldsymbol{q}_z) \end{Bmatrix} \quad \boldsymbol{w}_{100} = \boldsymbol{u}\begin{Bmatrix} floor(\boldsymbol{q}_x) \\ floor(\boldsymbol{q}_y) \\ ceil(\boldsymbol{q}_z) \end{Bmatrix} \quad \boldsymbol{w}_{101} = \boldsymbol{u}\begin{Bmatrix} ceil(\boldsymbol{q}_x) \\ floor(\boldsymbol{q}_y) \\ ceil(\boldsymbol{q}_z) \end{Bmatrix}$$

$$\boldsymbol{w}_{110} = \boldsymbol{u}\begin{Bmatrix} floor(\boldsymbol{q}_x) \\ ceil(\boldsymbol{q}_y) \\ ceil(\boldsymbol{q}_z) \end{Bmatrix} \quad \boldsymbol{w}_{111} = \boldsymbol{u}\begin{Bmatrix} ceil(\boldsymbol{q}_x) \\ ceil(\boldsymbol{q}_y) \\ ceil(\boldsymbol{q}_z) \end{Bmatrix}$$

4 Interpolate $\{\boldsymbol{w}_{000} \dots \boldsymbol{w}_{111}\}$ along x direction to get $\{\boldsymbol{w}_{00} \dots \boldsymbol{w}_{11}\}$

$$\boldsymbol{w}_{00} = \boldsymbol{w}_{000}(1 - x_d) + \boldsymbol{w}_{001}x_d \qquad \boldsymbol{w}_{01} = \boldsymbol{w}_{010}(1 - x_d) + \boldsymbol{w}_{011}x_d$$

$$\boldsymbol{w}_{10} = \boldsymbol{w}_{100}(1 - x_d) + \boldsymbol{w}_{101}x_d \qquad \boldsymbol{w}_{11} = \boldsymbol{w}_{110}(1 - x_d) + \boldsymbol{w}_{111}x_d$$

5 Interpolate $\{\boldsymbol{w}_{00} \dots \boldsymbol{w}_{11}\}$ along y direction to get $\{\boldsymbol{w}_0, \boldsymbol{w}_1\}$

$$\boldsymbol{w}_0 = \boldsymbol{w}_{00}(1 - y_d) + \boldsymbol{w}_{01}y_d \qquad \boldsymbol{w}_1 = \boldsymbol{w}_{10}(1 - y_d) + \boldsymbol{w}_{11}y_d$$

6 Interpolate $\{\boldsymbol{w}_0, \boldsymbol{w}_1\}$along z direction to get $\boldsymbol{w}$

$$\boldsymbol{w} = \boldsymbol{w}_0(1 - z_d) + \boldsymbol{w}_1 z_d$$

7 **Output**: Interpolated vector $\boldsymbol{w}$ at point $\boldsymbol{q}$ in the vector field $\boldsymbol{u}$

In trilinear interpolation, the direction and magnitude the interpolated vector at the query point $\boldsymbol{q}$ is estimated by the 8 closest vectors. Once the data is loaded into the memory, linear interpolation is performed in a step-by-step manner that covers all x, y, and z directions. However, as such interpolation involves 3-dimensional interpolation of a 3-dimensional vector field, this composition operation will have to be repeated 3 times for 3 separate vector components, each demanding the processor to loop through the array for pixel/voxel-wise interpolation. Such heavily looped and repeated workflow can not only demand

substantial computation throughput, but also require considerable memory bandwidth between the processing units.

3.3.2.4 Velocity field update

Finally, d*iffeomorphic log-demons* continuously updates the velocity field until the resultant deformation field converges onto an optimal solution. The updates are governed not only by the pixel/voxel intensity difference, but also the intensity gradients between the fixed image and the immediate moving image.

Algorithm 3.5 Pseudocode showing the key computation procedure for vector field update, which contains a considerable amount of branching and arithmetic computations.

Pseudocode: Compute velocity field update

Input: Fixed image $\boldsymbol{F}$, Immediate moving image $\boldsymbol{M}$,
Registration Parameter $\boldsymbol{\alpha}$

Compute gradient of $\boldsymbol{F}$ and $\boldsymbol{M}$ as $\nabla\mathbf{F}$ and $\nabla\mathbf{M}$

For each voxel $\boldsymbol{F}_i$ and $\boldsymbol{M}_i$ in $\boldsymbol{F}$ and $\boldsymbol{M}$ do

Compute voxel-wise difference of $\boldsymbol{F}_i$ and $\boldsymbol{M}_i$ as $\boldsymbol{D}_i$:
$\boldsymbol{D}_i \leftarrow \boldsymbol{F}_i - \boldsymbol{M}_i$

Compute the Hadamard product of $\nabla\mathbf{F}$ and $\nabla\mathbf{M}$ as $\boldsymbol{Q}$:
$\mathbf{Q_i} \leftarrow \nabla\boldsymbol{F}_i \times \nabla\boldsymbol{M}_i$

Compute the magnitude of $\nabla\mathbf{M}$ **as** $\mathrm{norm}(\boldsymbol{\nabla M})$:
$\mathrm{norm}(\boldsymbol{\nabla M}_i) \leftarrow \sqrt{\nabla\mathbf{M}_{\mathbf{ix}}^{\mathbf{2}} + \nabla\mathbf{M}_{iy}^{2} + \nabla\mathbf{M}_{iz}^{\mathbf{2}}}$

Compute update magnitude $\boldsymbol{p}$: $\boldsymbol{p}_i \leftarrow \frac{\boldsymbol{D}_i}{\mathrm{norm}(\boldsymbol{\nabla M})^2 + \alpha^2 \times D_i^2}$

If $\mathrm{norm}(\boldsymbol{\nabla M}) = 0$ **and** $\boldsymbol{D}_i = 0$ **do**:

Handle extremities: $\boldsymbol{p}_i \leftarrow 0$

If Hadamard product $\boldsymbol{Q}_i < 0$ **do**:

Invert the direction of update: $\boldsymbol{p}_i \leftarrow -\boldsymbol{p}_i$

Compute velocity field update $\boldsymbol{u}$: $\boldsymbol{u}_i \leftarrow \nabla\mathbf{M} \times \boldsymbol{p}_i$

Output: vector field update $\boldsymbol{u}$

Algorithm 3.5 presents a workflow of a commonly used field update method. As shown in the workflow, the computation for the velocity field update can be complicated. Gradient decomposition in 3D requiring access to the intensity values of direct adjacent pixel/voxels, which will incur strided memory access along the *y*-/*z*-directions. Generation of the update field also involves several branching operations that can be expensive when performed by GPU. Besides, computing the updated velocity field involves the complicated vector field composition as described in *Section 3.3.2.3* again. As a result, despite the velocity field update operations does not require much computation power nor computation time, there is ample amount of optimization work that can be done to improve the computation efficiency.

3.4 Computation optimization

With the computation bottleneck of *diffeomorphic log-demons* being presented in the previous sections, this section provides an in-depth analysis of the arithmetic and memory demand of the bottlenecking operations. With such demand being thoroughly understood, one can resolve the computation demand by utilizing different abilities of the GPU to achieve fast computation.

3.4.1 Gaussian smoothing

Section 3.3.2.1 has already shown that the computation of Gaussian smoothing is inherently parallelizable, which can be enhanced by the ability of parallel processing by GPU. However, the smoothing operation requiring stencil memory access pattern (*Section 3.3.1.4*) demands data fetching from neighboring elements. This fetching procedure can be memory intensive, and it may take up substantial bandwidth. As a result, even with a fully-parallelized implementation of Gaussian smoothing on the GPU, this computation process can still be memory bound, due to simultaneous access to the memory by numerous threads. Due to overlapping of adjacent convolution kernels, the data are often recurrently fetched and accessed by a number of threads which can lead to extensive memory bottleneck. Therefore, there is a need for the processor to share the data among different threads within a thread block for more efficient computation.

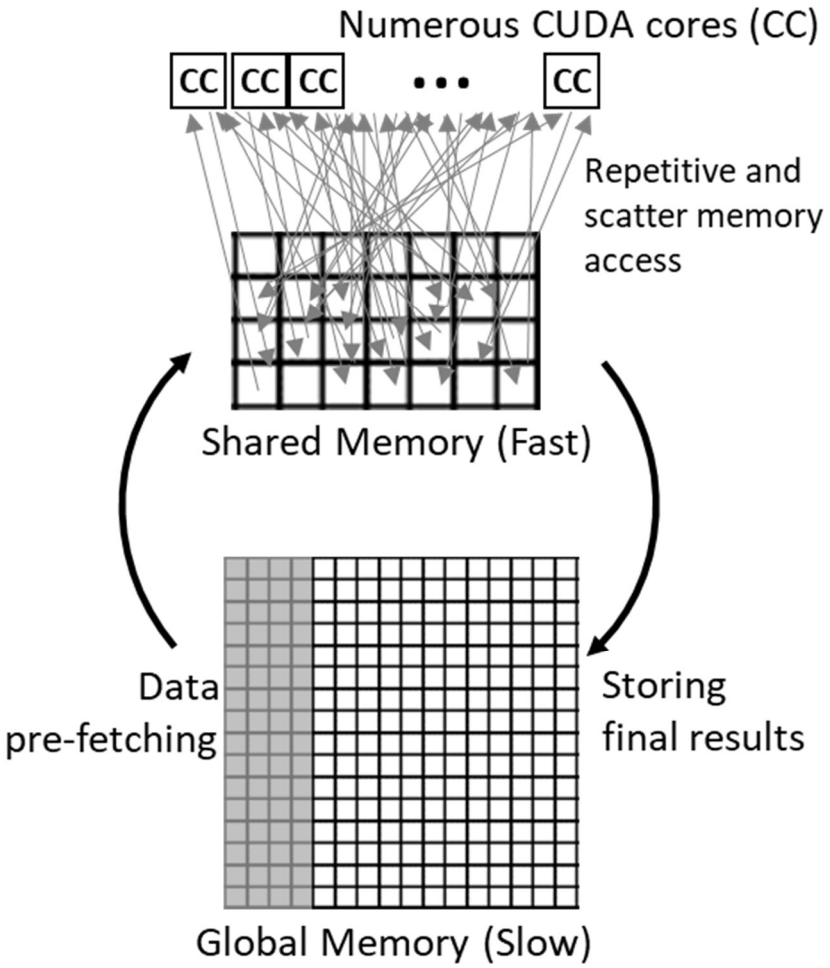

Figure 3.6 On-chip high-bandwidth shared memory used as a user-managed cache. To avoid unnecessary global memory transection, the required data are pre-fetched onto the shared memory. Such data can be swiftly accessed and reused by numerous CUDA cores. The final results are stored back to the global memory after the computation.

In this regard, the GPU's on-chip shared memory can allow data to be pre-fetched onto the shared memory for data reuse. The high-bandwidth data channel between the processing elements and the shared memory allows data to be load/stored at a substantially faster manner. This ability of fast load/store of memory among all threads in a thread block facilitates data reuse. **Figure 3.6** outlines the basic idea of using the high-bandwidth shared memory as a user-

managed cache. As the result of such caching, data can be read from the shared memory without any substantial overhead or memory contention. Any intermediate results can also be stored for subsequent fast access by any other threads.

Given the intuitive algorithm of multi-pass Gaussian smoothing, a natural way to perform the computation is to instantiate thread blocks with their size equal to the *x*-/*y*-/*z*- dimensions of the image. With such block dimension, the neighboring row/column/slice data can be loaded onto the shared memory at once for fast convolution. However, such arrangement in thread blocks can set back memory transaction efficiency when the global memory is accessed in a strided pattern. In particular, during the *y*- and *z*-pass of the convolution. As the global memory is optimized to perform 128-bytes coalesced memory transaction through the L1 and L2 cache, much of the memory bandwidth will be under-utilized if there is only a fraction of data being used by the threads.

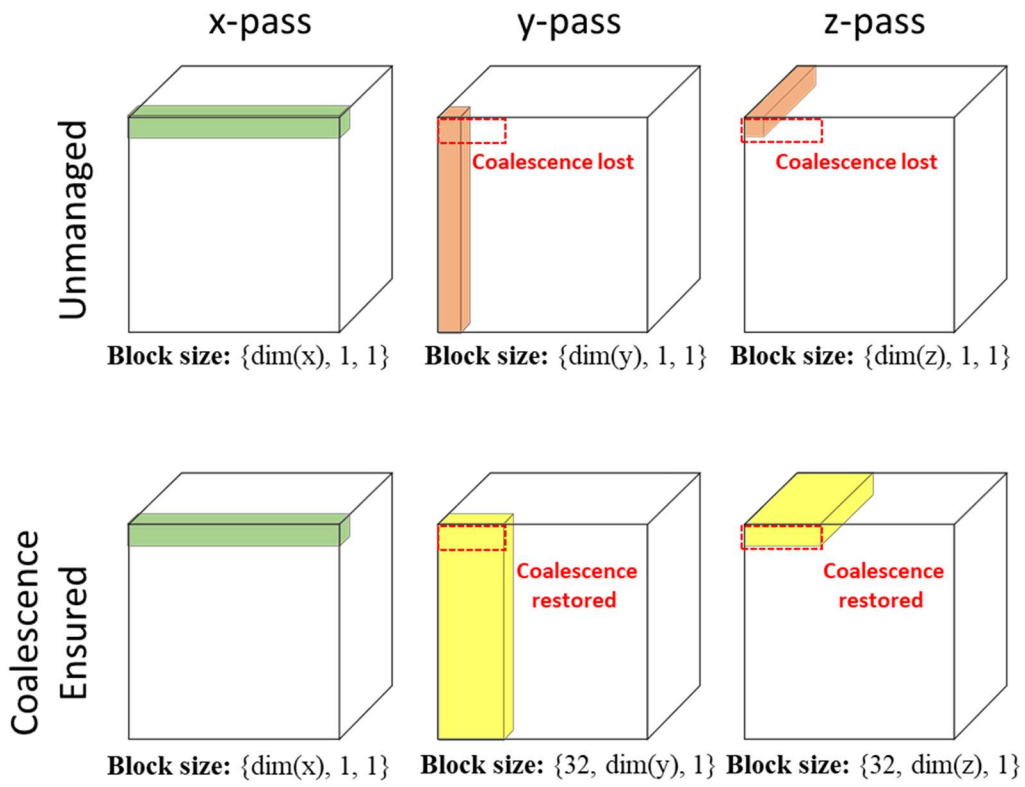

Figure 3.7 Thread block management and memory coalescence in GPU multi-pass convolution. In the unmanaged arrangement, the slender thread blocks along the *y*- and *z*-directions breaks *x*-direction memory coalescences (*red blocks*). Careful management of the thread block dimensions ensures memory coalescence for efficient memory transaction (*yellow blocks*).

To this end, one can enforce coalesced memory access to maximize the efficiency of memory transactions between the global and shared memory space. This can be achieved by instantiating the thread block with appropriate *x*-dimension in the *y*- or *z*- pass of the implementation (**Figure 3.7**). For example, if the convolution is performed on an array of single-precision elements (i.e. 4 bytes), it is often a good choice to instantiate thread blocks with x-dimension of 32. As such, simultaneous fetching of 32 single-precision float elements from a warp can fulfill 128-byte L1 memory coalescence. Full utilization of the memory bandwidth can therefore be achieved. However, as the CUDA architecture to-date supports a maximum of 1024 threads being launched per thread block, one may encounter the problem that the number of threads required exceeds the maximum allowable number of threads supported by CUDA. This can be resolved by launching extra thread blocks along the *y*- or *z*-direction, but the overlapping convolution areas between two adjacent thread blocks requires redundant memory transactions, which hampers efficiency. Launching extra thread blocks can also incur additional kernel overheads.

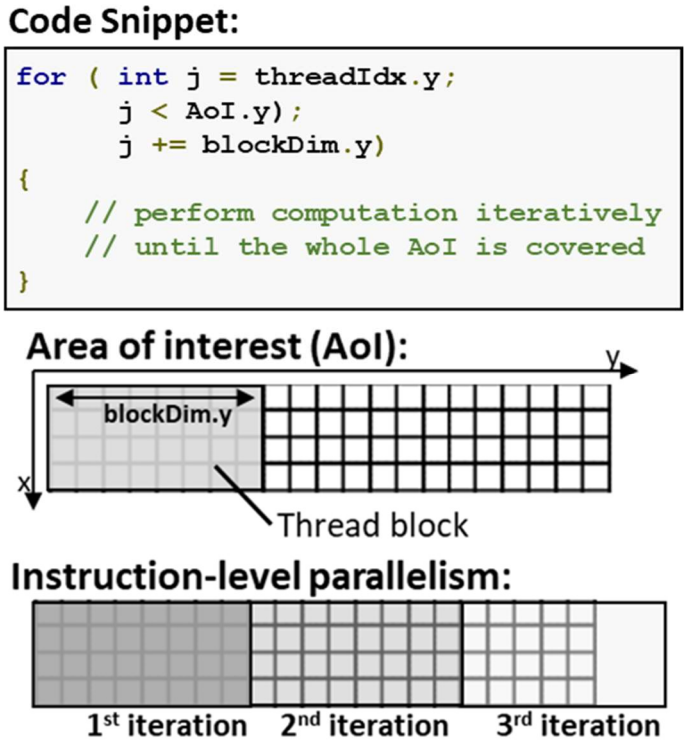

Figure 3.8 Code snippet showing instruction-level parallelism for efficient thread reuse. Computation for multiple voxels can be conducted by a single thread without the need of launching additional thread blocks.

Instead of launching and initializing extra thread blocks for the computation, instruction-level parallelism can be employed to allow the threads to handle multiple pixel/voxels (**Figure 3.8**). By parallelizing the computation at the instruction-level, one can avoid extra latency brought by launching any additional thread blocks. Furthermore, as different components of the 3D vector field are temporally coherent (i.e. they have to be fetched altogether), it is better to store such coherent data in an array-of-structure (AoS) format instead of the conventionally used structure-of-array (SoA) format. It is because fetching different components stored in SoA require the memory controller to transverse across different memory locations. Such memory access to discrete regions of the memory can upset standard memory caching. Thus, incurring additional memory latency that precludes fast computation. Also, because of the discrete memory address of the data, multiple instructions will also have to be issued by the compiler to load all required variables. As the result, the read/write efficiency will be further hampered due to fetch/decoding overheads of these extra instruction cycles.

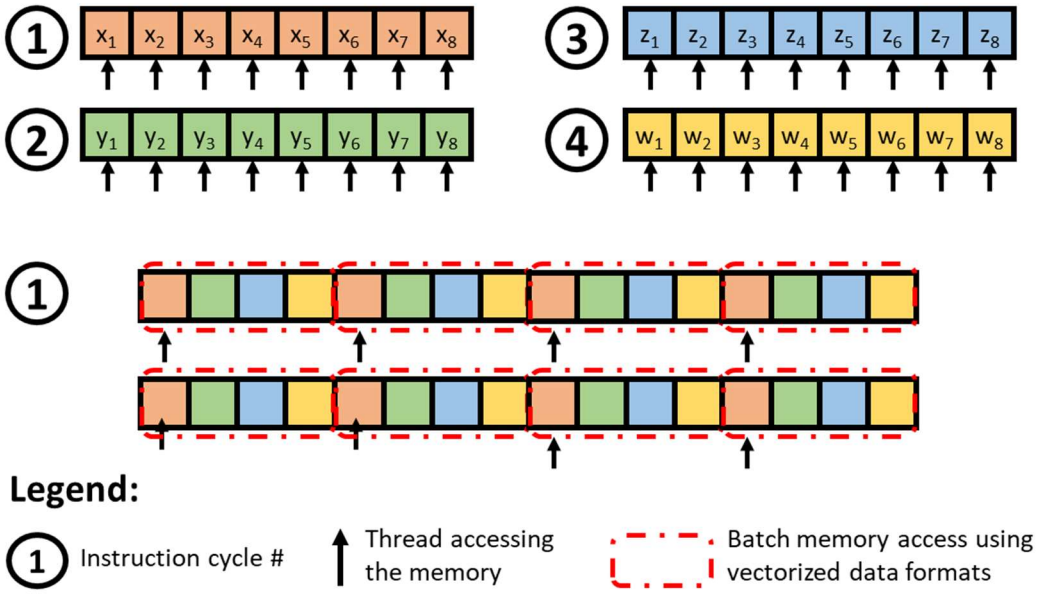

Figure 3.9 Multiple instruction cycles required to access data stored in SoA *(top)*; in contrast, accessing data stored in AoS only require a single instruction cycle *(bottom)*.

By reorganizing the 3-dimensional data into an array-of-structure formant, different components can be stored in a coalesced structure which is capable be fetched under a single instruction (**Figure 3.9**). In this regard, the vectorized data

type *float4* consisting of 4 coalesced 32-bit floating point variables is ideal to accommodate the magnitude and individual component values of a vector. In the PTX instruction set it is observable that the compiler will automatically optimize the fetching instruction from multiple *ld.global.f32* instructions to a single *ld.global.f32.v4* instruction, which can be considerably faster.

As an added plus for storing data in AoS format, the required number of threads in the *x*-direction to achieve the 128-byte memory coalescence at L1 can also be reduced (**Table 3.2**). Moreover, as the memory controller is able to fetch the all elements in a vectorized data using a single instruction, the overall transaction request can be reduced to alleviate data contention on global memory.

Table 3.2 The required number of threads in *x*-direction to enforce 128 Byte global memory transaction coalescence for various float data types in AoS format.

Data type	Size of the data type (bytes)	Dimension of thread block in *x*-direction required to achieve 128-byte memory coalescence
float	4	32
float2	8	16
float3	12	(unable to achieve 128B coalescence)
float4	16	8

In all, to achieve optimal computation efficiency for Gaussian smoothing, one has to effectively utilize the GPU's shared memory as a user managed cache for data reuse. In that regard, the dimensions of the thread blocks launched for parallel computation shall be carefully managed. Besides, vectorizing the temporally coherent data into AoS format can also assist global memory bandwidth utilization. As a result of these memory optimization strategies, contention on the memory controller can be alleviated. By effectively utilizing the memory bandwidth, it is possible to perform Gaussian smoothing in a swift manner using the GPU.

3.4.2 Vector field composition

Regarding the large memory and computation throughput in image warping and vector field composition, the GPU can utilize its ability of parallel processing

to provide the computation throughput required. As depicted in *Section 3.3.2.3*, the composition process is entirely pixel-independent, of which it is capable to be parallel processed without any data racing conditions. However, even with the subroutine being able to be parallel processed, a large number of gather operations (*Section 3.3.1.2*) during the interpolation of two non-parametric vector fields can incur substantial latency. Moreover, as linear interpolation requires the nearest 8 nearest vectors as the input, this gather operation will also involve a stencil pattern in 3D, thus incurring memory strides along the *y*- and *z*- directions. Such stencil access from random locations can again result in non-sequential memory access on the GPU memory.

As this gathering operation involves random memory address, it is impossible to restore memory coalescence solely by resizing the thread block dimensions. In addition, the intensive arithmetic computation involved in field interpolation can further hamper computation efficiency. To this end, the GPU's texture hardware pipeline can be effectively used to efficiently handle both the memory and the arithmetic demand of the operation. To address the non-sequential access on memory, GPU's texture cache on the streaming multiprocessors are optimized for fetching memory in 3D, which can achieve fast requisition of neighboring data values. Therefore, fast fetching from spatially-coalesced elements for interpolation can be achieved. By binding specific regions on the global memory to define a texture object, it is possible for the texture API to automatically manage the complicated memory transactions involved. Once the texture fetching instruction is called, the hardware can automatically load all essential data and perform fast texture filtering.

The texture API requires strict memory address alignment. Contrast to the global memory of which the memory address alignment can be resolved by L2 caching, L2 coalescing is not supported by the texture memory. Instead, the texture memory requires data in each row to be fetched are properly aligned on the 128-byte L2 cache lines. However, as the vectorized 3-dimensional arrays stored in a row-major manner can be of any size, one cannot guarantee the address are properly aligned for every set of the input image in the linear memory (**Figure 3.10**). Such misalignment can be tackled by padding (append) the memory at the end of each

row. However, manually padding the memory to enforce alignment will result in a non-sequential address index, which will be confusing and counterinitiative to most programmers.

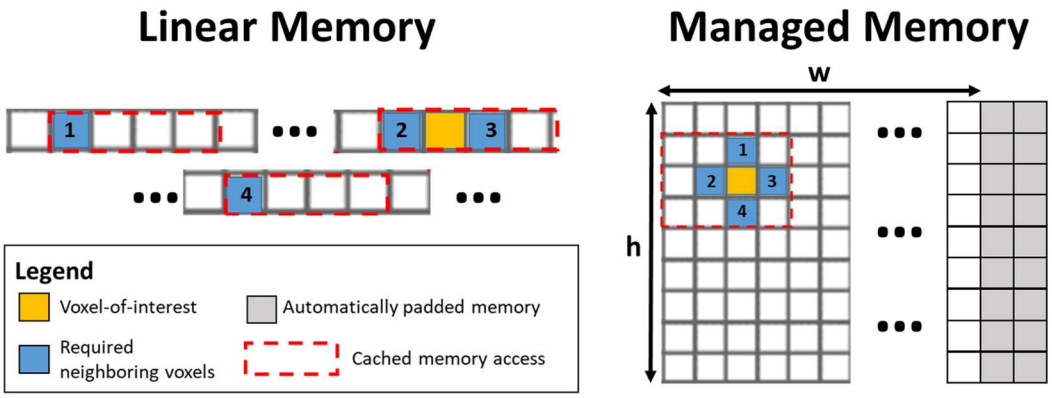

Figure 3.10 Strided access to linear memory *versus* managed memory. Multiple cached memory access may be required for reading the misaligned data stored in linear memory *(left)*. In contrast, spatially localized elements stored in managed memory can be efficiently cached and accessed with the automatically aligned memory address *(right)*.

To this end, GPU possesses the ability to manage the 3-dimensional memory arrays using special sets of "pitched" memory pointers for fast data fetching through an automatically managed memory array. Through the pitched memory pointer, the device to automatically allocate and enforce memory alignment for fast access of 3D spatially localized memory elements. Such fast access to 2/3D spatially localized memory by the graphics pipeline cannot be supported by the ordinary linear array. Once the texture data are fetched to the graphics pipeline, the GPU's texture mapping unit is able to swiftly perform the interpolation (known as "texture filtering") through its hardwired computation units. However, the accelerated interpolation does come at the cost of reducing interpolant precision (9-bit fixed point), which will still be sufficient in most scenarios in intensity-based image registration due to the limited data range (usually 12-bit unsigned integers) provided by most medical images.

3.4.3 Velocity field update

Section 3.3.2.4 has presented the major computation resides in 3D gradient decomposition of the fixed and moving image, as well as the composition of velocity update field. Similar to Gaussian smoothing, a stencil pattern accessing the von Neumann neighborhood is required in 3D gradient decomposition. Likewise, the 3D vector field composition required to update the velocity field is identical to the operations presented in *Section 3.3.2.3.* Besides, the heavily branched computation required in generating the update field can also impose stall the execution of warps on a GPU.

As the stencil memory access pattern involved in gradient decomposition similar to Gaussian smoothing. The operation can be optimized using similar data reusing techniques which are presented in *Section 3.4.1*. Similarly, the immense computation load involved in vector field composition can also be resolved using the GPU's texture hardware as presented in *Section 3.4.2.* Additionally, any computation bottleneck brought by instruction branching can be resolved by code optimization. Such optimization can be achieved by decomposing a complex branch into a simpler control flow operation, which allows the processor to access the relevant results that are computed preemptively. As a result of the decomposed flow operation, the compiler can automatically optimize for the thread divergence to completely mitigate any overhead.

3.5 Conclusion

In this chapter, I have introduced different performance-aware programming methods on GPU in *Chapter 3.2*. I have also presented possible access patterns that can occur within the massively parallelized computation in the GPU in *Section 3.3.1*. In fact, there are some potentially complicated access patterns including gather/scatter, stencil or even random memory access incurred by the *diffeomorphic log-demons algorithm*. In response to those potentially bottlenecking memory transactions, the GPU possess different strategies to resolve the transactions efficiently.

With such GPU memory accessing schemes in mind, we have looked into the algorithm breakdown of *diffeomorphic log-demons* in an attempt to find out the bottlenecking operations in *Section 3.3.2*. Particularly, we found that two of the operations, namely Gaussian regularization and vector field composition, are bottlenecking the whole registration process. By revisiting the hardware microarchitecture of the GPU, I have proposed to exploit the memory access patterns and texture hardware to resolve the demanding computations in *Section 3.4*. As a result, different performance-aware optimization strategies are proposed, which will be implemented and tested in **Chapter 4**.

Chapter 4

GPU-BASED IMPLEMENTATION OF DIFFEOMORPHIC LOG-DEMONS

4.1 Introduction

The computation bottleneck of the *diffeomorphic log-demons* algorithm has been identified in the previous chapter. As presented in *Chapter 3.4*, a number of optimization strategies can be adopted on the GPU to resolve the major computation bottleneck that precludes fast computation. This chapter evaluates the optimization effectiveness of such strategies. To effectively visualize the enhancement by such performance-aware computing techniques, I have performed individual testing on each blocking operations of *diffeomorphic log-demons* (*Section 3.3.2*). Multiple implementations of these bottlenecking operations will be presented and compared to the run-time required by the CPU. Finally, the optimal implementations of different computation modules are assembled into a working, optimized GPU implementation of *diffeomorphic log-demons* for validation in terms of computation enhancement.

4.2 Major performance limiter

To apply performance-aware programming techniques onto the computation kernel, knowing which part of the GPU is bounding the computation is essential. Indeed, there are numerous reasons that can hold back the GPU's computation performance. Instead of exhausting the list of bottlenecking reasons, they are categorized into 3 major performance-bounding factor, namely compute-

bound, memory-bound, or latency-bound. Using the NVidia visual profiler (nvvp) that comes with the CUDA toolkit, one can identify the performance limiter by looking into the performance metrics. In fact, implementing performance-aware techniques on GPU is an iterative process, which involves numerous iterations of profiling and optimization of the GPU kernel. Once the performance-limiting factor is identified and pinpointed, it is required to take corresponding actions to resolve such issue.

Table 4.1 Key metrics shown in the profiler for the identification of performance limiter in a GPU kernel.

Performance limiter	Key identifying metrics	Warp stall reasons
Computation-bound	High arithmetic pipe utilization	Execution dependency
Memory-bound	High memory bandwidth usage,	Memory throttle
Latency-bound	High L2 Transactions per request Low Memory-computation overlap	Memory dependency

Table 4.1 presents the key identifying metrics of the three major types of performance limiters shown by the CUDA profiler. As depicted by the name of the performance limiter, the performance of compute-bound kernel is limited by the immense workload that saturated the multiprocessors. This can be introduced by the workload exceed the theoretical operations per second (OP/s) of the hardware, or because the algorithm is not optimized. Similarly, memory-bound situations can be invoked by hardware limit, when the required memory transaction exceeds the hardware capability. Finally, latency-bound kernels are not bounded by the memory or computation hardware capability. Instead, it is introduced by the latencies that present in computation and/or memory transactions due to poor memory transaction pattern, or poor overlap between the computation and memory operations.

4.2.1 Compute-bound and memory-bound kernels

Compute-bound kernels can be identified by high compute throughput with respect to the hardware limit. A general rule-of-thumb of identifying compute-bound kernels is the achieved compute throughput is over 60% of the hardware limit. Similarly, memory-bound kernels can be identified if the profiler the memory throughput is over 60% of the theoretical memory bandwidth. Compute-bound and

memory-bound kernels are generally caused by high utilization of the device. As such, if the profiler indicates signs of the kernel are being bounded by compute- or memory-related issues, the programmer can affirm that a considerable amount of computation has been done efficiently. However, there is also the possibility that extra computation effort was spent due to the algorithm is not optimized, or due to unnecessary memory transactions that bottlenecked the kernel. Therefore, the first step to resolve for compute bound kernels is to always look for ways to improve the compute efficiency.

For example, enabling the *--use_fast_math* compiler option enables the GPU device to utilize the more optimized fast math library for faster execution of mathematical functions such as sin, $sqrt$ and $logf$, of which they can be computationally expensive. This compiler option enables the multiprocessors to further off-load such expensive computation to the special function unit in each multiprocessor for faster computation with slightly lower accuracy. While the speed difference has not been reported in the CUDA programming guide to-date, it has been reported that using the *--use_fast_math* options alone can, not only reduce the number of instructions generated for the computation, but also reduce the latency of performing such computation.

Furthermore, thread divergence is also another important factor to be tackled when the kernel is bound by computation. Under the SIMT architecture, warps of 32 threads will have to be executed together. Thread divergence will be observed if the threads within a warp take different execution path due to divergence in the control flow. As such, upon executing a divergent warp, both execution paths will have to be executed, with some threads being deactivated. Thus, resulting in a low thread execution efficiency. In such regard, it may be wise to decompose any complex execution branch, or group such diverged threads into a single warp for efficient execution.

4.2.2 Latency-bound kernels

Contrary to compute- or memory-bound kernels, latency-bound kernels can be identified by low compute and memory throughput compared to the hardware limit of the device. As a rule of thumb, the kernel can be classified as latency-bound if the kernel is not bound by either compute or memory throughput using the 60% utilization rule. Computation latency is one of the major reasons that can lead to under-utilization of the device. Normally, the CUDA hardware can automatically mitigate such memory/compute latency by concurrently executing multiple warps under SIMT. However, if there is a great amount of latency in a kernel of which they that cannot be hidden by warp concurrency, the CUDA device will be stalled and therefore being bounded by latency.

It is often important to reemphasize fetching data from the global memory costs hundreds of GPU cycles, which is approximately 100x costlier than performing basic arithmetic operations [105]. This extended time required for the SM to access the global memory incurs much latency. Given memory operations can be ubiquitous throughout the kernel run time, it is the most common reason that causes latency bottleneck. Thus, being bound by memory latency is one of the most common reasons for most GPU kernels to stall. To ensure memory transactions are performed efficiently is the key to deal with latency-bound kernels. *Chapter 3.4* has presented a number of ways to ensure full utilization of the memory bandwidth. In all, it is of the utmost significance of ensuring coalesced memory read on the global memory, as well as properly reusing the data by the shared memory.

However, it is not uncommon to discover the kernels are still bounded by memory latency, even with all precautions being taken. In that sense, neither the computation pipelines nor the memory bandwidth was bottlenecking the computation. Such situation is usually observed in small kernels with low warp concurrency with relative light computation. Besides, latency-bound kernels are also commonplace when thread barriers are extensively used to prevent any data-racing conditions within a thread block. In this light, one can manually decrease the thread-level parallelism by reducing the block size, but increase the block-level concurrency to allow the GPU to run more blocks at once in order to hide such latency (**Figure 4.1**).

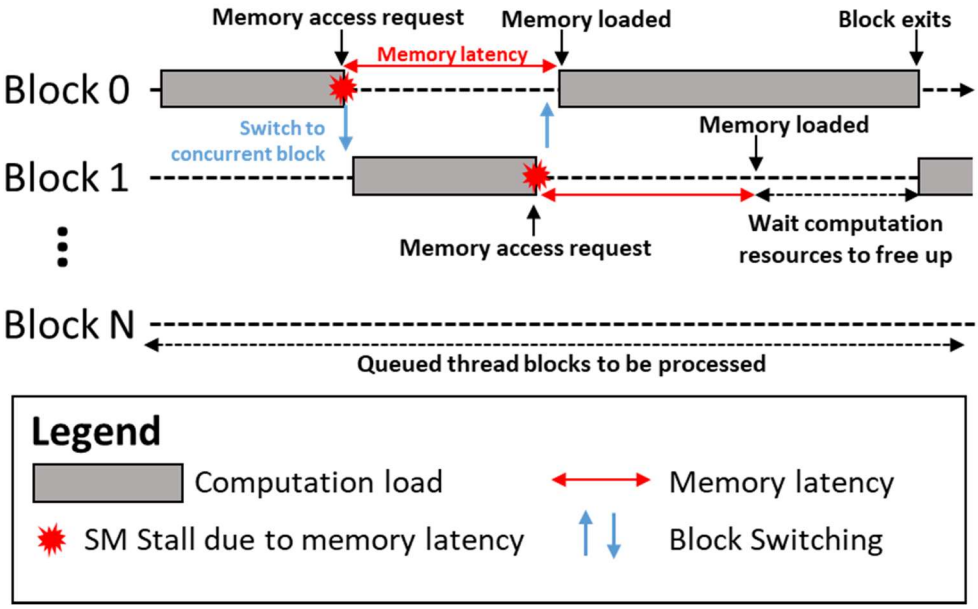

Figure 4.1 Concurrent execution of multiple thread blocks on a single streaming multiprocessor. This block-level concurrency can mitigate memory latency.

4.3 Optimization results for bottlenecking operations

As presented in *Chapter 3.4*, the two bottlenecking operations of *diffeomorphic log-demons* are: *(1)* Gaussian smoothing involved in regularization of the update field and velocity field; and *(2)* vector field composition required in the *scaling and squaring* methods for computing the deformation field. As the *diffeomorphic log-demons* algorithm is iterative in nature, these bottlenecking operations are repeatedly called throughout the registration process. Because the operations can incur a vast amount of different memory access patterns that preclude fast computation, several optimization strategies have been proposed in *Chapter 3.4.* Such strategies aim to resolve these computation hurdles with the aid of different hardware features in GPU. In this section, I will present the optimization results for the bottlenecking operations that presented previously.

4.3.1 Testing platform

To illustrate the importance of different performance-aware programming techniques, I have conducted a series of experiments on the bottlenecking operations. These operations, namely Gaussian smoothing and vector field composition, are the major bottleneck in *diffeomorphic log-demons*. Most of the

computation time has been spent on these operations. In that light, one could achieve fast registration if these computation bottlenecks are resolved. The experiments also allow quantification of computation enhancement brought by the performance-aware computing techniques. All experiments were conducted using a vector field/deformation field generated by a pre-deformed 3D MRI brain imzage using TPS. To investigate the effect of varying image dimensions on the computation, the dataset was up/down-sampled to 7 separate sets of images with resolution ranging from 1×10^6 to 3.6×10^7 voxels. **Figure 4.2** shows the pair of input image sets used for testing of the bottlenecking operations.

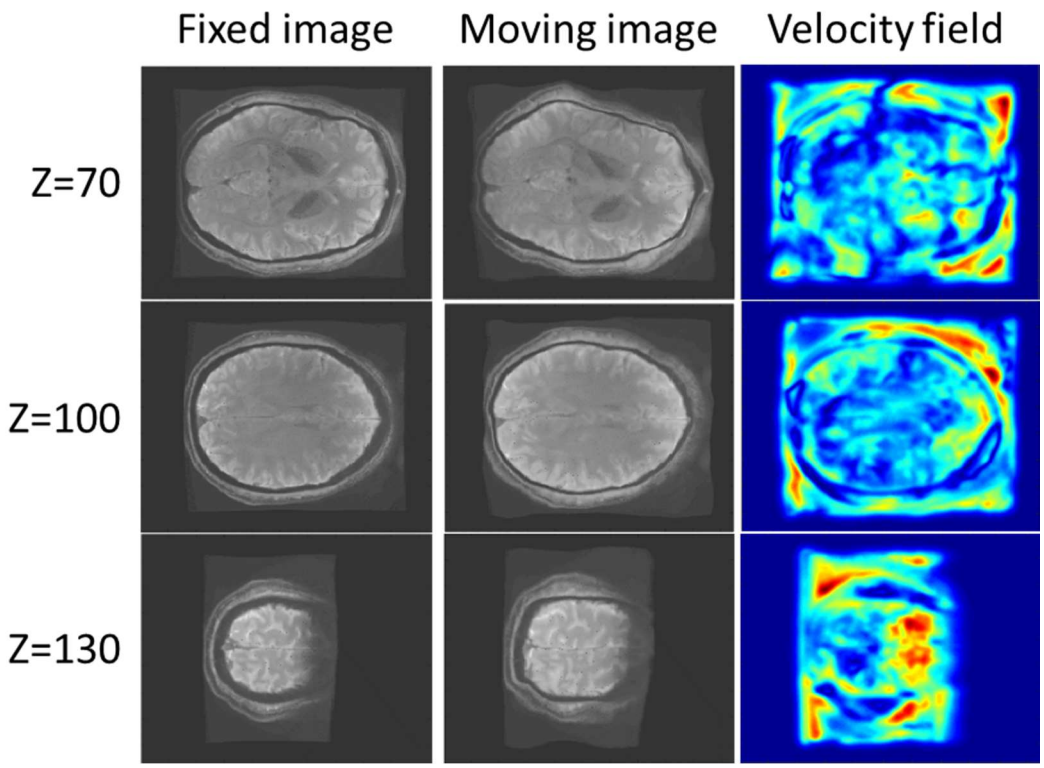

Figure 4.2 Dataset used to evaluate the optimization performance at 3 distinct slices. Pre-/post- deformed brain MRI images were used as the fixed/moving images of the registration. The generated velocity vector field from the fixed/moving image pair is shown on the right with warmer color indicating higher vector magnitude.

The GPU-related experiments were conducted by a PC with an i7-4790 CPU running at 3.6GHz equipped with an NVIDIA GTX Titan X GPU. The GTX Titan X is a high-end GPU which features 24 streaming multiprocessors and 12GB global memory. To avoid interference due to host-device memory transaction overhead, all necessary data are transferred in prior to the experiments in this section.

4.3.2 Optimization for Gaussian smoothing

As presented in Section 3.4.1 the GPU's shared memory is able to mitigate much of the memory transaction latency by achieving effective caching. Using the GPU's shared memory to compute for Gaussian smoothing consist of a 3-phased process: *(1)* pre-fetch all necessary data onto the shared memory; *(2)* once the data are loaded onto the shared memory, then each thread compute the weighted sum of all elements within the Gaussian kernel along the direction of multi-pass convolution; and finally (3) when all the computation is completed, the threads store back the temporarily stored results from the register to the shared memory, and subsequently store the result back to the shared memory. To avoid data-racing conditions, these computations have to be performed in a step-wise manner. The data on the shared memory must be initialized prior to computation. Any temporary results must wait for all computation in a block to finish before storing them back to the shared memory. Step-wise computation can be assured by using synchronization instructions (i.e. calling *__syncthreads()* in the kernel) at the end/beginning of each phase to avoid indeterminstic output due to any potential racing conditions.

To systemically observe and evaluate the effectiveness of the optimization, the computation time on the GPU is compared against the time of a single-threaded CPU implementing the same algorithm. In particular, multi-pass Gaussian smoothing on a vector field stored in SoA format is performed by three implementations on GPU, namely: *(i) n*aïve implementation of multi-pass Gaussian smoothing without any data reusing *(global memory only)*; *(ii)* reusing the data with shared memory but without enforcing memory coalescences *(shared memory)*; and (*iii)* reusing the data with optimized global-shared memory transaction *(shared memory /w coalesced transactions)*. The variance (σ) of the Gaussian kernel was set to 3, which is a typical value used by *diffeomorphic log-demons*.

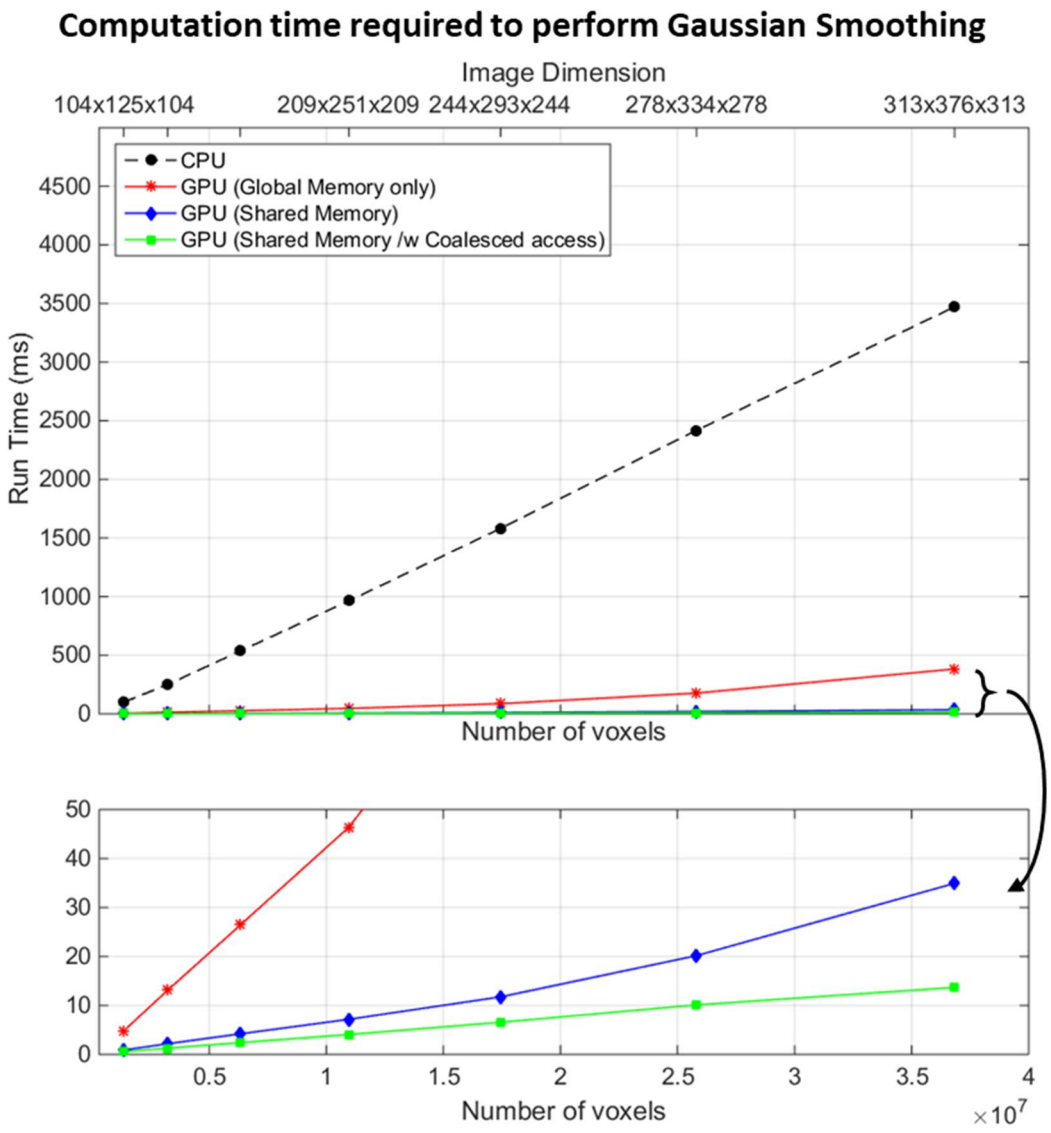

Figure 4.3 Computation time required to perform Gaussian smoothing on the velocity fields at 7 levels of resolutions by CPU and the 3 implementations on GPU. The GPU implementation that ensures data coalescence during shared memory initialization significantly outperforms other implementations.

Figure 4.3 presents the computation time required to perform Gaussian smoothing. It is obvious that the CPU struggles to provide enough computation throughput for the operations. As shown by the black dotted line, the required time for computation using CPU ranges from 100ms for the smallest test vector field dimension (104×125×104) to 3500ms for the largest input vector field dimension (313×376×313). In contrast, the GPU can perform the computation significantly faster compared to CPU, even without any optimization. The most naïve approach of using GPU can achieve the same computation within 5ms to 400ms depending

on the resolution. Such computation speed-up is achieved by the massive computation throughput by the GPU due to SIMT. However, as discussed in previous chapters, the computation bottlenecks do not solely reside in the computation throughput, but also in the memory latency between the global memory and streaming multiprocessors. With the introduction of shared memory to alleviate the memory bandwidth demand, a further reduction of computation time is observed. This reduction can be visualized by the blue line, which indicates the GPU requires 0.9ms to 35ms to compute for the Gaussian smoothing. Further computation improvements can be yielded by managing the thread block dimensions to enforce memory read coalescence. As shown by the green line the computation time can be further reduced to 0.5ms – 13.5ms.

Breakdown of the computation time consumed by different convolution passes in the sub-optimal and optimal implementation of Gaussian smoothing is presented in **Figure 4.4**. It has been observed that the time consumed by the *z*-pass smoothing on the sub-optimal implementation takes up the majority of the computation runtime. The time required is significantly worsened when the input dimensions increase to 26M voxels (278×334×278 voxels). At this input image dimension, fetching data onto the shared memory across the *z*-direction requires accessing the global memory with a large stride of 278×334×sizeof(float)=363KB which will exceed the maximum caching capacity of that an L2 cache chip can provide. Therefore, much memory latency will be introduced as the memory controller attempt to fetch the requested data from the neighboring L2 chips. As such, not only the memory bus is under-utilized due to loss of memory coalescence, but also each transaction request from/to the global memory will incur additional latency due to cache misses. Conversely, with coalesced memory transaction in the implementation with managed thread block size, such memory latency can be partially mitigated due to a full utilization of the memory bus.

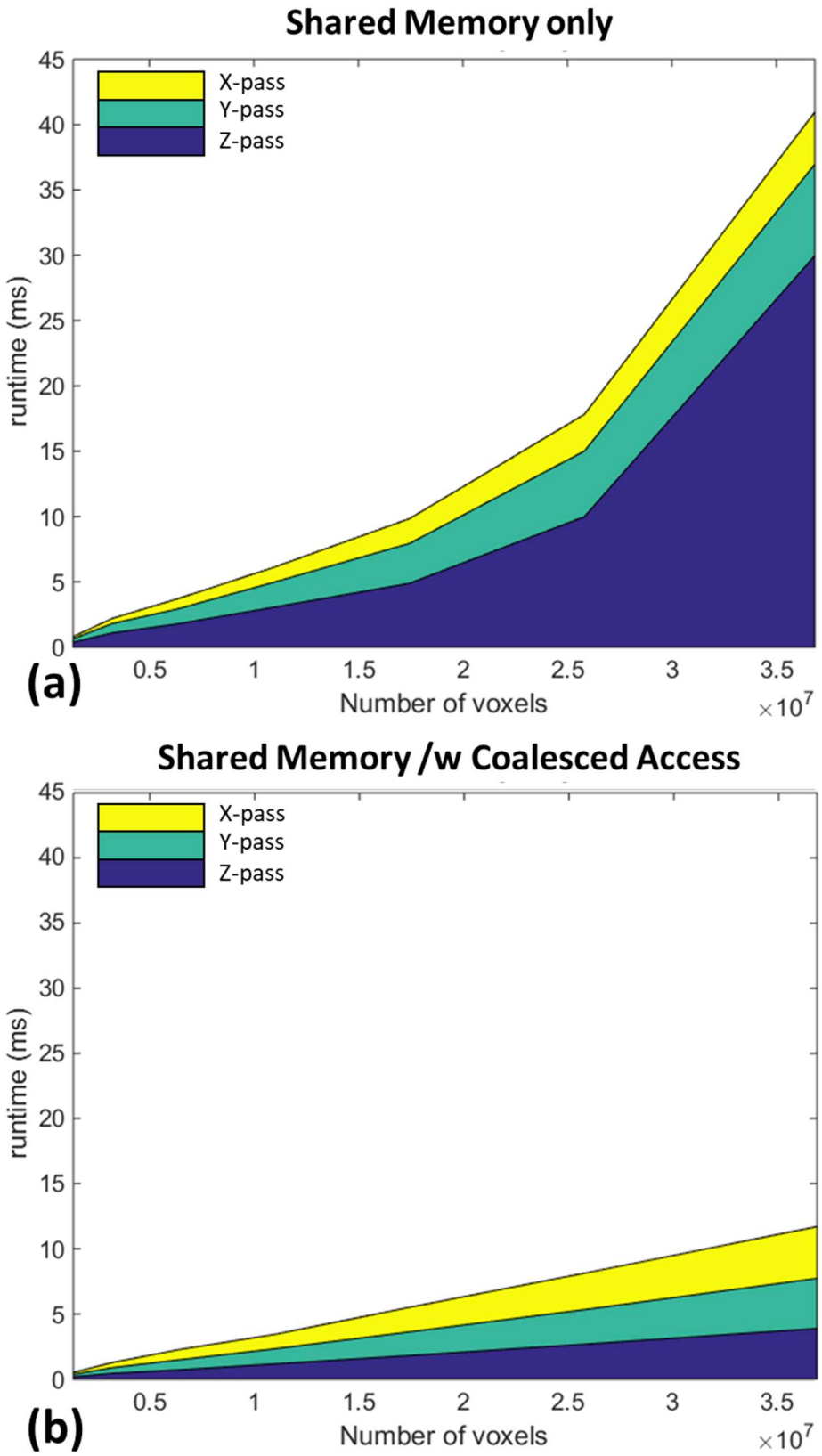

Figure 4.4 Breakdown of computation time by various passes in **(a)** sub-optimal implementation; and **(b)** optimal implementation of multi-pass Gaussian smoothing process. The *z*-pass in the sub-optimal implementation takes up considerable computation runtime.

Fetching the data with the across the *y*-direction shares the same idea. Despite the algorithm will only require striding through the memory with relative small step of 278×sizeof(float) = ~1KB , the bandwidth under-utilization due to loss of memory coalescence can still hamper data transaction efficiency. Such inefficient memory transaction can be reflected by the run-time discrepancies between the sub-optimal and the optimal implementation of the algorithm.

Figure 4.5(a) presents the derived computation speed-up of different implementation on GPU compared to a single-threaded CPU, which can reveal a clearer picture on the computational enhancement. **Figure 4.5**(b) indicates the kernels are not limited memory bandwidth, as the achieved bandwidth sits well below the maximum allowable bandwidth of the GPU (120GB/s out of 336.5 GB/s). Instead, the performance limiter resides in memory latency. Regardless, the naïve GPU implementation resulted in a 20× computation speed-up initially but falls off when the input dimension increases. The sub-optimal implementation using shared memory in general outperforms 6-10 times better than the naïve implementation. It showed around ~140× computation speed-up but falls to ~100× upon high input dimension like the naïve implementation. Let aside the abnormality of computational speed-up at low input dimension, the optimal implementation, which enforces memory coalescences, perform 1.7-2.2× better than the sub-optimal GPU implementation. Moreover, the performance does not fall off upon high input dimension, but instead leveling off at ~250× speed-up. A similar pattern is observed in **Figure 4.5**(b), where naïve implementation that only uses the GPU's global memory have under-utilized the memory bandwidth due to repeated, un-coalesced memory access. The sub-optimal implementation using shared memory utilized 5-10× more bandwidth than the naïve implementation, and the optimal implementation further utilized 1.7× more global memory bandwidth than the sub-optimal implementation.

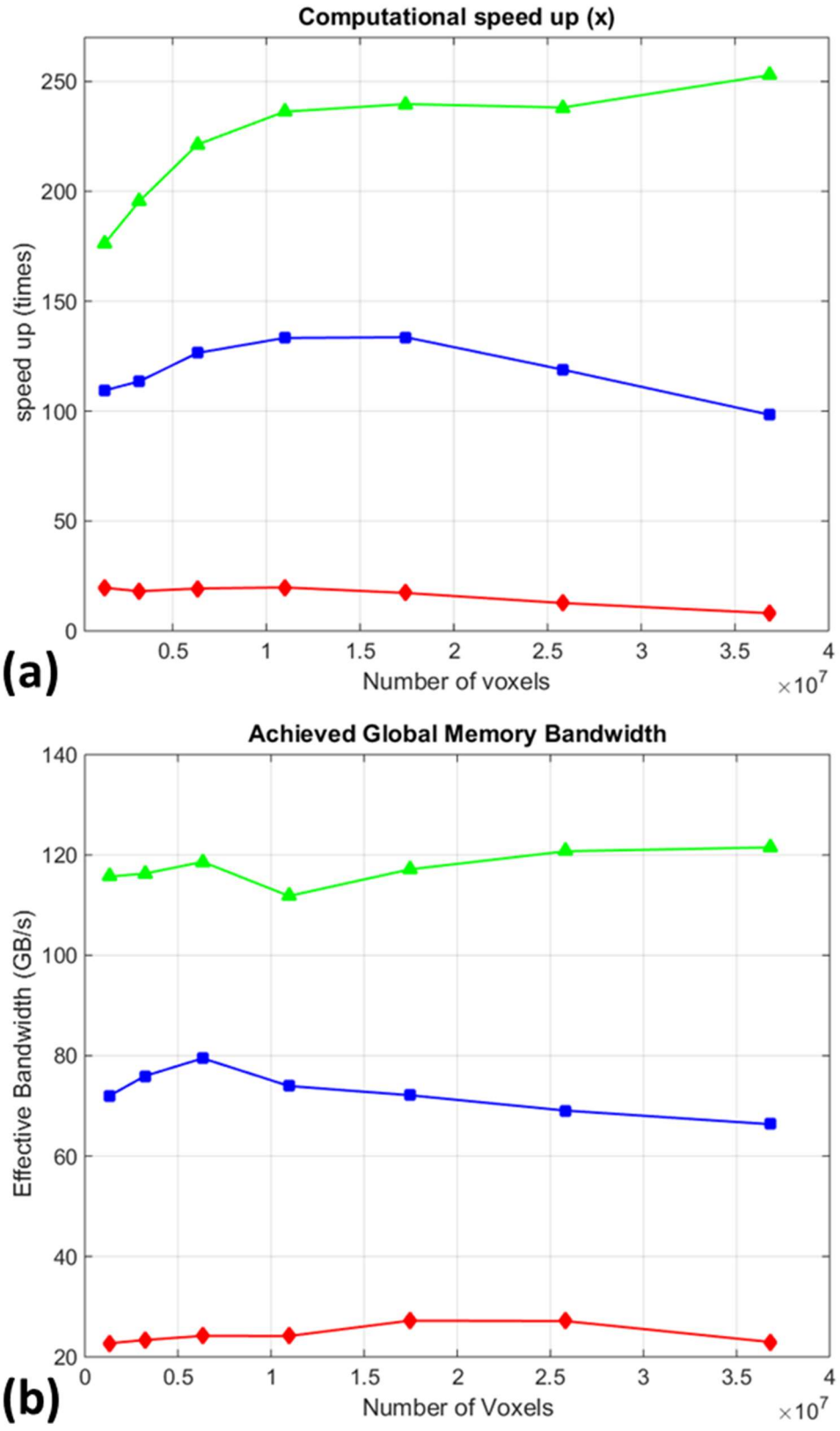

Figure 4.5 **(a)** Effective computation speedup relative to CPU; and **(b)** Achieved global memory bandwidth by various GPU implementations at 7 different input resolutions.

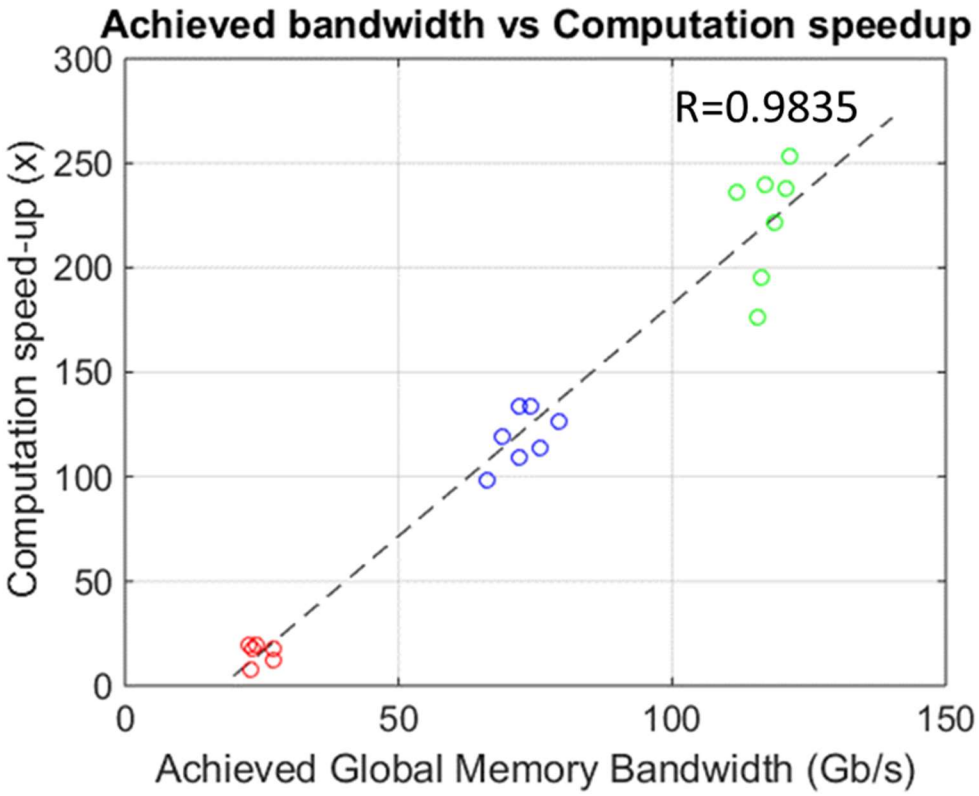

Figure 4.6 Correlation between computation speed-up and achieved global memory bandwidth. Significant correlation (R=0.9835) suggests the sub-optimal implementations are bounded by the memory latency which causes bandwidth underutilization.

Figure 4.6 illustrates significant correlation between the utilized memory bandwidth and the achieved computation enhancement. Such correlation further suggests the computation of Gaussian smoothing is bounded by the memory latency caused by suboptimal memory access schemes. However, it is also noticeable in **Figure 4.5**(a) that, for the optimal implementation, the computation speed-up is lower when the input vector field dimension is small (~1M voxel) while the achieved memory bandwidth utilization remains constant. This suggests that the computation kernel has been bounded by the kernel's computation instead of memory latency. In fact, the memory latency can be hidden by concurrent warp execution under SIMT. However, at low vector field dimension, there will be insufficient number of warps eligible for the streaming multiprocessor to execute concurrently. Thus, the lack of eligible warps stalled the device which undermined computation efficiency. At higher vector field resolution, there will be more warps being eligible for concurrent execution by the streaming multiprocessor which can be used to effectively hide part of the memory latency.

4.3.3 Optimization for vector field composition

Similar to the experiments performed on Gaussian smoothing, I have conducted experiments on various implementations of vector field composition to quantify the effectiveness of computation enhancement. Again, the run-time of single-threaded CPU will be used as the baseline of the computation. As depicted in previous sections, the CPU will not be able to provide sufficient computation throughput. Three different implementations will be tested: *(i)* manual implementation using trilinear interpolations on the global memory *(manual interpolation)*; *(ii)* using texture hardware to perform interpolation on a vector field stored in SoA *(texture interpolation)*; and *(iii)* using texture hardware interpolate the vector fields in AoS *(vectorized texture interpolation)*.

Figure 4.7 presents the computation time required to perform a self-composition on CPU and GPU. Similar to the investigations in the experiments performed on Gaussian Smoothing, the CPU struggles to provide sufficient computation throughput to support the vector field composition, especially at a large input dimension. Even with the lowest vector field resolution (104×125×104), the CPU still requires 133ms to complete the composition process. Computation time also increases linearly with respect to the number of voxels. At highest testing resolution (313×376×313) the CPU requires over 3500ms to complete a single composition operation.

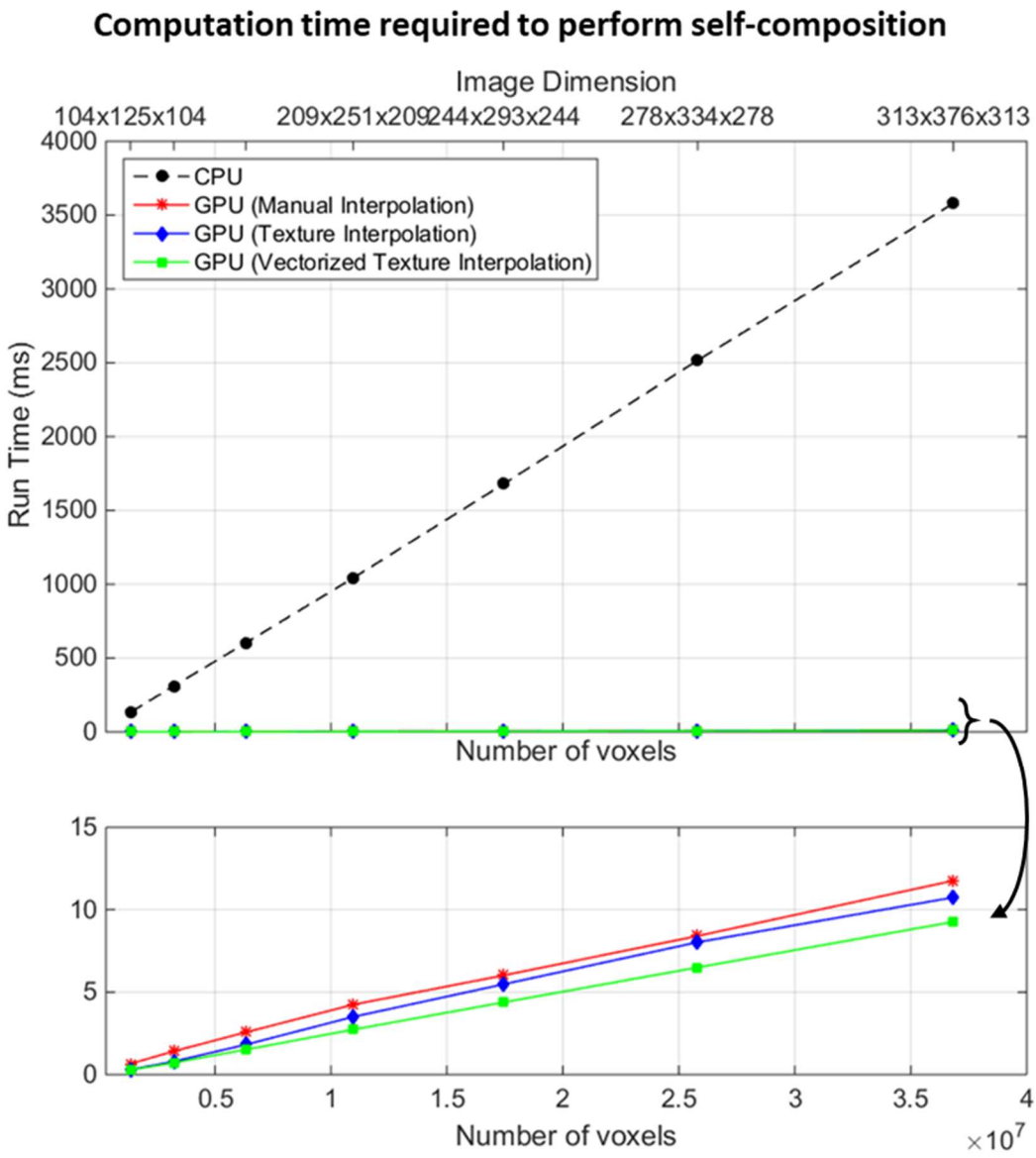

Figure 4.7 Computation time required for self-composing a velocity field on CPU and the 3 GPU implementations at 7 levels of vector field resolution.

On the GPU, all 3 implementations achieved significant performance enhancement compared to CPU. This suggests that the GPU can provide the much-needed computation throughput for the operation. Contrary to the experimental results obtained in Gaussian smoothing, the performance enhancements obtained among different GPU implementations of vector field composition does not show as large variations. Particularly, all of the implementations require around 1ms to self-compose the smallest test vector field, and 9-11ms to self-compose the largest vector field. Despite the three implementations showed similarity in terms of run-

time, they still show deviations in terms of achieved computation speed-up as presented in **Figure 4.8**.

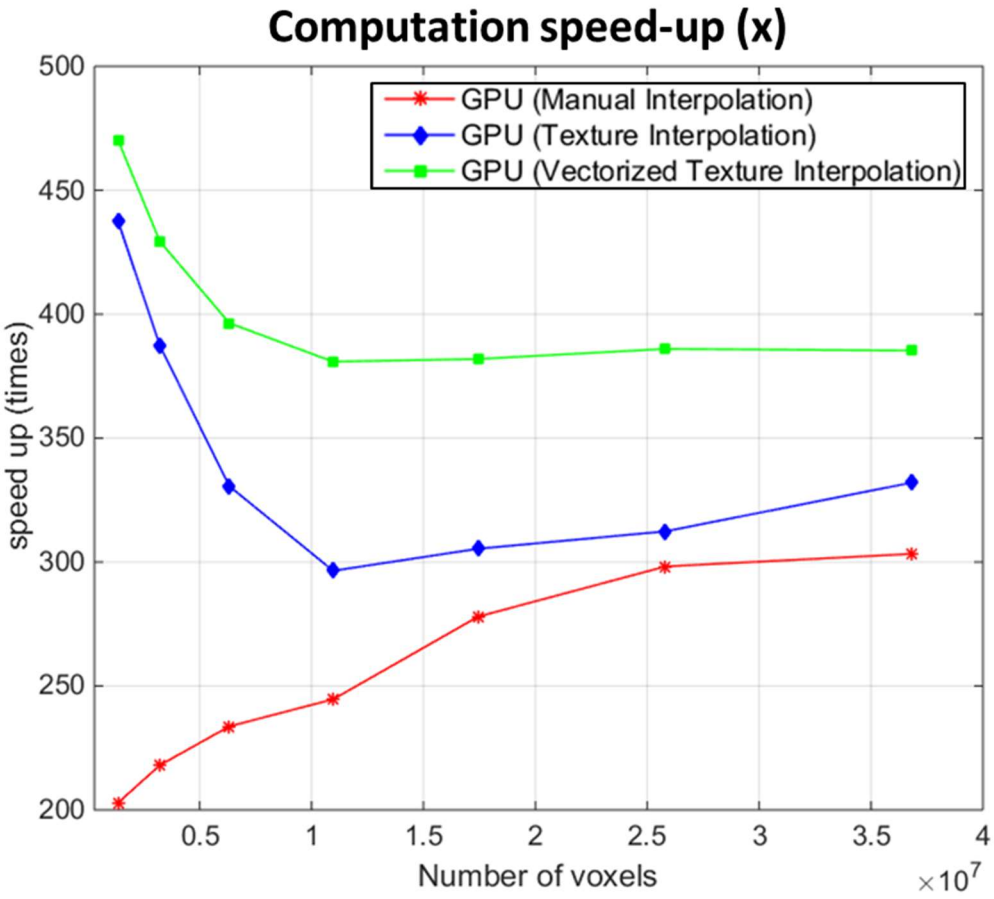

Figure 4.8 Achieved computation speed-up relative to CPU by the 3 GPU implementations of vector field self-composition.

Figure 4.8 presents the achieved computation speedup by different GPU implementation compared to CPU. As illustrated by the red line, manual interpolation using native computation instruction on GPU struggles to achieve a very high-performance enhancement at low resolution. However, as the input dimension increases, the performance slowly increases until being leveled off at around 300× speed-up. Besides, both texture memory implementations showed a very high-performance speed-up at lower input resolution. However, the speed-up achieved by both texture implementations falls off when the input dimension increases. For the implementation without AoS, the computation speed-up level-off from ~440× to ~300× at 10M voxels. Texture implementation with AoS shows significant improvement. Despite the speed up also levels off at 10M voxel for the

texture implementation with AoS, the improvements fall from~470× to ~380×, which is a 10-20% performance increase.

Although the GPU implementations of texture interpolation showed significant (>250×) computation improvement, there are still a few questions that are left unanswered by **Figure 4.8** which have to be accounted for, namely:

1. The reason for the increased performance for both texture memory implementations memory at low resolution;
2. The causes for the performances gain to stabilize and level off at higher input dimensions; and
3. The increased performance of the native interpolation implementation upon the increase of input vector field dimension.

To resolve such questions, one has to resort to micro-benchmark the performance of GPU kernels. In performance-aware programming, it is often important to identify the bottlenecking by micro-benchmarking the computation. In fact, the profiler has shown that most of the kernels are indeed bounded by the global memory bandwidth. As presented in *Section 2.4.1*, any memory access to the global memory is cached by both L1 and L2 caches. Therefore, the term *"bottlenecked by global memory bandwidth"* can imply bottlenecking due to insufficient L1-L2 bandwidth, insufficient L2-GDDR bandwidth, or both. In this light, I looked into the achieved bandwidth between the L1 and the L2 cache, as well as the bandwidth between the L2 cache and global memories for these implementations.

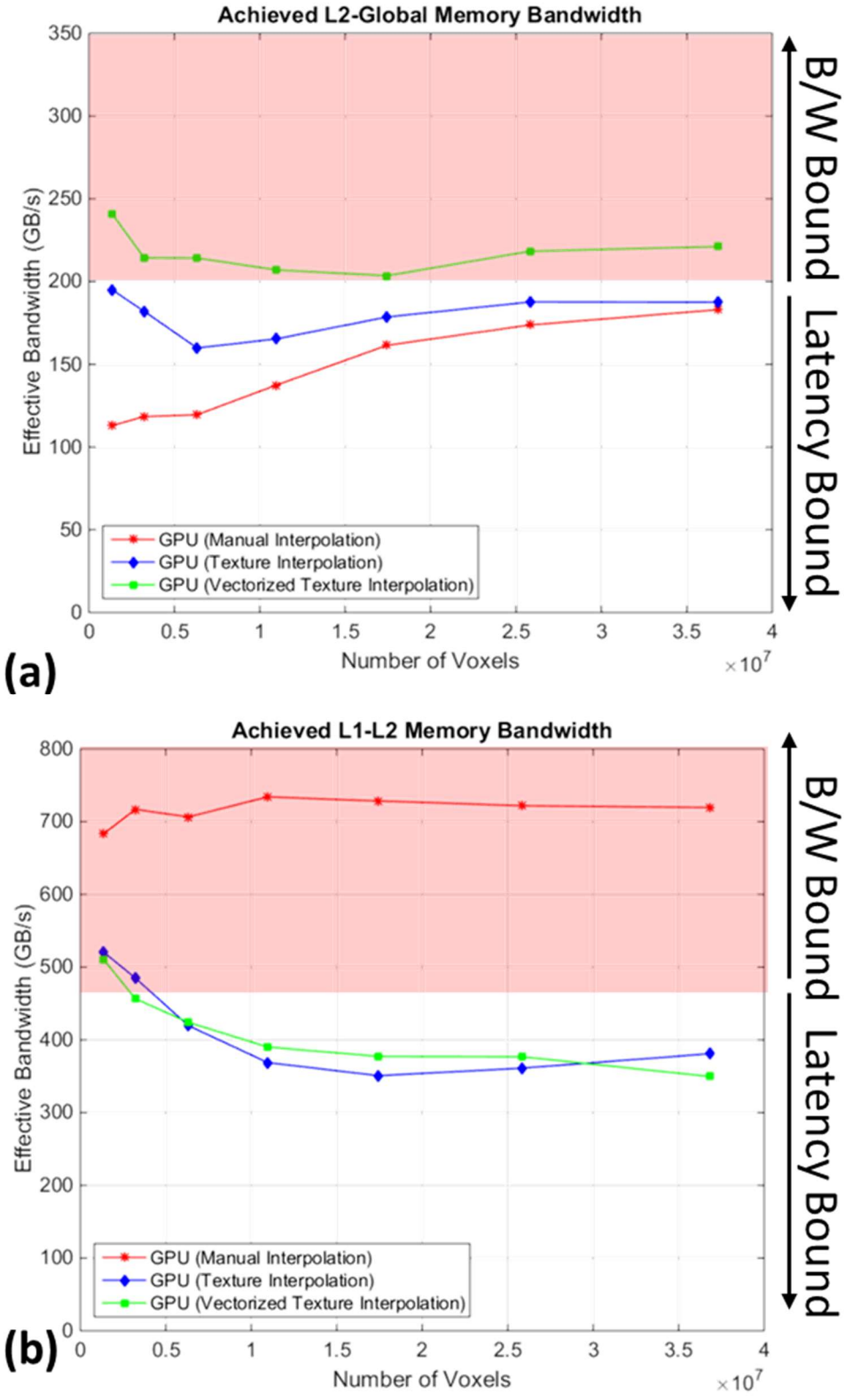

Figure 4.9 Bandwidth statistics obtained by NVidia profiler. **(a)** Achieved L2-global memory bandwidth; **(b)** achieved L1-L2 memory bandwidth for the 3 implementations of vector field composition. The benchmark for distinguishing bandwidth or latency bound kernels (200 GB/s for L2-global memory; 470 GB/s for L1-L2) are also indicated.

Figure 4.9 presents the achieved memory bandwidth involved in the global memory transaction. The L2-Global memory bandwidth shows an overall good utilization for all 3 implementations, but there is a significant (>10%) better utilization on the L2-global memory bandwidth for the vectorized texture interpolation implementation (green curve).

The reason for the increased performance for both texture implementations at low resolution can be explained by the achieved bandwidth between the L1 and L2 caches. As shown in the L1-L2 bandwidth curve, there is a significantly higher effective bandwidth achieved when the vector field resolution is low. This can be accounted by the fact that the L2 cache can accommodate a larger portion of data. Therefore, the increased L2 hit rate facilitates memory performance. Moreover, lower input resolution also implies a smaller memory stride for any for texture hardware-managed memory fetches.

To address the disparity between the texture implementations with and without AoS, particularly at high input resolution, I looked into the memory throughput of the device. The theoretical L2-Global memory throughput of the device, defined by $channel\ throughput\ \times bus\ width$, is found to be

$$7Gbit/s \times 384\ = 2688Gbit/s = 336\ GB/s \tag{10}$$

Therefore, the 60% utilization rule threshold can be calculated:

$$336GB/s \times 60\% = 205GB/s \tag{11}$$

As such, the benchmark for distinguishing the kernel to be bandwidth bound or latency bound is defined. By applying such rule, one can conclude that the vectorized texture interpolation implementation was bottlenecked by the L2-Global memory throughput regardless of input dimension. However, for the texture implementation without AoS, despite having identical L1-L2 bandwidth usage, the L2-Global memory bandwidth showed under-utilization (blue curves in **Figure 4.9a**). As a matter of fact, due to the lack of vectorization, there will be a significantly more transaction request on the global memory. These extra

transaction requests can cause contention on the memory controller, thus resulting in the under-utilization of the global memory bandwidth.

Regarding the increasing performance of the native interpolation implementation upon the increase of input vector field dimension, I have to look into the L1-L2 memory bandwidth utilization. As depicted in **Figure 4.9b**, despite the implementation of vector field composition using manual interpolation have underutilized the L2-Global memory bandwidth, a very high L1-L2 bandwidth utilization is observed. Such discrepancy in bandwidth utilization indicates a low L1 hit rate with a very high L2 hit rate, which is commonplace when numerous threads attempt to stride through the memory from indexed memory locations.

Due to the non-parametric property of the vector field, it is not possible to employ data reuse strategies onto the implementation of vector field composition. As the vector field defining the query points are non-parametric, one can consider the underlying memory transaction pattern as random. As such, all memory transactions will have to be accessed directly onto the global memory through the L2 cache. However, the L1 cache is optimized to perform coalesced memory transfer by reading/storing 128 bytes at once, numerous memory transaction requests form a large number of threads for parallel computation will introduce much traffic between the L1 and the L2 caches. To the best of our knowledge, the theoretical bandwidth between the L1 and L2 cache are not announced by the GPU manufacturer. However, the nvvp will consider the L1-L2 bandwidth to be bottlenecking when the L1-L2 utilization exceeds 460GB/s. Therefore, it can be confirmed that the latency due to insufficient L1-L2 bandwidth bottlenecks the computation.

To recovery of computation speed-up in the implementation with manual interpolation at higher resolution can be accounted by warp concurrency. With sufficient block occupancy, the warp scheduler can afford operations with higher latency, as long as there are remaining eligible warps for the underlying CUDA cores to execute. Warp concurrency can also be facilitated by the relatively balanced memory/computation load in manual interpolation which allows an even distribution of workload between the memory and execution controllers.

4.4 Overall optimization results

Previous sub-sections presented the performance-aware programming techniques used to optimize the two bottlenecking operations in *diffeomorphic log-demons*. With those optimization strategies in mind, the whole *diffeomorphic log-demons* algorithm was eventually implemented in a modular manner. For instance, each module was iteratively profiled and carefully optimized before assembling. The assembled registration tool can effectively register the images, as shown in **Figure 4.10**, even with very large deformation.

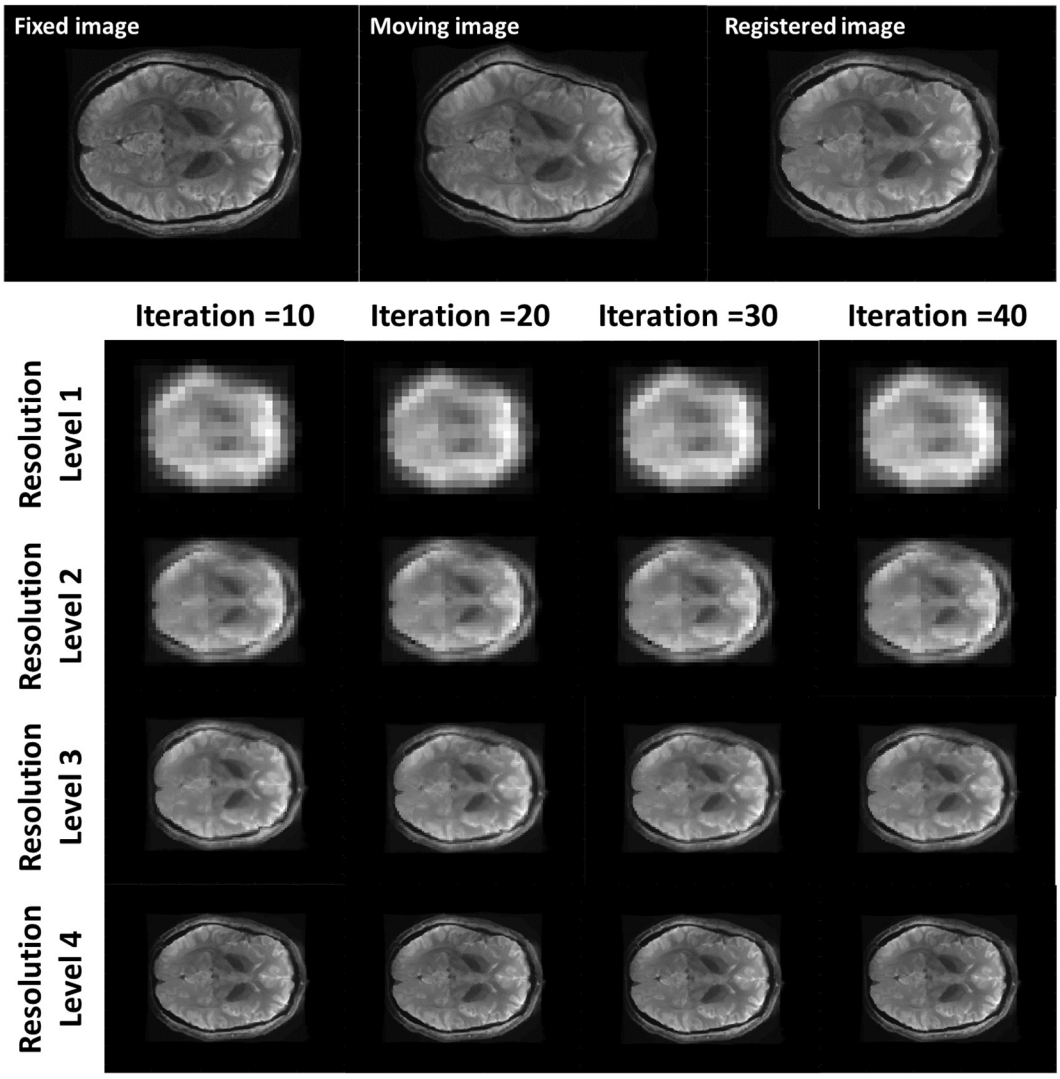

Figure 4.10 Registration process of a highly deformed image. Misalignments between the fixed and the moving images are iteratively resolved by our GPU *diffeomorphic log-demons* implementation using a coarse-to-fine multi-resolution registration approach.

In this section, I present the optimization strategies applied onto different modules, as well as the resultant computation speedup achieved. Further to the algorithm breakdown of *diffeomorphic log-demons* in *Section 3.3.2*, the detailed modular breakdown of our GPU implementation, including the critical computation modules are presented:

Table 4.2 List of optimized GPU computation modules in the implementation of diffeomorphic log-demons.

Computation Module	Used by	Optimization Strategies
Gradient decomposition	Update field generation Registration energy evaluation	Enforce coalesced memory transfer Data re-use using shared memory Use of vectorized data type
Finite image difference	Update field generation Registration energy evaluation	Enforce coalesced memory transfer Optimize occupancy
Gaussian smoothing	Regularization of vector fields	Enforce coalesced memory transfer Data re-use using shared memory Use of vectorized data type
Vector field composition	Computation of deformation field Updating the velocity field	Make use of texture interpolation Optimize for occupancy Use of vectorized data type
Image warping (composition)	Updating the moving image for next iteration	Make use of texture interpolation Optimize for occupancy
Computation to generate velocity update field	Update field generation	Ensure coalesced memory transfer Tackle thread divergence
Sum/ maximum reduction*	Computation of deformation field Registration energy evaluation	Warp shuffling Loop unrolling Tackle thread divergence

*An open-sourced GPU library, thrust [103], was used to implement sum/maximum reduction.

Table 4.2 presents the list of optimized GPU computation modules that are used in our implementation of *diffeomorphic log-demons*. As the GPU utilizes its highly parallelized SIMT architecture to perform high throughput computation, it is not surprising to see that most of the optimization techniques are focusing on efficient use of its memory. Previous investigations on the two bottlenecking operations also illustrated most GPU kernels are indeed bounded by memory

latency. Such latency-bounded kernels are eligible to be optimized by allowing data reuse, employing vectorized data structure, as well as using specialized GPU hardware. With such regard, the optimization performance of each computation step, for all test cases from 1M voxels to 37M voxels, in the optimized GPU-enabled *diffeomorphic log-demons* is presented:

Table 4.3 Performance gain for different computation steps in an iteration using the GPU implementation of diffeomorphic log-demons.

Computation step per iteration	**CPU time (ms)**	**GPU time (ms)**	**Speed-up (×)**
Compute update field	35 – 904	0.6 – 18.7	**48× – 63×**
Vector field regularization	266 – 7226	1.1 – 23.2	**188× – 274×**
Update velocity field	133 – 3579	0.5 – 13.8	**255× – 294×**
Compute deformation (avg.)	359 – 15448	1.3 – 52.5	**275× – 296×**
Update moving image	107 – 2786	0.3 – 8.9	**311× – 336×**
Evaluate registration energy	54 – 1454	0.4 – 13.4	**108× – 151×**
Avg. time required per iteration	***884 - 31146***	***3.9 – 140.4***	***216× - 234×***

Table 4.3 presents the performance speedup achieved by optimized GPU implementation of the computation steps in *diffeomorphic log-demons*. As almost all computation steps in the algorithm are inherently parallelizable, the GPU is able to achieve an impressive >200× computational speed-up for most computation. Despite CPU have twice the clock speed compared to GPU, its inability to perform SIMD or SIMT have heavily set back its computation throughput. As such, the CPU will have to heavily rely on loops to perform the needed voxel-wise computation, which will come with the cost of a significant overhead due to pointer arithmetic and control flow. In fact, the majority of the computation time on CPU will be spent on loop control operations when there is insignificant arithmetic computation load [104].

Gaussian Smoothing is a typical example of which its efficiency suffers from control and loop overheads. By launching numerous threads to independently tackle the pixel/voxel-wise computation, much of the overheads spent on control flow can be eliminated. However, the computation bottleneck will then shift to the

memory bandwidth, which can be demanding when large-scaled thread parallelism is employed. By carefully managing and optimizing the memory transaction, a significant 190-270× computation speed-up can be achieved depending on the dimension of the input.

Vector field composition and image warping are relatively compute-intensive operations. To this end, the GPU utilizes is underlying texture filtering hardware to perform efficient interpolation. The computation speed-up brought by the optimized hardware for specialized computation, in addition to the profound speedup that brought up by thread parallelism, results in an impressive 255-294× speed-up for vector field interpolation. The similar computation required for image warping is also speed up for 311-336×. In addition to the optimized vector field interpolation operation, the application of efficient GPU-based maximum reduction by the thrust library [103] also assisted fast computation of the scaling-and-squaring method. As a result, computation of the deformation field from the log-domain velocity field have also received significant computation speed-up by 275-296×.

However, computing the update field and registration energy on GPU does not show as large computation speed-up as other GPU kernels. Computation of update field only achieved a relatively low speed-up of 48-63×. Evaluation of registration energy received a moderate speed-up of 108-151×. Such regression in terms of speed-up versus CPU can be accounted by the fact that the CPU can handle such computation more efficiently. Due to an increased computation load compared to the loop control overhead, the speed-up brought by mitigating such overhead by thread parallelism has been diminished. Furthermore, the efficiency of computing the update field on GPUs are also limited by multiple branching operations that exist in the update field computation process.

Table 4.4 Registration parameters and the iterations required, and the time required for GPU to initialize and perform the registration computation for different test images.

Image Dimension	Iterations Required	Self-compositions Required	Initialization Overhead (ms)	Computation Time (ms)
104 × 125 × 104	25	66	179	92
139 × 167 × 139	32	106	211	309
174 × 209 × 174	39	135	327	748
209 × 251 × 209	43	154	511	1479
244 × 293 × 244	47	181	787	2707
278 × 334 × 278	50	205	1126	4803
313 × 376 × 313	53	225	1617	7160

Registration parameters:
$\boldsymbol{\sigma_f = 3},\ \boldsymbol{\sigma_d = 3}, \boldsymbol{\sigma_i = 1},\ \boldsymbol{\sigma_x = 1}$;
Registration termination condition: Gradient descent, threshold: 2.5% of initial energy

Table 4.4 presents the registration parameters and the computation time required for our GPU implementation to complete the registration process. It is worth mentioning that the registration time increases not only because of the increased image dimensionality, but also because of the increased number of iterations required for the registration. As the test images are produced by up-/down-sampling from a pair of pre-deformed brain image, the resultant magnitude of the vector field representing the true deformation will also scale accordingly. As the *diffeomorphic log-demons* registration framework restricts the magnitude of update field to be <0.5 pixels per iteration, more iterations will be required for the registration upon larger deformation [55]. Furthermore, as the magnitude of the vector field increases, computing the deformation from the velocity field in the registration progress using the scaling-and-squaring method will also require more self-compositions, which can lead to higher computation time.

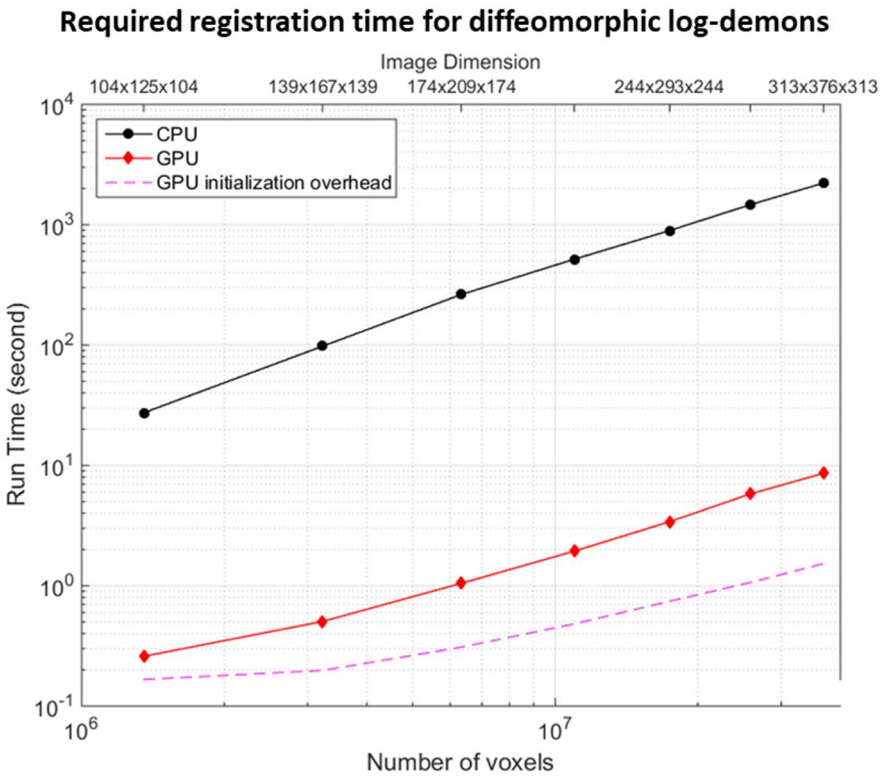

Figure 4.11 Time required for CPU (black line) and GPU with optimized kernels (red line) to complete the registration. The overhead due to memory transfer in the initialization of GPU is also included (dotted line in purple).

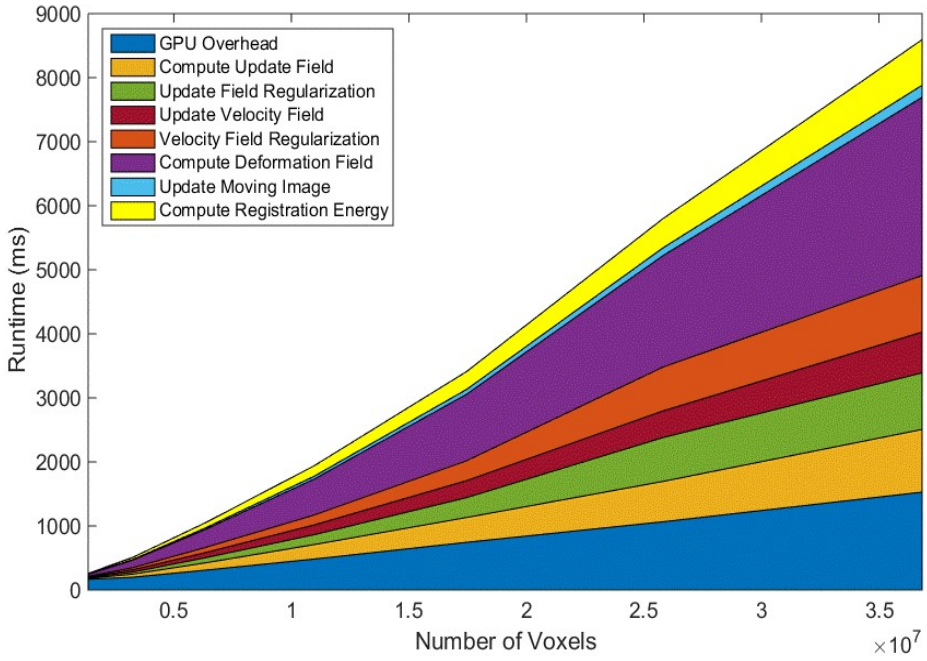

Figure 4.12 Breakdown of required computation time by the major computation steps in for *diffeomorphic log-demons*. These recorded run time are obtained by to register the testing images in 7 levels of resolution with number of voxels ranging from 1×10^6 to 3.6×10^7 voxels.

Figure 4.11 showed a vast acceleration can be achieved with appropriately optimized GPU kernels. The overall time required can be reduced by two orders of magnitude. In fact, multiresolution approaches [80] are usually adopted when *diffeomorphic log-demons* is used to register such a large image. Nonetheless, as the goal of the thesis is to compare the computation speed instead of registration performance, we intend to directly compare the run-time on CPU and optimized GPU using a single registration level. In the optimized GPU implementation of *diffeomorphic log-demons*, it is observable that the GPU initialization takes up substantial overheads, especially when the input image resolution is small. Our optimized GPU implementation is able to consistently achieve computation speedup by two orders of magnitudes, which can have to potential to complete any registrations within seconds.

Figure 4.12 presents the breakdown of time required for the 8 major computation steps essential for the registration algorithm. In contrast to the profiling report presented in *Section 3.3.2*, Gaussian regularization as well as field composition no longer take up most of the computation time in the GPU implementation, which suggests the major computation bottleneck of the *diffeomorphic log-demons* algorithm has been resolved.

Finally, **Figure 4.13** presents the reported registration energy for the first 50 iterations from both CPU and GPU implementations when registering a test image set with resolution of 174×209×174 voxels. There was no significant disparity observed between the two sets of registration energy reported, which suggests the presented GPU implementation of *diffeomorphic log-demons* is as accurate as the CPU implementation.

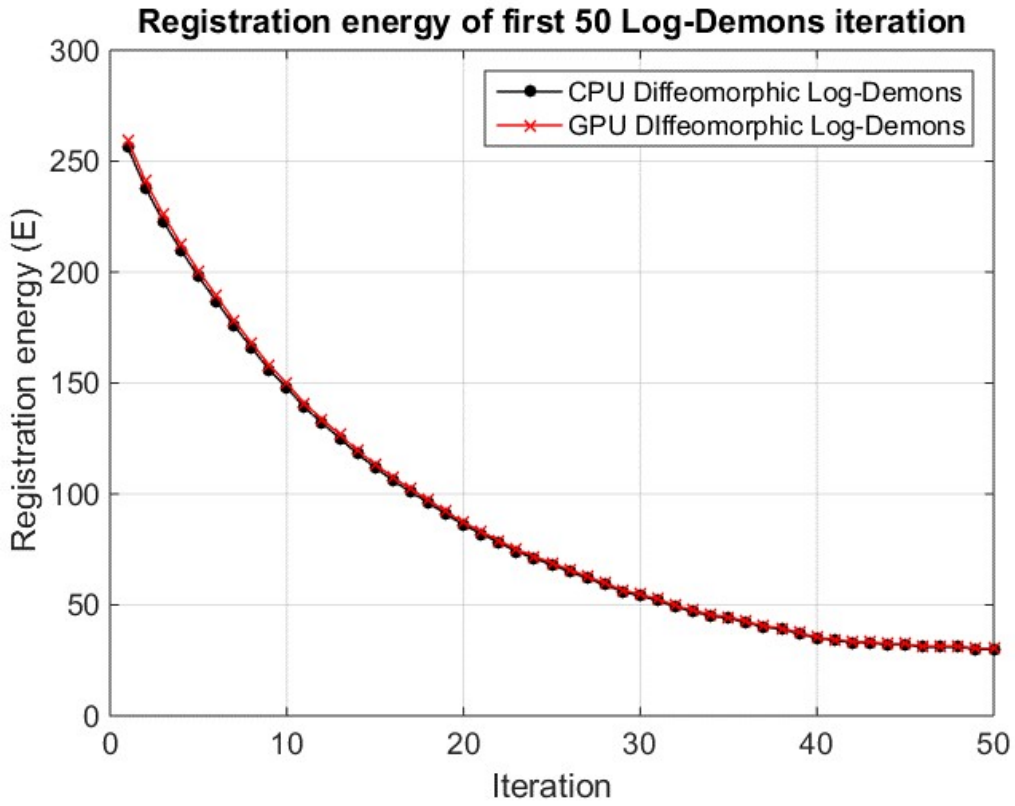

Figure 4.13 Evaluated registration energy of first 50 log-*demons* iterations from CPU and GPU implementations. No significant disparity between the evaluated energy values by CPU and GPU is found. This confirms consistent behavior among the CPU and optimized GPU implementations.

4.5 Conclusion

In this chapter, I have identified the major performance limiter of different computation sub-modules in the GPU implementation of the *diffeomorphic log-demons* algorithm. In response to the identified performance limiter, subsequent actions were deduced and presented in *Chapter 4.2*. With the bottlenecking operations being found in *Section 3.3.2*, I employed various performance-aware programming techniques to iteratively optimize and improve the computation for those computations.

For quantification of the improvement brought by the performance-aware programming, extensive testing and profiling were performed on both the optimal and sub-optimal implementations of the computation modules. The results were presented in *Chapter 4.3*. With the performance-aware optimization strategies in mind, I have implemented an optimized implementation for other computation

required in the algorithm. As a result, fully-optimized GPU implementation of the algorithm is presented in *Chapter 4.4*. An average of 216-234 times computation speed-up is achieved in the optimized implementation. In other words, the GPU is potent to complete the computation workload within seconds, which otherwise will take minutes to hours if performed by a single threaded CPU.

Chapter 5

TECHNICAL CONSIDERATIONS FOR EXTENSIVE APPLICATIONS

5.1 Open-sourced high-performance registration tool

There are many open-sourced implementations of image registration (e.g. SPM, NiftyReg, elastix and ANTS) publicly available, capable of registering medical images in 3D. However, the popularity of the *demons* algorithm is not appropriately complemented by enough open-sourced GPU support. Regarding on the advanced *demons* algorithm, there were a few attempts on implementing the *diffeomorphic log-demons* on the GPU, as presented in *Section 2.4.2*. However, those implementations are neither open sourced nor optimized. The well-known open-sourced medical image analysis toolkit, ITK, which possesses a wide variety of image registration implementations, also lacks GPU support on newer *demons* variations. To-date, the ITK toolkit only implements Thirion's original *demons* algorithm [55] on GPUs. The lack of GPU support for advance Demon's variants can lead to prolonged computation time. Such prolonged time not only frustrates related research on the same area, but also preclude adoption of the algorithm in time-critical applications.

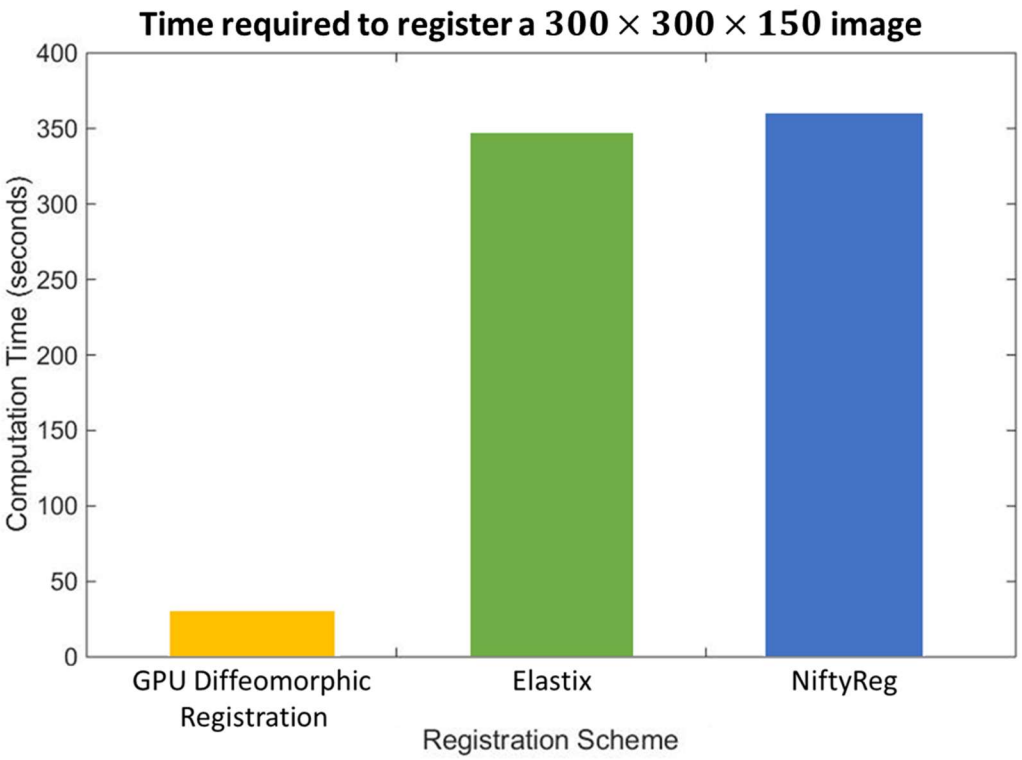

Figure 5.1 Computation time required to register an image with resolution of 300×300×150. The presented *diffeomorphic log-demons* implementation on GPU significantly outperforms the other 2 open-sourced image registration packages.

Figure 5.1 presented the run-time comparison to register an image with 13.5M voxels for different open-sourced image registration packages. To ensure fairness, all registration employed multiresolution approaches by registering the sub-sampled image in 3 levels. The presented *diffeomorphic log-demons* implementation in GPU in this work completed all 3 levels of registration within 30 seconds, which significantly outperforms other open-source image registration packages that require approximately 5 minutes to register the images.

The presented GPU implementation of the *diffeomorphic log-demons* algorithm on GPU using performance-aware techniques in this thesis will be open-sourced to the general public in the near future. With the source codes and the optimization strategies being open to the public, this work will be expected to be one of the cornerstones in the field of high-performance image registration. It is also worth noting that, given the popularity of the *demons* algorithm, there is indeed a lot of work that has been built based on the algorithm of *diffeomorphic log-demons*. For example, spherical *demons* [106], LCC-*demons* [107], adaptive *demons* [108],

and many other improved *demons* approaches [109-111]. As these works may share the same *demons* framework which consists of diffeomorphic deformation field computation, vector field regularization, and so on, the optimization approaches used and presented in this work may be able to share among these different implementations.

5.2 Limitations of GPU image registration

In the previous chapters, the essential optimization required to achieve full GPU utilization via different performance-aware programming techniques have been presented. However, it has to be noted that the GPU possesses several limitations. It is worth mentioning that these limitations are originated due to the fact that GPUs are originally designed for video frame rendering. Although it is viable to use GPU run intensity-based image registration algorithms in most cases, such limitations may cause issues when one attempt to use the GPU to register very large image datasets or require very high precisions.

5.2.1 Graphics memory consumption

As GPU specializes in rendering frame output, the processors are highly optimized for parallel processing numerous single-precision floating point operations. Our implementation of *diffeomorphic log-demons* on the GPU takes advantages of the optimized architecture to process single-precision float variables to ensure fast computation. As the computation of the *diffeomorphic log-demons* algorithm generates a number of vector fields essential for the computation, the resultant memory requirement can easily escalate beyond the hardware limit of allowable graphics memory. To illustrate the intense memory size requirement, I have listed the items used in *diffeomorphic log-demons* that can consume a considerable amount of memory in **Table 5.1**:

Table 5.1 Memory requirement for accommodating different essential variables with dimension dim for the *diffeomorphic log-demons* algorithm within the GPU memory.

Item	*Symbol*	*Representation (bytes)*	*Stored in*	*Memory required (in bytes)*
Fixed image	F	Single-precision float (4 bytes)	Global memory	$4 \times dim$
Moving image	M	Single-precision float (4 bytes)	Texture memory	$4 \times dim$
Immediate moving image	$M \circ s$	Single-precision float (4 bytes)	Global memory	$4 \times dim$
Fixed image gradient	$\nabla(F)$	Single-precision float4 (16 bytes)	Global memory	$16 \times dim$
Moving image gradient	$\nabla(M \circ s)$	Single-precision float4 (16 bytes)	Global memory	$16 \times dim$
Deformation field	s	Single-precision float4 (16 bytes)	Global memory	$16 \times dim$
Update field	u	Single-precision float4 (16 bytes)	Global memory	$16 \times dim$
Velocity field	v	Single-precision float4 (16 bytes)	Global memory	$16 \times dim$
			Texture Memory	$16 \times dim$
			Total:	$\mathbf{108 \times dim}$ ***bytes***

For any input image with size dim for registration, it is observable that the GPU will need to allocate considerable memory for the input image sets for registration. Despite the *fixed* and *moving* images only account for $(8 \times dim)$ bytes of the graphics memory utilization, the subsequent computation to register the images will require substantial memory being allocated. Apart from the vector fields that are required to be stored inside the graphics memory, the image gradient of both the fixed and immediate moving image are essential to compute for the velocity field update as well as registration energy. As a result, the memory requirement of the *diffeomorphic log-demons* algorithm for different input dataset dimensions are presented in **Figure 5.2**. At high image resolution, the memory required may exceed the typical graphics memory limit that can be provided by a GPU.

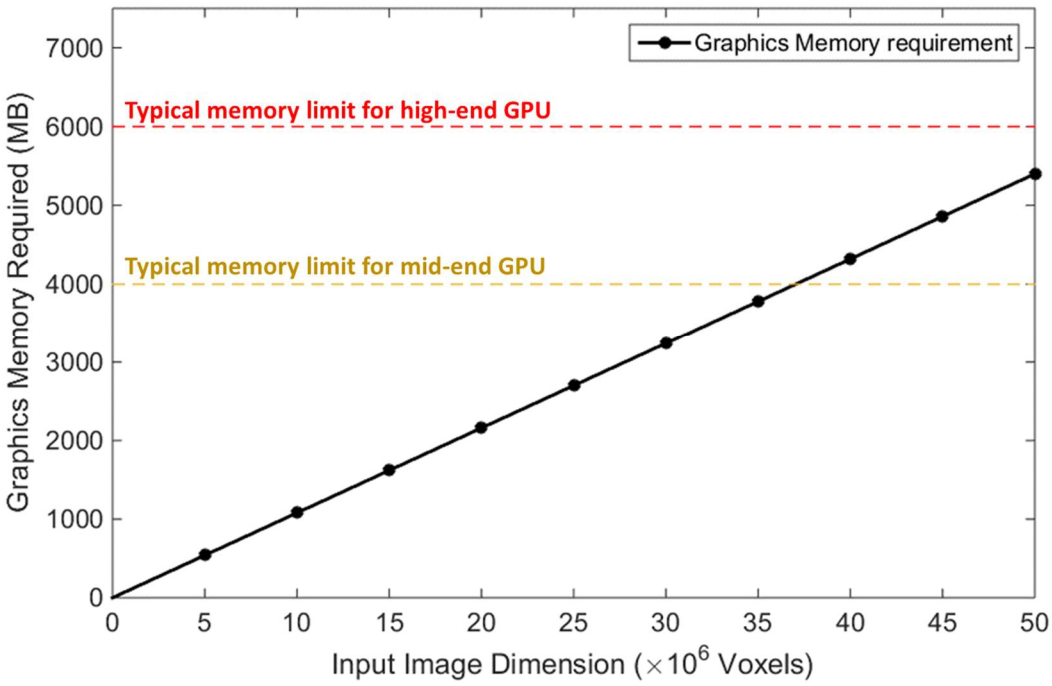

Figure 5.2 Memory size requirement on the GPU with respect to the input image dimensions. The memory size demand can exceed the allowable memory provided by mid-end GPU (e.g. NVidia GTX1050Ti), or even high-end GPUs (e.g. NVidia GTX980) when the image dimension is large.

5.2.2 Interpolation precision

The GPU's texture hardware is able to efficiently perform interpolation using its texture filtering pipeline. However, such fast texture filtering comes with a decreased precision. According to the CUDA Programming guide [105], the texture filtering is performed with the interpolant being rounded down to 9-bit fixed-point precision with 8-bit fractional values. Hence, with lower interpolant precision, the interpolation will be less precise than the usual CPU implementation.

To investigate the effect of reduced precision by the hardwired texture filtering function, **Figure 5.3** presents the results of image warping using with full float precision, as well as 9-bit interpolant precision using the texture hardware. It is observed that despite the GPU's graphics hardware uses reduced interpolant precision, the resultant absolute error between the full and reduced interpolant precision is only around 0.3%. This 0.3% error is negligible, considering the intensity range of most medical image is often limited to 0-4095 (12-bit precision). Even for the vector field exponentiation operations which require 4-5 self-composition, the absolute error will not exceed 2%, which is acceptable for the

demons algorithm considering the registration algorithm will to converge unconditionally.

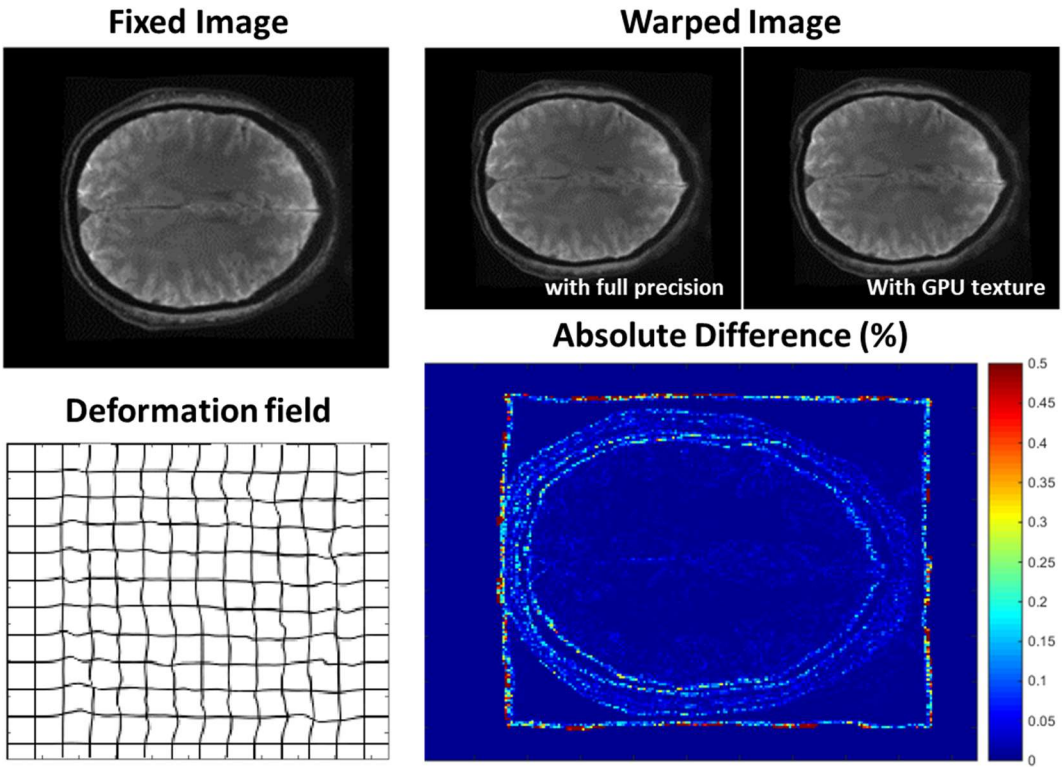

Figure 5.3 Fixed image warped by deformation field using full and reduced interpolant precision. Although hardwired texture filtering uses reduced interpolant precision, the resultant difference in terms of intensity is negligible (typ. 0.25%).

5.3 Potential applications

This presented GPU implementation of *diffeomorphic log-demons* achieved significant computation speed-up compared to the CPU counterparts. With this optimized GPU registration being able to accelerate the registration process by over two hundred times, this computation enhancement is unprecedented. As a result, whole registration time required can be reduced from minutes to seconds. With the promising computation speed-up provided, this presented GPU registration tool opens up countless research and applicational opportunities. This section discusses the potential application of the presented GPU registration tool.

5.3.1 Algorithmic improvements by meta-heuristic search of parameters

One of the challenges in intensity-based image registration is the inability to determine the optimal registration parameters. Different input image sets not only consist of different image features, but they may also have different data ranges and dimensions. These variations in parameters can utterly alter the optimal registration parameters. To date, there are several studies focusing on optimization-based image registration. Klein *et al.* [112] looked into several optimization algorithms including gradient descent, quasi-Newton, and evolution strategies to look for the best parameters for registering CT images of the heart. Also, meta-heuristic based methods were also proposed, in a bid to search for a set of optimal image registration parameters. Later, Zheng *et al.* [114] have utilized and improved the particle swarm optimization (PSO) algorithm to register multimodal images. The clinical impact of such PSO-based search for image registration has also been presented in [115], exemplified by performing rigid 3D registration for medical images. However, much of these works only focus on registrations with relatively simple algorithms. Recently, Cuckoo Search algorithm, a variant of PSO, was employed to 2D *diffeomorphic log-demons* registration in an attempt to find out the optimal registration parameters [116]. However, this Cuckoo Search algorithm will require >11 hours to converge. Due to extensive computation demand of the *demons* algorithm, employing meta-heuristic based image registration for 3D *diffeomorphic log-demons* using CPU may not be practical.

With the GPU-enabled *diffeomorphic log-demons* tool being open-sourced, the immense computation speed-up brought by the GPU tool can facilitate any optimization-based method on searching the optimal parameters. With the presented registration tool being able to accelerate the registration progress by approximately 200 times, these PSO-like search algorithms can have their speed and accuracy improved. Above all, the whole optimization can be accelerated due to the decrease in registration time. As the result of the decreased registration time, the cost of spawning a particle in PSO can be decreased. Therefore, more particles can be spawn in each PSO iteration, facilitating a more accurate search for the parameters.

5.3.2 Clinical applications

Having a rapid non-rigid image registration is beneficial to many interventional procedures. For example, the MRI-guided EP process aiming to treat cardiac arrhythmia by electrically isolating specific cardiac tissue can be benefited by rapidly realigning the pre-op roadmap. Besides, intensity-modulated radiotherapy is also another potential clinical application which can be benefited by fast nonlinear re-alignment during radiation dosage evaluation.

5.3.2.1 Accelerated mapping of electro-anatomical map

In the EP procedure, the construction of the cardiac roadmap heavily relies on EA map acquired using a mapping catheter. Such electro-anatomical map reveals the cardiac electrophysiologic activity throughout the cardiac cycle [117]. Any tissue with abnormal electrical signals can be revealed. Thus, such cardiac-phase-dependent EA map can assist surgeons to pinpoint the cardiac tissue that needs to be isolated, thereby improving treatment accuracy and effectiveness.

The major technical challenge resides in the re-alignment of the intraoperatively acquired EA mapping data on the preoperatively constructed cardiac image. To construct a cardiac roadmap, these EA mapping data gathered by numerous contact points (**Figure 5.4a**) needed to be promptly registered back to the pre-op cardiac model (**Figure 5.4b**). However, the distribution of these EA contact points may not align with the pre-op model, due to the physiological movement of the cardiac chambers, as well as respiratory motion.

Non-rigid image registration can be used to find out the spatial misalignment between the images. Once the misalignment is found, the EA data can be re-aligned back to the pre-op cardiac atlas (**Figure 5.4c**). However, such registration needs to be repeated for numerous times to register every single contact points back to the anatomically correct pre-op model. As such, this repeated registration demands considerable computation. The currently available clinical system can only provide an approximate alignment using rigid image registration [121], thus it relies heavily on the surgeons' experience to estimate the correspondence between the EA map and the pre-op atlas.

Therefore, having a fast, reliable non-rigid image registration scheme can facilitate the EA mapping process. By promptly realigning the EA mapping data back to the anatomically-correct cardiac atlas, a reliable EA map can be produced. With our implementation of *diffeomorphic log-demons* being capable to accelerate the registration computation by ~220 times, it is possible for the registration to be completed within a couple of seconds for accurate visualization.

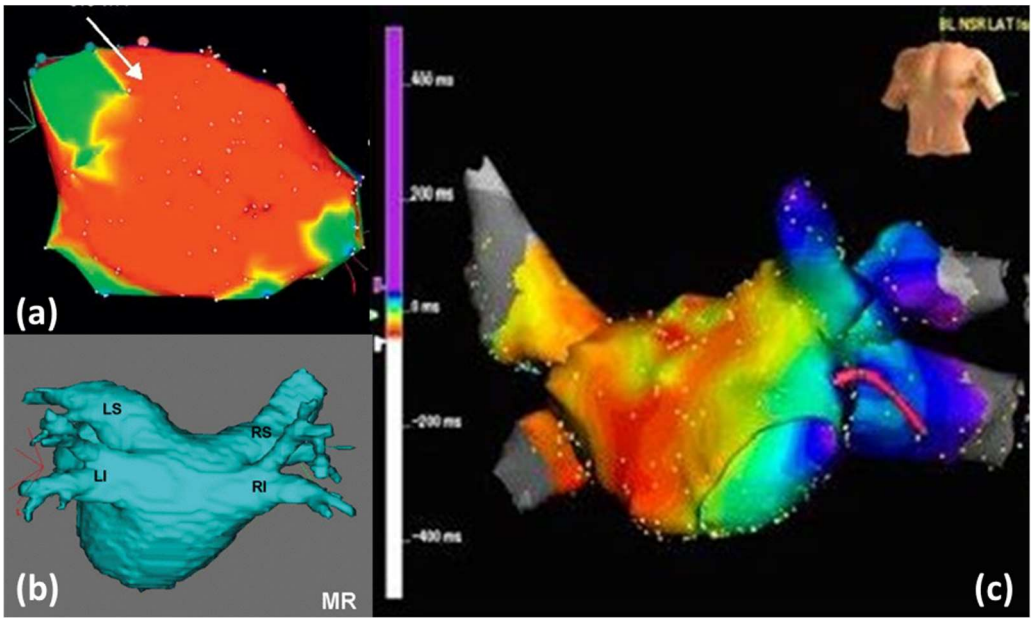

Figure 5.4 **(a)** EA map of the left atrium obtained prior to the EP process. The mapping gathered by EA contact points may not be anatomically correct. **(b)** Anatomically correct left atrium model obtained by segmentation of an MR image. **(c)** Image registration used to realign the EA mapping data back to the anatomically correct left atrium model. Images retrieved from [118-120].

5.3.2.2 Intra-op registration of EP target

iMRI possesses the ability to visualize scars and edema created by during the EP procedure. However, due to large-scale tissue deformation of the rapidly moving cardiac chamber, the left atrium on the iMRI image may not be aligned with the pre-op image, as well as the EA roadmap. Similar to the EA mapping procedure, the major challenge resides in having a proper realignment between the lesion registered back to the cardiac roadmap [122]. Due to the rapid pumping motion of the heart, the iMRI image can exhibit misalignment between cardiac cycles, even if the image is gated. Therefore, it is essential to have prior knowledge of the tissue deformation between the pre-op and the intra-op images. In this light,

non-rigid registration can be used to realign the lesion on the intra-op images back to the pre-op roadmap augmented with the EA mapping.

Figure 5.5 demonstrates the workflow of integrating non-rigid image registration on the cardiac EP procedure. Non-rigid image registration is essential to resolve the large misalignment between the pre-op cardiac roadmap and the iMRI image. Once the misalignment is found, the lesion spots on the intra-op image can be accurately realigned back on the pre-op model. With the presented non-rigid image registration tool in this thesis, it is believed that such image realignment can be performed in fast and frequent manner. As such, the number of iMRI scanning can be further increased to ensure favorable post-operative outcome.

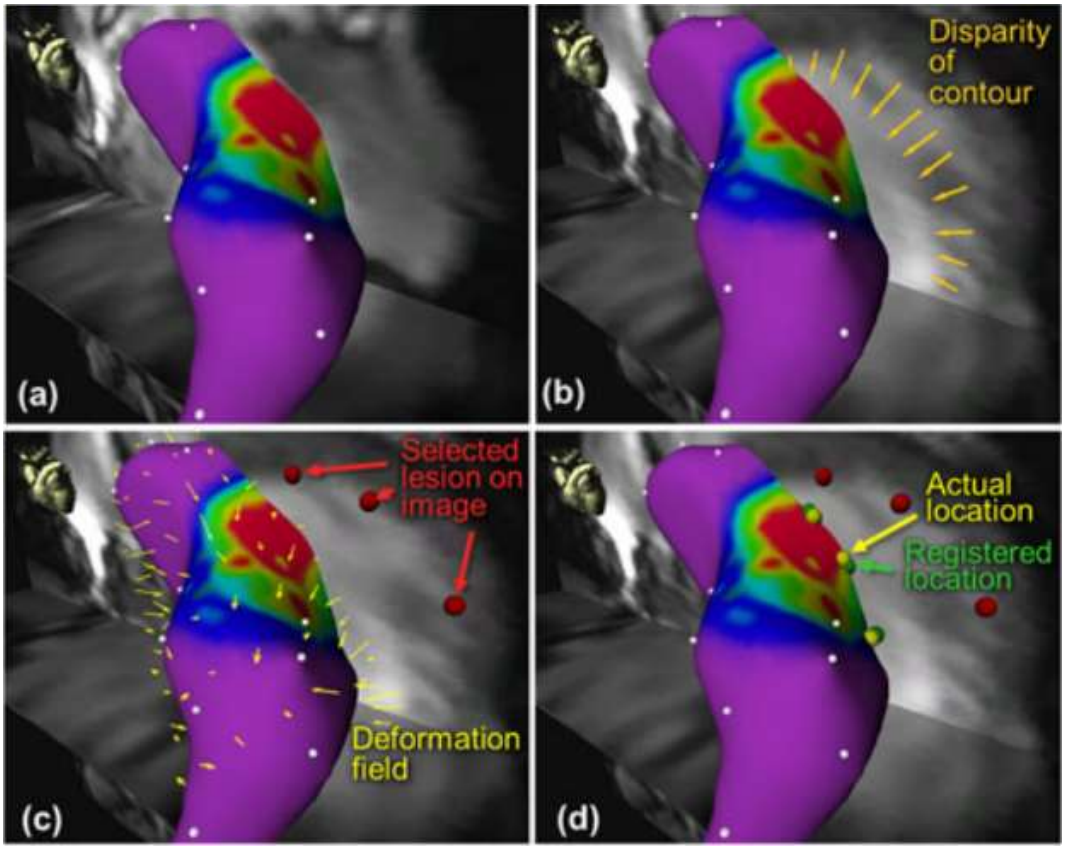

Figure 5.5 **(a)** EP roadmap of left atrium rendered based on pre-op MR images. **(b)** Significant mismatch indicated by orange arrows between the roadmap and intra-op images. **(c)** Ablation landmarks selected on a slice of 2-D intra-op MR images. Yellow arrows illustrate the deformation. **(d)** Ablation landmarks realigned appropriately on the 3-D roadmap based on the deformation field.

5.3.2.3 Towards extensive use of IGRT

Radiotherapy is one of the areas that extensively adopt intra-op imaging. However, current use of intra-op imaging to adjust radiotherapy treatment plan is limited. Instead, the onboard CBCT imagers are mostly used in patient localization and position calibration [123]. As presented in **Figure 5.6**, tissue deformation can be observed when the patient undergoes the month-long radiotherapy treatment, which can lead to discrepancies between the treatment plan and the actual anatomy.

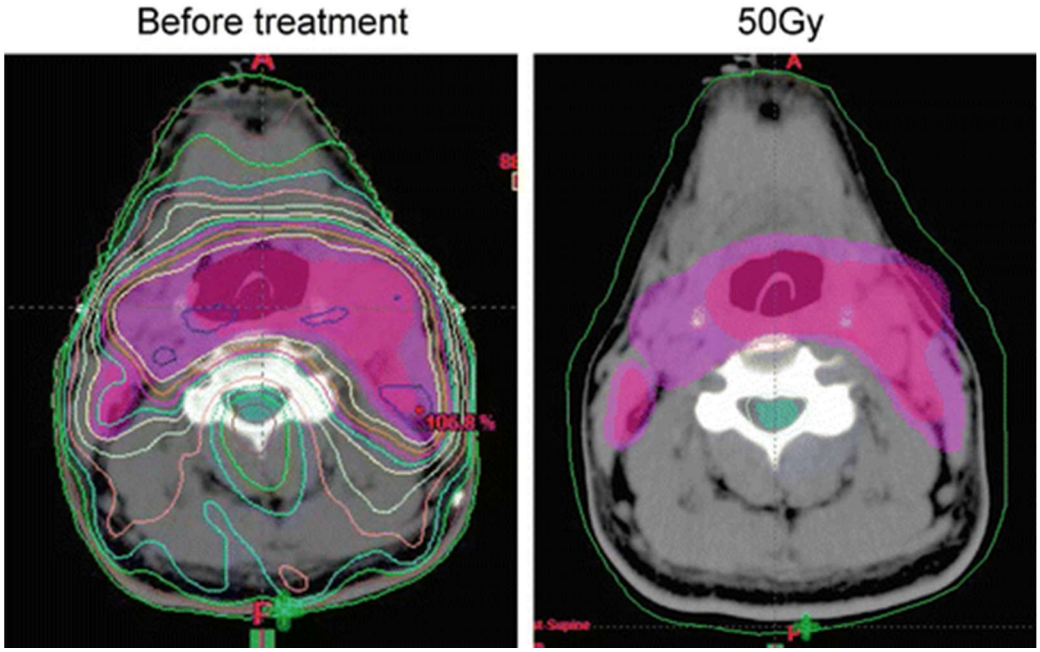

Figure 5.6 CT image showing significant tissue deformation at the thorax after receiving 50Gy of radiation in radiotherapy. This large-scale deformation may lead to damage to critical organs if not compensated. Image retrieved from [124].

To mitigate any unintended damage to the healthy tissues or critical organs due to tissue shrinkage and deformation throughout the radiotherapy treatment, IGRT possesses the ability to perioperatively localize the treatment target. Non-rigid image registration is the key and to quantify the misalignments between the pre-op image and the intra-op CBCT images. Such knowledge on the misalignment is for essential treatment plan adjustment. However, most non-rigid image registration schemes are slow, and can be bottlenecking. As the image registration require perioperatively acquired CBCT image, the required registration time will occupy significant timeslots of the radiotherapy machine, thus, precluding any extensive use of IGRT [125]. As such timeslots are limited, a common practice is

to employ IGRT only to the high risks patients. For example, only the patients suffering from recurrent cancer [126], or patients with the tumors located near critical organs [127]. With the presented work in this thesis which is capable to accelerate the registration process, it is expected the operations of IGRT can be streamlined, resulting in a more widespread and extensive use of IGRT.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1.1 Achievement of this thesis

Non-rigid image registration has long been validated as an effective strategy to co-register misaligned image sets. Such realignment is decisive towards safer and precise image-guided interventions, but suffers from prolonged registration time. The recent advancement in high-performance computation devices, enabled fast computation for different applications. However, to effectively utilize the computation device for image registration, one has to obtain a solid knowledge of both the algorithm and the hardware microarchitecture. This thesis attempted to bridge the technological gap between the registration algorithm and its potential application in the clinical scenario.

In this thesis, I have introduced several performance-aware techniques on GPU which is essential to achieve the near-real-time image registration, which will also be applicable to numerous other time-critical applications. By thoroughly analyzing the algorithm, it is found that most of the operations in *diffeomorphic log-demons* are not bottlenecked by computation, contrary to the popular belief. Instead, the major bottleneck resides in the memory latency during data transactions. By appropriately resolving the memory bottleneck, we confirm the ability achieve impressive acceleration of the intensity-based image registration process with a GPU. The achievements are summarized below:

In **Chapter 2**, an overview of current intra-op imaging technique, as well as the clinical demand for image-guided intervention is highlighted. We have pointed out the demands of fast non-rigid image registration, as well as the necessity of using high-performance application accelerators for time-critical applications.

In **Chapter 3**, the basis of performance-aware programming on GPU have been introduced. Besides, testing and profiling have been conducted on the *diffeomorphic log-demons*. I have also presented the underlying computation bottleneck operations, as well as proposed the essential performance-aware programming techniques to work around with such bottleneck.

The optimized implementations of these bottlenecking operations are extensively tested and analyzed in **Chapter 4**. Following the in-depth analysis of the computation demand and improvements, I have presented a working, optimized GPU implementation of *diffeomorphic log-demons* which will be open-sourced in the near future.

Finally, a brief comparison between our open-sourced GPU *diffeomorphic log-demons* registration tool and other open-source image registration toolkits was conducted earlier in **Chapter 5**. Also, I have briefly discussed the limitation of using GPU in intensity-based image registration, as well as the potential applications of the presented high-performance registration implementation on GPU.

6.1.2 Ongoing research and future work

The *diffeomorphic log-demons* algorithm possesses a lot of potential in terms of translational research, due to its ability to reliably realign mismatched images. However, the high computation requirement precludes the usage of the algorithm in a lot of time-critical applications. With significant computation speed-up being achieved in the GPU implementation in as presented in this thesis, it is expected that a lot of research opportunities can be opened up. For example, integrating the registration into intra-op navigation systems can undoubtedly increase surgical safety and accuracy. Ongoing research as an extension of this work includes the integration of the registration framework onto the various surgical robotic systems enabled with intra-op scans. For example, the MR-safe robotic system [128] capable of providing intra-op MRI guidance, or the hydraulic driving robot capable of performing MRI-guided stereotactic neurosurgery [129].

The work presented in this thesis further confirmed the potent ability to accelerate various intensity-based image registration schemes using GPU. Future work includes the integration of this optimized GPU implementation into the ITK framework. Also, this work can be extended by investigating the possibility of using GPU to accelerate the computation of another variant of the *demons* algorithm, including LCC-*Demons* [107], Spectral *demons* [74], and other approaches.

REFERENCE

[1] D. W. Roberts, J. W. Strohbehn, J. F. Hatch, W. Murray, and H. Kettenberger, "A frameless stereotaxic integration of computerized tomographic imaging and the operating microscope," *Journal of neurosurgery,* vol. 65, no. 4, pp. 545-549, 1986.

[2] A. Polo, C. Salembier, J. Venselaar, and P. Hoskin, "Review of intraoperative imaging and planning techniques in permanent seed prostate brachytherapy," *Radiotherapy and Oncology,* vol. 94, no. 1, pp. 12-23, 2010.

[3] S. L. Troyan, V. Kianzad, S. L. Gibbs-Strauss, S. Gioux, A. Matsui, R. Oketokoun, L. Ngo, A. Khamene, F. Azar, and J. V. Frangioni, "The FLARE™ intraoperative near-infrared fluorescence imaging system: a first-in-human clinical trial in breast cancer sentinel lymph node mapping," *Annals of surgical oncology,* vol. 16, no. 10, pp. 2943-2952, 2009.

[4] M. I. Miga, T. K. Sinha, D. M. Cash, R. L. Galloway, and R. J. Weil, "Cortical surface registration for image-guided neurosurgery using laser-range scanning," *IEEE Transactions on Medical Imaging,* vol. 22, no. 8, pp. 973-985, 2003.

[5] S. Nakajima, H. Atsumi, R. Kikinis, T. M. Moriarty, D. C. Metcalf, F. A. Jolesz, and P. M. Black, "Use of cortical surface vessel registration for image-guided neurosurgery," *Neurosurgery,* vol. 40, no. 6, pp. 1201-1210, 1997.

[6] R. Hammerstingl, A. Huppertz, J. Breuer, T. Balzer, A. Blakeborough, R. Carter, L. C. Fusté, G. Heinz-Peer, W. Judmaier, and M. Laniado, "Diagnostic efficacy of gadoxetic acid (Primovist)-enhanced MRI and spiral CT for a therapeutic strategy: comparison with intraoperative and histopathologic findings in focal liver lesions," *European radiology,* vol. 18, no. 3, pp. 457-467, 2008.

[7] M. Knauth, C. R. Wirtz, V. M. Tronnier, N. Aras, S. Kunze, and K. Sartor, "Intraoperative MR imaging increases the extent of tumor resection in patients with high-grade gliomas," *American journal of neuroradiology,* vol. 20, no. 9, pp. 1642-1646, 1999.

[8] R. Gennari, V. Galimberti, C. De Cicco, S. Zurrida, F. Zerwes, F. Pigatto, A. Luini, G. Paganelli, and U. Veronesi, "Use of technetium-99m–labeled colloid albumin for preoperative and intraoperative localization of nonpalpable breast lesions," *Journal of the American College of Surgeons,* vol. 190, no. 6, pp. 692-698, 2000.

[9] A. Nabavi, P. M. Black, D. T. Gering, C.-F. Westin, V. Mehta, R. S. Pergolizzi Jr, M. Ferrant, S. K. Warfield, N. Hata, and R. B. Schwartz, "Serial intraoperative magnetic resonance imaging of brain shift," *Neurosurgery,* vol. 48, no. 4, pp. 787-798, 2001.

[10] U. Veronesi, R. Orecchia, A. Luini, G. Gatti, M. Intra, S. Zurrida, G. Ivaldi, G. Tosi, M. Ciocca, and A. Tosoni, "A preliminary report of intraoperative radiotherapy (IORT) in limited-stage breast cancers that are conservatively treated," *European journal of cancer,* vol. 37, no. 17, pp. 2178-2183, 2001.

[11] R. M. Comeau, A. F. Sadikot, A. Fenster, and T. M. Peters, "Intraoperative ultrasound for guidance and tissue shift correction in image-guided neurosurgery," *Medical physics,* vol. 27, no. 4, pp. 787-800, 2000.

[12] R. Schulze, U. Heil, D. Groβ, D. Bruellmann, E. Dranischnikow, U. Schwanecke, and E. Schoemer, "Artefacts in CBCT: a review," *Dentomaxillofacial Radiology,* vol. 40, no. 5, pp. 265-273, 2011.

[13] I. D. Gelalis, N. K. Paschos, E. E. Pakos, A. N. Politis, C. M. Arnaoutoglou, A. C. Karageorgos, A. Ploumis, and T. A. Xenakis, "Accuracy of pedicle screw placement: a systematic review of prospective in vivo studies comparing free hand, fluoroscopy guidance and navigation techniques," *European Spine Journal,* vol. 21, no. 2, pp. 247-255, 2012.

[14] M. Goitein, M. Abrams, D. Rowell, H. Pollari, and J. Wiles, "Multi-dimensional treatment planning: II. Beam's eye-view, back projection, and projection through CT sections," *International Journal of Radiation Oncology* Biology* Physics,* vol. 9, no. 6, pp. 789-797, 1983.

[15] T. Grünheid, J. R. K. Schieck, B. T. Pliska, M. Ahmad, and B. E. Larson, "Dosimetry of a cone-beam computed tomography machine compared with a digital x-ray machine in orthodontic imaging," *American Journal of Orthodontics and Dentofacial Orthopedics,* vol. 141, no. 4, pp. 436-443, 2012.

[16] J. Kottoor, N. Velmurugan, R. Sudha, and S. Hemamalathi, "Maxillary first molar with seven root canals diagnosed with cone-beam computed tomography scanning: a case report," *Journal of endodontics,* vol. 36, no. 5, pp. 915-921, 2010.

[17] F. Haiter-Neto, A. Wenzel, and E. Gotfredsen, "Diagnostic accuracy of cone beam computed tomography scans compared with intraoral image modalities for detection of caries lesions," *Dentomaxillofacial Radiology,* 2014.

[18] M. S. Mizel, N. D. Steinmetz, and E. Trepman, "Detection of wooden foreign bodies in muscle tissue: experimental comparison of computed

tomography, magnetic resonance imaging, and ultrasonography," *Foot & ankle international,* vol. 15, no. 8, pp. 437-443, 1994.

[19] R. Damadian, "Tumor detection by nuclear magnetic resonance," 1971.

[20] S. Fratz, M. Hauser, F. M. Bengel, A. Hager, H. Kaemmerer, M. Schwaiger, J. Hess, and H. C. Stern, "Myocardial scars determined by delayed-enhancement magnetic resonance imaging and positron emission tomography are not common in right ventricles with systemic function in long-term follow up," *Heart,* vol. 92, no. 11, pp. 1673-1677, 2006.

[21] J. M. Mislow, A. J. Golby, and P. M. Black, "Origins of intraoperative MRI," *Magnetic resonance imaging clinics of North America,* vol. 18, no. 1, pp. 1-10, 2010.

[22] J. F. Schenck, F. A. Jolesz, P. B. Roemer, H. E. Cline, W. E. Lorensen, R. Kikinis, S. G. Silverman, C. J. Hardy, W. D. Barber, and E. T. Laskaris, "Superconducting open-configuration MR imaging system for image-guided therapy," *Radiology,* vol. 195, no. 3, pp. 805-814, 1995.

[23] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic resonance in medicine,* vol. 58, no. 6, pp. 1182-1195, 2007.

[24] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly, "Compressed sensing MRI," *IEEE signal processing magazine,* vol. 25, no. 2, pp. 72-82, 2008.

[25] R. Ranjan, E. G. Kholmovski, J. Blauer, S. Vijayakumar, N. A. Volland, M. E. Salama, D. L. Parker, R. MacLeod, and N. F. Marrouche, "Identification and Acute Targeting of Gaps in Atrial Ablation Lesion Sets Using a Real-Time Magnetic Resonance Imaging SystemClinical Perspective," *Circulation: Arrhythmia and Electrophysiology,* vol. 5, no. 6, pp. 1130-1135, 2012.

[26] K.-W. Kwok, K.-H. Lee, Y. Chen, W. Wang, Y. Hu, G. C. Chow, H. S. Zhang, W. G. Stevenson, R. Y. Kwong, and W. Luk, "Interfacing fast multi-phase cardiac image registration with MRI-based catheter tracking for MRI-guided electrophysiological ablative procedures," *Circulation,* vol. 130, no. Suppl 2, pp. A18568-A18568, 2014.

[27] Y. Chen, W. Wang, E. J. Schmidt, K.-W. Kwok, A. N. Viswanathan, R. Cormack, and Z. T. H. Tse, "Design and Fabrication of MR-Tracked Metallic Stylet for Gynecologic Brachytherapy," *IEEE/ASME Transactions on Mechatronics,* vol. 21, no. 2, pp. 956-962, 2016.

[28] G. Minchev, G. Kronreif, M. Martínez-Moreno, C. Dorfer, A. Micko, A. Mert, B. Kiesel, G. Widhalm, E. Knosp, and S. Wolfsberger, "A novel miniature robotic guidance device for stereotactic neurosurgical interventions: preliminary experience with the iSYS1 robot," *Journal of Neurosurgery,* pp. 1-12, 2016.

[29] J. González-Martínez, J. Bulacio, S. Thompson, J. Gale, S. Smithason, I. Najm, and W. Bingaman, "Technique, results, and complications related to robot-assisted stereoelectroencephalography," *Neurosurgery,* vol. 78, no. 2, pp. 169-180, 2016.

[30] P. A. Starr, A. J. Martin, J. L. Ostrem, P. Talke, N. Levesque, and P. S. Larson, "Subthalamic nucleus deep brain stimulator placement using high-field interventional magnetic resonance imaging and a skull-mounted aiming device: technique and application accuracy," *Journal of neurosurgery,* vol. 112, no. 3, p. 479, 2010.

[31] G. Widmann, R. Stoffner, M. Sieb, and R. Bale, "Target registration and target positioning errors in computer-assisted neurosurgery: proposal for a standardized reporting of error assessment," *The International Journal of Medical Robotics and Computer Assisted Surgery,* vol. 5, no. 4, pp. 355-365, 2009.

[32] N. Archip, O. Clatz, S. Whalen, D. Kacher, A. Fedorov, A. Kot, N. Chrisochoides, F. Jolesz, A. Golby, and P. M. Black, "Non-rigid alignment of pre-operative MRI, fMRI, and DT-MRI with intra-operative MRI for enhanced visualization and navigation in image-guided neurosurgery," *Neuroimage,* vol. 35, no. 2, pp. 609-624, 2007.

[33] C. R. Maurer Jr, D. L. Hill, R. J. Maciunas, J. A. Barwise, J. M. Fitzpatrick, and M. Y. Wang, "Measurement of intraoperative brain surface deformation under a craniotomy," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 1998, pp. 51-62: Springer.

[34] J. Wadley, N. Kitchen, and D. Thomas, "Image-guided neurosurgery," *Hospital medicine (London, England: 1998),* vol. 60, no. 1, pp. 34-38, 1999.

[35] P. Francesco, S. Luigi, M. Alberto, and D. Francesco, *Intraoperative ultrasound (ious) in neurosurgery : from standard b-mode to elastosonography*. New York, NY: Springer Berlin Heidelberg, 2016, p. pages cm.

[36] E. Kholmovski, R. Ranjan, J. Silvernagel, J. Blauer, and N. Marrouche, "Assessment of Acute Cryo and RF Ablation Lesions by Non-contrast and Contrast Enhanced MRI Techniques: Similarities and Differences," ed: Am Heart Assoc, 2014.

[37] C. Pappone, S. Rosanio, G. Oreto, M. Tocchi, F. Gugliotta, G. Vicedomini, A. Salvati, C. Dicandia, P. Mazzone, and V. Santinelli, "Circumferential radiofrequency ablation of pulmonary vein ostia a new anatomic approach for curing atrial fibrillation," *Circulation,* vol. 102, no. 21, pp. 2619-2628, 2000.

[38] J. L. Duerk, J. S. Lewin, M. Wendt, and C. Petersilge, "Invited. Remember true FISP? a high SNR, near 1-second imaging method for T2-like contrast in interventional MRI at. 2 T," *Journal of Magnetic Resonance Imaging,* vol. 8, no. 1, pp. 203-208, 1998.

[39] C. Lu, S. Chelikani, X. Papademetris, J. P. Knisely, M. F. Milosevic, Z. Chen, D. A. Jaffray, L. H. Staib, and J. S. Duncan, "An integrated approach to segmentation and nonrigid registration for application in image-guided pelvic radiotherapy," *Medical Image Analysis,* vol. 15, no. 5, pp. 772-785, 2011.

[40] M. Guckenberger, J. Meyer, J. Wilbert, K. Baier, O. Sauer, and M. Flentje, "Precision of image-guided radiotherapy (IGRT) in six degrees of freedom and limitations in clinical practice," *Strahlentherapie und Onkologie,* vol. 183, no. 6, pp. 307-313, 2007.

[41] M. Foskey, B. Davis, L. Goyal, S. Chang, E. Chaney, N. Strehl, S. Tomei, J. Rosenman, and S. Joshi, "Large deformation three-dimensional image registration in image-guided radiation therapy," *Physics in Medicine & Biology,* vol. 50, no. 24, p. 5869, 2005.

[42] S. Hassfeld, J. Zöller, F. K. Albert, C. R. Wirtz, M. Knauth, and J. Mühling, "Preoperative planning and intraoperative navigation in skull base surgery," *Journal of cranio-maxillo-facial surgery,* vol. 26, no. 4, pp. 220-225, 1998.

[43] D. T. Gering, A. Nabavi, R. Kikinis, W. E. L. Grimson, N. Hata, P. Everett, F. Jolesz, and W. M. Wells, "An integrated visualization system for surgical planning and guidance using image fusion and interventional imaging," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 1999, pp. 809-819: Springer.

[44] A. Shoamanesh, C. Kwok, and O. Benavente, "Cerebral microbleeds: histopathological correlation of neuroimaging," *Cerebrovascular Diseases,* vol. 32, no. 6, pp. 528-534, 2011.

[45] J. R. Reichenbach, R. Venkatesan, D. A. Yablonskiy, M. R. Thompson, S. Lai, and E. M. Haacke, "Theory and application of static field inhomogeneity effects in gradient-echo imaging," *Journal of Magnetic Resonance Imaging,* vol. 7, no. 2, pp. 266-279, 1997.

[46] D. L. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in medicine and biology,* vol. 46, no. 3, p. R1, 2001.

[47] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing,* vol. 21, no. 11, pp. 977-1000, 2003.

[48] A. Vasileisky, B. Zhukov, and M. Berger, "Automated image coregistration based on linear feature recognition," in *Proceedings of the Second Conference Fusion of Earth Data, Sophia Antipolis, France*, 1998, pp. 59-66.

[49] B. Manjunath, C. Shekhar, and R. Chellappa, "A new approach to image feature detection with applications," *Pattern Recognition,* vol. 29, no. 4, pp. 627-640, 1996.

[50] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence,* no. 6, pp. 679-698, 1986.

[51] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern recognition,* vol. 26, no. 9, pp. 1277-1294, 1993.

[52] G. Stockman, S. Kopstein, and S. Benett, "Matching images to models for registration and object detection via clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* no. 3, pp. 229-241, 1982.

[53] W. Liu and E. Ribeiro, "A survey on image-based continuum-body motion estimation," *Image and Vision Computing,* vol. 29, no. 8, pp. 509-523, 2011.

[54] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, 1992, pp. 586-606: International Society for Optics and Photonics.

[55] 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, 1998.

[56] J.-P. Thirion, "Fast non-rigid matching of 3D medical images," INRIA, 1995.

[57] H. Wang, L. Dong, J. O'Daniel, R. Mohan, A. S. Garden, K. K. Ang, D. A. Kuban, M. Bonnen, J. Y. Chang, and R. Cheung, "Validation of an accelerated '*demons*' algorithm for deformable image registration in radiation therapy," *Physics in medicine and biology,* vol. 50, no. 12, p. 2887, 2005.

[58] P. Rogelj and S. Kovačič, "Symmetric image registration," *Medical image analysis,* vol. 10, no. 3, pp. 484-493, 2006.

[59] X. Pennec, P. Cachier, and N. Ayache, "Understanding the "demon's algorithm": 3D non-rigid registration by gradient descent," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 1999, pp. 597-605: Springer.

[60] D. Yang, H. Li, D. A. Low, J. O. Deasy, and I. El Naqa, "A fast inverse consistent deformable image registration method based on symmetric optical flow computation," *Physics in medicine and biology,* vol. 53, no. 21, p. 6143, 2008.

[61] T. Vercauteren, X. Pennec, E. Malis, A. Perchant, and N. Ayache, "Insight into efficient image registration techniques and the *demons* algorithm," in *Biennial International Conference on Information Processing in Medical Imaging*, 2007, pp. 495-506: Springer.

[62] S. Benhimane and E. Malis, "Real-time image-based tracking of planes using efficient second-order minimization," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 2004, vol. 1, pp. 943-948: IEEE.

[63] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 2004, vol. 2, pp. 1843-1848: IEEE.

[64] E. Haber and J. Modersitzki, "Numerical methods for image registration," 2004.

[65] P. Cachier, E. Bardinet, D. Dormont, X. Pennec, and N. Ayache, "Iconic feature based nonrigid registration: the PASHA algorithm," *Computer vision and image understanding,* vol. 89, no. 2, pp. 272-298, 2003.

[66] J. Ashburner, "A fast diffeomorphic image registration algorithm," *Neuroimage,* vol. 38, no. 1, pp. 95-113, 2007.

[67] G. E. Christensen, R. D. Rabbitt, M. I. Miller, S. C. Joshi, U. Grenander, T. A. Coogan, and D. C. Van Essen, "Topological properties of smooth anatomic maps," 1995.

[68] V. Arsigny, O. Commowick, X. Pennec, and N. Ayache, "A log-euclidean framework for statistics on diffeomorphisms," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2006, pp. 924-931: Springer.

[69] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Diffeomorphic *demons*: Efficient non-parametric image registration," *Neuroimage,* vol. 45, no. 1, pp. S61-S72, 2009.

[70] M. Hernandez, M. N. Bossa, and S. Olmos, "Registration of anatomical images using paths of diffeomorphisms parameterized with stationary vector field flows," *International Journal of Computer Vision,* vol. 85, no. 3, pp. 291-306, 2009.

[71] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Symmetric log-domain diffeomorphic registration: A *demons*-based approach," in *International conference on medical image computing and computer-assisted intervention*, 2008, pp. 754-761: Springer.

[72] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, "Non-parametric diffeomorphic image registration with the *demons* algorithm," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2007, pp. 319-326: Springer.

[73] A. Klein, J. Andersson, B. A. Ardekani, J. Ashburner, B. Avants, M.-C. Chiang, G. E. Christensen, D. L. Collins, J. Gee, and P. Hellier, "Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration," *Neuroimage,* vol. 46, no. 3, pp. 786-802, 2009.

[74] H. Lombaert, L. Grady, X. Pennec, N. Ayache, and F. Cheriet, "Spectral log-*demons*: diffeomorphic image registration with very large deformations," *International journal of computer vision,* vol. 107, no. 3, pp. 254-271, 2014.

[75] S. Oguro, J. Tokuda, H. Elhawary, S. Haker, R. Kikinis, C. Tempany, and N. Hata, "MRI signal intensity based B-Spline nonrigid registration for pre- and intraoperative imaging during prostate brachytherapy," *Journal of magnetic resonance imaging,* vol. 30, no. 5, pp. 1052-1058, 2009.

[76] J. C. Park, B. Song, J. S. Kim, S. H. Park, H. K. Kim, Z. Liu, T. S. Suh, and W. Y. Song, "Fast compressed sensing-based CBCT reconstruction using Barzilai-Borwein formulation for application to on-line IGRT," *Medical physics,* vol. 39, no. 3, pp. 1207-1217, 2012.

[77] W. R. Crum, T. Hartkens, and D. L. Hill, "Non-rigid image registration: theory and practice," (in eng), *Br J Radiol,* vol. 77 Spec No 2, pp. S140-53, 2004.

[78] Z. Gu and B. Qin, "Nonrigid registration of brain tumor resection mr images based on joint saliency map and keypoint clustering," *Sensors,* vol. 9, no. 12, pp. 10270-10290, 2009.

[79] P. Thevenaz, U. E. Ruttimann, and M. Unser, "A pyramid approach to subpixel registration based on intensity," *IEEE transactions on image processing,* vol. 7, no. 1, pp. 27-41, 1998.

[80] M. Tan and A. Qiu, "Large Deformation Multiresolution Diffeomorphic Metric Mapping for Multiresolution Cortical Surfaces: A Coarse-to-Fine Approach," *IEEE Transactions on Image Processing,* vol. 25, no. 9, pp. 4061-4074, 2016.

[81] X. Huang, J. Moore, G. Guiraudon, D. L. Jones, D. Bainbridge, J. Ren, and T. M. Peters, "Dynamic 2D ultrasound and 3D CT image registration of the beating heart," *IEEE transactions on medical imaging,* vol. 28, no. 8, pp. 1179-1189, 2009.

[82] G. H. Glover, T. Q. Li, and D. Ress, "Image-based method for retrospective correction of physiological motion effects in fMRI: RETROICOR," *Magnetic resonance in medicine,* vol. 44, no. 1, pp. 162-167, 2000.

[83] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of CUDA and OpenCL," in *Parallel Processing (ICPP), 2011 International Conference on*, 2011, pp. 216-225: IEEE.

[84] D. Kirk, "NVIDIA CUDA software and GPU parallel computing architecture," in *ISMM*, 2007, vol. 7, pp. 103-104.

[85] W. Palenstijn, K. Batenburg, and J. Sijbers, "Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)," *Journal of structural biology,* vol. 176, no. 2, pp. 250-253, 2011.

[86] D. Rogers, "Fifty years of Monte Carlo simulations for medical physics," *Physics in medicine and biology,* vol. 51, no. 13, p. R287, 2006.

[87] C. Sadowsky, J. D. Cohen, and R. H. Taylor, "Rendering tetrahedral meshes with higher-order attenuation functions for digital radiograph reconstruction," in *Visualization, 2005. VIS 05. IEEE*, 2005, pp. 303-310: IEEE.

[88] F. P. Oliveira and J. M. R. Tavares, "Medical image registration: a review," *Computer methods in biomechanics and biomedical engineering,* vol. 17, no. 2, pp. 73-93, 2014.

[89] B. B. Avants, N. J. Tustison, M. Stauffer, G. Song, B. Wu, and J. C. Gee, "The Insight ToolKit image registration framework," (in eng), *Front Neuroinform,* vol. 8, 2014.

[90] H. J. Johnson, M. McCormick, and L. Ibanez, "The ITK software guide," *Kitware, Inc.,* 2013.

[91] G. Sharp, N. Kandasamy, H. Singh, and M. Folkert, "GPU-based streaming architectures for fast cone-beam CT image reconstruction and *demons* deformable registration," *Physics in medicine and biology,* vol. 52, no. 19, p. 5771, 2007.

[92] C. Dittamo and A. Cisternino, "GPU White paper," 2008.

[93] N. Courty and P. Hellier, "Accelerating 3D non-rigid registration using graphics hardware," *International Journal of Image and Graphics,* vol. 8, no. 01, pp. 81-98, 2008.

[94] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2003, pp. 92-101: Eurographics Association.

[95] P. Muyan-Ozcelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of *demons*," in *Computational Sciences and Its Applications, 2008. ICCSA'08. International Conference on*, 2008, pp. 223-233: IEEE.

[96] S. S. Samant, J. Xia, P. Muyan-Özçelik, and J. D. Owens, "High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy," *Medical physics,* vol. 35, no. 8, pp. 3546-3553, 2008.

[97] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. Choi, E. Castillo, A. Majumdar, T. Guerrero, and S. B. Jiang, "Implementation and evaluation of various *demons* deformable image registration algorithms on a GPU," *Physics in Medicine and Biology,* vol. 55, no. 1, p. 207, 2010.

[98] Y. Huang, T. Tong, W. Liu, Y. Fan, H. Feng, and C. Li, "Accelerated diffeomorphic non-rigid image registration with CUDA based on *demons* algorithm," in *Bioinformatics and Biomedical Engineering (iCBBE), 2010 4th International Conference on*, 2010, pp. 1-4: IEEE.

[99] M. D. McCool, A. D. Robison, and J. Reinders, *Structured parallel programming: patterns for efficient computation*. Elsevier, 2012.

[100] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," in *ACM SIGARCH Computer Architecture News*, 2014, vol. 42, no. 1, pp. 35-50: ACM.

[101] H. Casanova, A. Legrand, and Y. Robert, *Parallel algorithms*. CRC Press, 2008.

[102] Y. Bai, N. Guo, and G. Agbegha, "Fuzzy Interpolation and Other Interpolation Methods Used in Robot Calibrations," *Journal of Robotics,* vol. 2012, 2012.

[103] N. Bell and J. Hoberock, "Thrust: A productivity-oriented library for CUDA," in *GPU computing gems Jade edition*: Elsevier, 2011, pp. 359-371.

[104] J. A. Stratton, V. Grover, J. Marathe, B. Aarts, M. Murphy, Z. Hu, and W.-m. W. Hwu, "Efficient compilation of fine-grained SPMD-threaded programs for multicore CPUs," in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, 2010, pp. 111-119: ACM.

[105] C. Nvidia, "Compute unified device architecture programming guide," 2007.

[106] B. T. Yeo, M. R. Sabuncu, T. Vercauteren, N. Ayache, B. Fischl, and P. Golland, "Spherical *demons*: fast diffeomorphic landmark-free surface registration," *IEEE transactions on medical imaging,* vol. 29, no. 3, pp. 650-668, 2010.

[107] M. Lorenzi, N. Ayache, G. B. Frisoni, X. Pennec, and A. s. D. N. Initiative, "LCC-*Demons*: a robust and accurate symmetric diffeomorphic registration algorithm," *NeuroImage,* vol. 81, pp. 470-483, 2013.

[108] L. Zhao and K. Jia, "Deep adaptive log-*Demons*: Diffeomorphic image registration with very large deformations," *Computational and mathematical methods in medicine,* vol. 2015, 2015.

[109] L. Zhang and Y. Wen, "Log-*Demons* with driving force for large deformation image registration," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, 2016, pp. 3052-3059: IEEE.

[110] H. Lu, M. Reyes, A. Šerifović, A. Šerifović, S. Weber, Y. Sakurai, H. Yamagata, and P. C. Cattin, "Multi-modal diffeomorphic *demons* registration based on point-wise mutual information," in *Biomedical Imaging: From Nano to Macro, 2010 IEEE International Symposium on*, 2010, pp. 372-375: IEEE.

[111] S. Nithiananthan, S. Schafer, D. J. Mirota, J. W. Stayman, W. Zbijewski, D. D. Reh, G. L. Gallia, and J. H. Siewerdsen, "Extra-dimensional *Demons*: A method for incorporating missing tissue in deformable image registration," *Medical physics,* vol. 39, no. 9, pp. 5718-5731, 2012.

[112] S. Klein, M. Staring, and J. P. Pluim, "Evaluation of optimization methods for nonrigid medical image registration using mutual information and B-splines," *IEEE transactions on image processing,* vol. 16, no. 12, pp. 2879-2890, 2007.

[113] H. Talbi and M. Batouche, "Particle swam optimization for image registration," in *Information and Communication Technologies: From Theory to Applications, 2004. Proceedings. 2004 International Conference on*, 2004, pp. 397-398: IEEE.

[114] L.-t. Zheng and R.-f. Tong, "Image registration algorithm using an improved PSO algorithm," in *International Conference on Information and Management Engineering*, 2011, pp. 198-203: Springer.

[115] C.-L. Lin, A. Mimori, and Y.-W. Chen, "Hybrid particle swarm optimization and its application to multimodal 3d medical image registration," *Computational intelligence and neuroscience,* vol. 2012, p. 6, 2012.

[116] S. Chakraborty, N. Dey, S. Samanta, A. S. Ashour, C. Barna, and M. Balas, "Optimization of non-rigid *Demons* registration using cuckoo search algorithm," *Cognitive Computation,* vol. 9, no. 6, pp. 817-826, 2017.

[117] L. Gepstein and S. J. Evans, "Electroanatomical mapping of the heart: basic concepts and implications for the treatment of cardiac arrhythmias," *Pacing and clinical electrophysiology,* vol. 21, no. 6, pp. 1268-1278, 1998.

[118] (06-Mar-2017). *The Future of Cardiac Mapping (Electrophysiology Study of the Heart: Mapping Procedure) (Cardiac Arrhythmias)* Available: http://what-when-how.com/cardiac-arrhythmias-new-considerations/the-future-of-cardiac-mapping-electrophysiology-study-of-the-heart-mapping-procedure-cardiac-arrhythmias-part-2/

[119] B. Benito and M. E. Josephson, "Ventricular Tachycardia in Coronary Artery Disease," *Revista Española de Cardiología (English Edition),* 10.1016/j.rec.2012.03.022 vol. 65, no. 10, pp. 939-955, 2012.

[120] J. Dong, T. Dickfeld, D. Dalal, A. Cheema, C. R. Vasamreddy, C. A. Henrikson, J. E. Marine, H. R. Halperin, R. D. Berger, and J. A. Lima, "Initial Experience in the Use of Integrated Electroanatomic Mapping with Three-Dimensional MR/CT Images to Guide Catheter Ablation of Atrial Fibrillation," *Journal of cardiovascular electrophysiology,* vol. 17, no. 5, pp. 459-466, 2006.

[121] J. Dong, H. Calkins, S. B. Solomon, S. Lai, D. Dalal, A. Lardo, E. Brem, A. Preiss, R. D. Berger, and H. Halperin, "Integrated electroanatomic mapping

with three-dimensional computed tomographic images for real-time guided ablations," *Circulation,* vol. 113, no. 2, pp. 186-194, 2006.

[122] K.-W. Kwok, K.-H. Lee, Y. Chen, W. Wang, Y. Hu, G. C. Chow, H. S. Zhang, W. G. Stevenson, R. Y. Kwong, and W. Luk, "Interfacing fast multi-phase cardiac image registration with MRI-based catheter tracking for MRI-guided electrophysiological ablative procedures," ed: Am Heart Assoc, 2014.

[123] J. Pouliot, A. Bani-Hashemi, J. Chen, M. Svatos, F. Ghelmansarai, M. Mitschke, M. Aubin, P. Xia, O. Morin, and K. Bucci, "Low-dose megavoltage cone-beam CT for radiation therapy," *International Journal of Radiation Oncology• Biology• Physics,* vol. 61, no. 2, pp. 552-560, 2005.

[124] H. Tachibana and T. Akimoto, "IGRT for IMRT," in *Intensity-Modulated Radiation Therapy: Clinical Evidence and Techniques*, Y. Nishimura and R. Komaki, Eds. Tokyo: Springer Japan, 2015, pp. 85-112.

[125] C. C. Ling, E. Yorke, and Z. Fuks, "From IMRT to IGRT: frontierland or neverland?," *Radiotherapy and oncology,* vol. 78, no. 2, pp. 119-122, 2006.

[126] K. Brock, M. Lee, C. Eccles, M. Velec, J. Moseley, and L. Dawson, "Deformable registration and dose accumulation to investigate marginal liver cancer recurrences," *International Journal of Radiation Oncology• Biology• Physics,* vol. 72, no. 1, p. S538, 2008.

[127] B. Sorcini and A. Tilikidis, "Clinical application of image-guided radiotherapy, IGRT (on the Varian OBI platform)," *Cancer/Radiothérapie,* vol. 10, no. 5, pp. 252-257, 2006.

[128] K.-H. Lee, K. C. D. Fu, Z. Guo, Z. Dong, M. C. Leong, C.-L. Cheung, A. P.-W. Lee, W. Luk, and K.-W. Kwok, "MR Safe Robotic Manipulator for MRI-Guided Intracardiac Catheterization," *IEEE/ASME Transactions on Mechatronics,* vol. 23, no. 2, pp. 586-595, 2018.

[129] Z. Guo, Z. Dong, K.-H. Lee, C. L. Cheung, H.-C. Fu, J. D. Ho, H. He, W.-S. Poon, D. T.-M. Chan, and K.-W. Kwok, "Compact Design of a Hydraulic Driving Robot for Intra-operative MRI-guided Bilateral Stereotactic Neurosurgery," *IEEE Robotics and Automation Letters,* vol. 3, no. 3, pp. 2515-2522, 2018.