GPU BASED IMAGE REGISTRATION

Parag Bhosale

CE-MS-2017-15

Abstract

Currently, non-rigid image registration algorithms are too time intensive to use in time-critical applications. To solve this problem, stochastic gradient descent (SGD) has been implemented in image registration. But, SGD depends on manual step size selection which is difficult and time consuming. To avoid such manual selection, SGD has been improved further by using adaptive stochastic gradient descent (ASGD) and fast adaptive stochastic gradient descent (FASGD) to select an optimal step size automatically. Although FASGD has reduced the computation time drastically, non-rigid registration still cannot be used in time critical applications. So far, a serial implementation of FASGD has been tested on CPU architecture in elastix toolbox. Thus, a parallel implementation of SGD can be a possible solution to this problem.

The work proposed in this thesis implemented a NiftyReg toolbox extension to graphic processing units (GPUs), divided into two methods. First, NiftyReg2, a possible optimization of the current NiftyReg. Second, NiftyRegSGD, a high performance implementation of SGD on the GPU framework of NiftyReg. A novel sampling strategy, random chunk sampling is also proposed which is tailored to the GPU architecture. Random chunk sampling is an optimization to utilize memory bandwidth of GPU effectively to increase throughput of CUDA kernels.

Experiments have been performed on 3D lung CT data of 19 patients, which compared NiftyRegSGD (with and without random chunk sampler) with CPU-based elastix FASGD and NiftyReg. The registration runtime was 21.5s, 13.02s, 4.4s and 2.8s for elastix-FASGD, NiftyReg2, NiftyRegSGD without, and NiftyRegSGD with random chunk sampling, respectively, while similar accuracy was obtained. Thus, proposed GPU based non-rigid registration can be used for a time critical application with further extensions. The abstract which discusses the work done during this thesis has been accepted for publication in the medical imaging conference of the Society of Photographic Instrumentation Engineers (SPIE).

TU Delft Delft University of Technology

CE Lab Computer Engineering Laboratory

GPU BASED IMAGE REGISTRATION

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

PARAG BHOSALE
born in Pune, India

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

GPU BASED IMAGE REGISTRATION

by PARAG BHOSALE

Abstract

Currently, non-rigid image registration algorithms are too time intensive to use in time-critical applications. To solve this problem, stochastic gradient descent (SGD) has been implemented in image registration. But, SGD depends on manual step size selection which is difficult and time consuming. To avoid such manual selection, SGD has been improved further by using adaptive stochastic gradient descent (ASGD) and fast adaptive stochastic gradient descent (FASGD) to select an optimal step size automatically. Although FASGD has reduced the computation time drastically, non-rigid registration still cannot be used in time critical applications. So far, a serial implementation of FASGD has been tested on CPU architecture in elastix toolbox. Thus, a parallel implementation of SGD can be a possible solution to this problem.
The work proposed in this thesis implemented a NiftyReg toolbox extension to graphic processing units (GPUs), divided into two methods. First, NiftyReg2, a possible optimization of the current NiftyReg. Second, NiftyRegSGD, a high performance implementation of SGD on the GPU framework of NiftyReg. A novel sampling strategy, random chunk sampling is also proposed which is tailored to the GPU architecture. Random chunk sampling is an optimization to utilize memory bandwidth of GPU effectively to increase throughput of CUDA kernels.
Experiments have been performed on 3D lung CT data of 19 patients, which compared NiftyRegSGD (with and without random chunk sampler) with CPU-based elastix FASGD and NiftyReg. The registration runtime was 21.5s, 13.02s, 4.4s and 2.8s for elastix-FASGD, NiftyReg2, NiftyRegSGD without, and NiftyRegSGD with random chunk sampling, respectively, while similar accuracy was obtained. Thus, proposed GPU based non-rigid registration can be used for a time critical application with further extensions. The abstract which discusses the work done during this thesis has been accepted for publication in the medical imaging conference of the Society of Photographic Instrumentation Engineers (SPIE).

Laboratory : Computer Engineering
Codenumber : CE-MS-2017-15

Committee Members :

Advisor:	Dr.ir. Floris Berendsen, LKEB, LUMC
Chairperson:	Dr.ir. Zaid Al-Ars, CE, TU Delft
Member:	Dr.ir. A. van Genderen, CE, TU Delft
Member:	Dr.ir. Marius Staring, LKEB, LUMC

Dedicated to my family and friends

Contents

List of Figures

List of Tables

List of Acronyms

SGD Stochastic gradient descent

GPU Graphic processing unit

ASGD Adaptive stochastic gradient descent

FASGD Fast adaptive stochastic gradient descent

CPU Central processing unit

CT Computed tomography

CUDA Compute Unified Device Architecture

IBM (InternationalBusiness Machine

SONAR SOund Navigation And Ranging

MRI Magnetic Resonance Imaging

IMPT Intensity Modulated Proton Therapy

DNA DeoxyriboNucleicAcid

GPGPU General-Purpose computing on Graphics Processing Units

CAD Computer Assisted Diagnosis

CAS Computer Assisted Surgery

CAT Computer Assisted Therapy

MI Mutual Information

NMI Normalized Mutual Information

DOF Degrees Of Freedom

B-spline Basis Spline

FFD Free Form Deformation

ITK Insight Segmentation and Registration Toolkit

AVX Advanced Vector Extensions

SPREAD Software Performance and Reproducibility of Emphysema Assessment: Demonstration

TRE Target Registration Error

Acknowledgements

First, I would like to thank Marius Staring to give me an opportunity to work on this thesis. With this thesis, I am ending my master's journey at TU Delft. Last two years have been incredible for my personal and academical growth. I would like to show my gratitude to the people who supported me during my study.

My thesis supervisor at university, Zaid-Al-Ars was extremely helpful during my thesis and my study. His door was always open for the advice. I am extremely thankful to Zaid for his guidance during this thesis. This thesis was done at Leiden University Medical Center in Netherlands. I was very fortunate to have Floris Berendsen as my thesis supervisor. He was the one who pushed me into the water to learn swimming. He was extremely supportive throughout my thesis. His guidance was critical in the thesis implementation. I can not thank him enough for helping me with my first research paper which has been accepted to the SPIE medical conference. I also thank Zaid and Marius to help me with the paper.

I am thankful to Nvidia for providing the latest graphic card for the thesis. I would like to thank Intel for providing free student license for the profiling tool.

Finally, I am extremely grateful to my parents and family who motivated and supported me throughout my life. Without them, this accomplishment would not be possible.

PARAG BHOSALE
The Hague, The Netherlands

1 Introduction

On November 8, 1895 in Germany, Wilhelm Conrad Roentgen made the remarkable discovery of X-rays [15]. These rays penetrated through human flesh to take a photograph of the bones. Roentgen asked his wife to be a volunteer, and he took a photograph of her hand. Only the bones and the ring on his wife's hand were visible in that photograph. This discovery was made public, and only within a month, X-rays were used to treat a patient successfully [15]. Two Birmingham doctors took a X-ray photograph to remove a broken needle from a patient's hand. This discovery has been helping physicians to diagnose a medical case without cutting the patient. Since then, radiology has been an integral part of medical therapy.

Around the same time, the United States census bureau was facing a problem during the 1890 census. Ten years earlier, 1880 census was done manually and it took seven years to process. For the 1890, the bureau predicted it would take them more than ten years to survey the growing population of 62 million people [16]. So, a contest was held to address this problem. This contest was won by Herman Hollerith, who proposed an electro-mechanical punched card tabulator to tabulate the census data. This invention enabled the census to be processed within six weeks saving a lot of time and money. Herman founded Tabulating Machine Company which later amalgamated with other three companies to form Computing-Tabulating-Recording Company in 1911, which is nowadays known as IBM (International Business Machines Corporation). Then, in the 20th century, IBM made faster and better tabulating machines mainly used for accounting in the finance industry.

Figure 1.1: Left picture shows *Hand mit Ringen* (hand with rings), the first medical X-ray photo taken by Wilhelm Roentgen of his wife Anna Bertha Ludwig's hand [1]. The right picture shows a tabulating machine used during 1890 US census [2].

Thus, in the late 19th century, both medical imaging and hardware computing made

a significant impact on their respective fields. In the 20th century, both fields had other breakthroughs from the enigma machine to the rise of Microsoft and Intel, from SONAR to computed tomography (CT). In mid 20th century, these two fields converged when CT was introduced by Hounsfield and Cormack for which they won the Nobel Prize in Physiology or Medicine. This was the first time a digital medical image was constructed. Apart from X-ray based CT, different modalities were developed like Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET) and ultrasound. All these various modalities have advantages and disadvantages and can be used to acquire medical images for diagnostics.

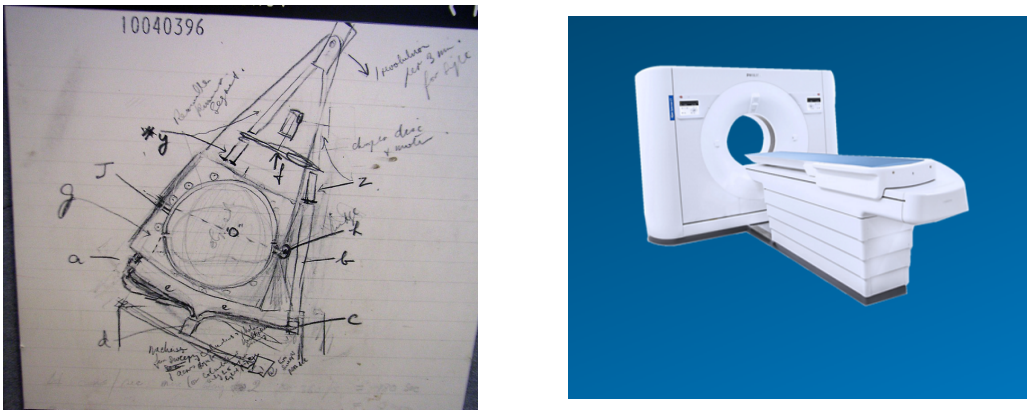


Figure 1.2: Figure on left shows the sketch of CT scan drawn by Godfrey Hounsfield [3] and figure on right shows the latest IQon spectral CT scanner by Philips [4]

Once a digital image is acquired and depending on the application, further analysis is needed for optimal diagnostic. Such optimal diagnostic can be done by image processing with the help of a computer hardware providing the necessary computing power. For example, image segmentation is an algorithm which differentiates an image into separate smaller images. Another example is to determine a spatial relation between two images. For example, two CT scans are taken, one before and another after the treatment. To find out the progression of the tumor, a spatial relation between the two images is needed. Another example is to align two images taken by two different modalities. This spatial correlation is established between MRI and ultrasound scanner to cancel the disadvantages by MRI and ultrasound. This spatial relation is determined by algorithms referred to as image registration.

1.1 Image registration and proton therapy

Image registration is one of the main tasks in medical processing pipeline for clinical applications. As stated earlier, registration aligns two images and establishes a spatial relation between images. This relation is established with the help of optimization of transformation parameters. In general, there are two types of registration, affine, and non-rigid. Affine registration can have around 16 parameters while non-rigid can have up to 100k (or more) parameters for a 3D medical image. Thus, such a high num-

ber of parameters under optimization makes the non-rigid registration computationally intensive.

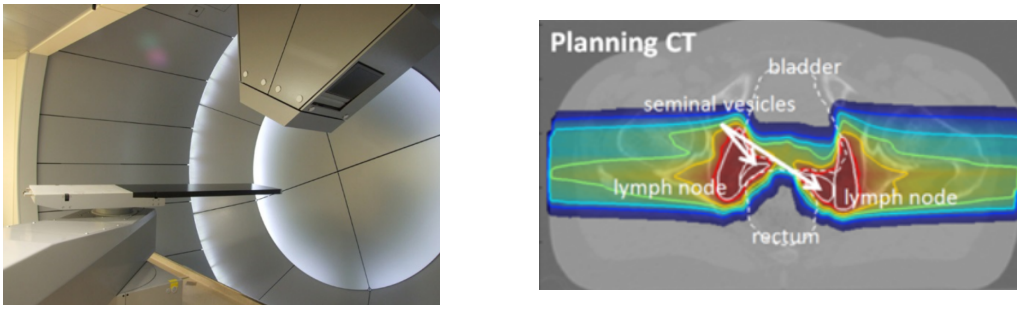


Figure 1.3: Left picture shows intensity modulated proton therapy (IMPT) equipment [5] which is used to guide high energy proton beams and a proton dose distribution is shown in the picture on the right [5].

Radiation therapy is a cancer treatment which uses high energy radiation of X-rays, gamma rays or charged particles to destroy cancer cells by damaging Deoxyribonucleic acid (DNA) of cancer beyond repair. These damaged cancer cells then eventually die. But, this radiation beams can also affect healthy cells nearby. In intensity modulated proton therapy (IMPT), a radiation dose is distributed in such a way that the proton beams will hit the tumor with the highest intensity while surrounding healthy organs will be hit with zero or lowest intensity, thus reducing irradiation of healthy tissues and avoiding complications. Proton therapy has some advantages over other beam radiation therapy. Proton therapy has fewer and lower side effect than other beam therapy. Hence, IMPT is used for the cancer treatment of lung, prostate and brain cancer. Before therapy, CT or MRI scans of a patient are taken. A non-rigid registration is used to locate cancer tumors using these scans. Once tumors are located, distribution of a proton radiation dose is finalized. With higher deformations and extra bending constraints, a non-rigid registration can take up to 10 minutes. But, the location of a tumor may change because of human respiration, bladder filling or patient's motion. This means during proton therapy, the distribution plan may not be optimal. Hence, healthy tissues can suffer from a radiation overdose while there can be an underdose in the tumor.

Thus, non-rigid registration needs to be revised to find tumor locations within few seconds to have an optimal treatment plan. Similar to a tabulating machine in 1890 census problem, modern day computers are assisting medical image processing with high computing power. But, this computing power is not enough for a non-rigid registration. Graphics Processing Unit (GPU) can be a possible solution to this problem. General-purpose computing on graphics processing units (GPGPU) has been used in the healthcare field. H.A.D. Nguyen et al. [17] and G. Smaragdos et al. [18] proposed the simulation of computationally intensive neuron model on GPU. Houtgast et al. [19] and Ren et al. [20] accelerated big data algorithms related to the genomics analysis using GPUs. Thus, GPGPU is a possible solution to accelerate the non-rigid registration. Hence, the work done in this thesis focuses on answering the question, **can non-rigid registration be used in near real time applications using GPGPU?**

To answer the above main question, it must be divided into sub-questions as follows:

1. What is image registration and what makes a non-rigid registration time consuming?
2. What are the current fast implementations for a non-rigid registration using GPU?
3. Can these implementations be further improved to perform faster?
4. What can be the possible optimizations to make them faster?
5. Are these optimizations enough to solve the main problem?

1.2 Thesis outline

This report discusses the above five sub-questions in order. Chapter 2 answers the first two research questions by introducing the basics of medical image and image registration. Related work and possible implementations are discussed as well. In Chapter 3, profiling and analysis of potential implementations are discussed to answer the third question. Possible optimizations are also proposed in Chapter 3 to answer the fourth question. Chapter 4 discusses the implementation of these optimizations in detail. Chapter 5 and 6 discuss the results and conclusions to answer the fifth question. Future work is also stated in Chapter 6 to answer the fifth research question further. This thesis is done at the division of image processing (Laboratorium voor Klinische en Experimentele Beeldverwerking LKEB) of the Leiden university medical center (LUMC). Based on the results of this work, we published a paper [21] which answers all the above research questions in the conference of the Society of Photographic Instrumentation Engineers (SPIE) on medical imaging, February 2018.[1]

[1]Refer the appendix for the paper abstract which has been accepted to SPIE conference

Background and related work 2

2.1 Medical image

Unlike a normal image, a medical image has two co-ordinate systems, world (or physical) and image coordinate. The image co-ordinate system is similar to a normal non medical image and it can be described using pixels or voxels. Physical coordinate system has spatial information about that image. This spatial information is a vital part of medical imaging. This information represents the physical difference between pixels (or voxels) and the physical location of the image in the space. The space denotes the area under a scanner.

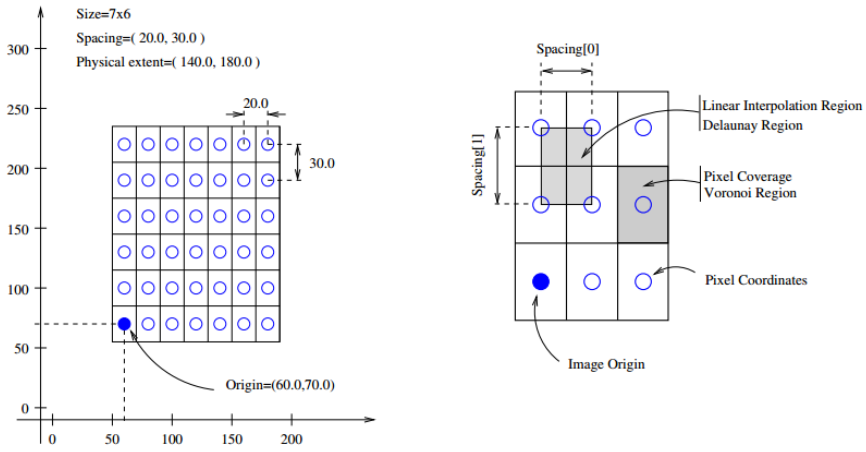


Figure 2.1: A typical medical image with spatial information [6]

Figure 2.1 shows the spacial information terminologies of a medical image. The image origin and spacing (δ) are two important factors to know the physical position of the image or a structure in that image. Medical diagnosis, image-guided surgery assisted radiation therapy or feature extraction is not possible without such spatial information.

2.2 Image registration

Medical imaging plays a vital role in clinical research and medical care. It is used to detect diseases, to follow up on a disease progression and to select therapies for a disease. Image registration is one of the key parts of the medical image processing and analysis used in Computer-assisted Diagnosis (CAD), Computer-aided Therapy (CAT) and Computer-assisted Surgery (CAS). The fundamental task of the image registration is

to align medical images or to find out a spatial relationship between two or more images. Each point in one image is mapped to the corresponding point in the other image. The first image in the previous statement is called floating, moving or source image while the latter is called fixed or target image. Such mapping of points is done by the transformation models. Typically, two types of transformations are used in registration, affine and non-rigid. The quality of a mapping (or an alignment) is measured using a cost function (or a metric). The cost function is either similarity (e.g. mutual information) or dissimilarity (e.g. mean squared distance). To align two images perfectly, the similarity between those images should be maximum or the dissimilarity between them should be minimum. To achieve this, transformation parameters are optimized iteratively with the help of optimizer (e.g. gradient descent). The transformation model, cost function, and optimizer are application dependent. Figure 2.2 shows a typical image registration framework. The components of a registration algorithm are discussed in the following sections.

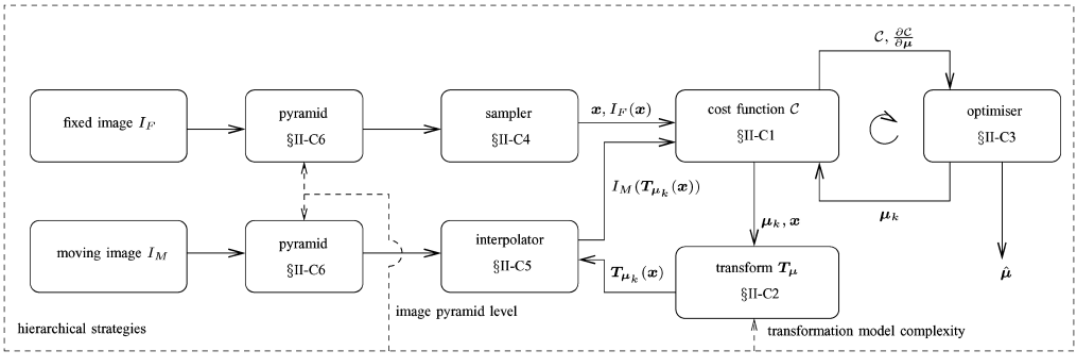


Figure 2.2: A typical image registration algorithm [7]

2.2.1 Cost function

Cost function is the measure of alignment. As stated in section 2.2, registration aims to find the optimal cost function between two images. The cost function in image registration can be expressed as:

$$C(\mu) = \Psi\left(\frac{1}{|\Omega_F|}\sum_{x_i \in \Omega_F} \xi(F(x_i), M(\mathbf{T}(x_i, \mu)))\right), \tag{2.1}$$

where $\Psi(u)$ and $\xi(u, v)$ are continuous and differentiable functions, and where Ω_F is a discrete set of voxel coordinates from the fixed image [22]. As stated earlier, the cost function can be either distance or similarity between images.

2.2.2 Sampler

The role of a sampler is to choose a set from the available sample space. Typically, full sampling, i.e., full images are used in registration. But in stochastic optimization, a

subset of a sample space is used. The sampler will be discussed in details in the stochastic gradient descent.

2.2.3 Interpolation

The optimized parameters are then applied to the moving image and an interpolated (or a warped or a deformed) image is obtained using an interpolator. This interpolated image is used as the moving image for the next iteration. Hence, at each iteration, a new moving image is interpolated. In some literature, this is also called resampling. This new image $M(\mathbf{T}(x_i, \mu)$, is also called deformed or warped image. Nearest neighbour, bilinear, and bi-cubic interpolation are usually used in registration.

2.2.4 Transformation models

The main goal of a transformation is to map any point from an image into the corresponding point from another image. A transformation model can be stated using parameters (or degrees of freedom (DOF)). A transformation is typically defined by following equation where transformation $\mathbf{T}$ maps a point (x, y, z) from source (moving) to (x', y', z') from target (fixed) image.

$$\mathbf{T} : (x, y, z) \mapsto (x', y', z') \tag{2.2}$$

In general, type of registration is defined by the nature of transformation used in registration framework. Affine and non-rigid are mainly two types of transformations, where affine is an extension of rigid transformation.

Rigid and affine registration

A rigid transformation has six parameters, three rotations and three translations. An extension of rigid is called an affine transformation. An affine transformation has twelve parameters. In addition to rotation and translation, shearing and scaling are also allowed by affine transform [23]. Affine or linear transformation is defined by [23],

$$\mathbf{T} : (x, y, z) = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.3}$$

Affine and rigid transformations are also called as the global transformations. Because, all points on an image are transformed by the same equation 2.3. But affine transform is not enough for some applications. For example, to find out the progression of a tumor inside the brain, a local transform is needed to be applied to the tumor only, not to the whole image. In such cases, a non-rigid transformation is used as local transform to map changes in a local object (like a tumor).

Nonrigid registration

Non-rigid transform is used to find the local spatial relation which can not be established by an affine transform. The basic idea in non-rigid is to use a high number of parameters

(or DOF), typically around 50k. In mathematical terms, this can be achieved by using a linear combination of the basis function θ to describe a deformation field [23]. Hence, equation 2.3 is modified as,

$$\mathbf{T} : (x, y, z) = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0n} \\ a_{10} & a_{11} & \cdots & a_{1n} \\ a_{20} & a_{21} & \cdots & a_{2n} \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \theta_1(x, y, z) \\ \vdots \\ \theta_n(x, y, z) \\ 1 \end{bmatrix} \tag{2.4}$$

Where θ is a basis function such as Fourier or wavelet basis functions [23] and n is the order of basis function. Usually, basis spline (B-spline) is used as a basis function. The word spline is referred to the long strips of a wood or metal, which are bent by attaching weights to use in ships and planes [23], refer figure 2.3.

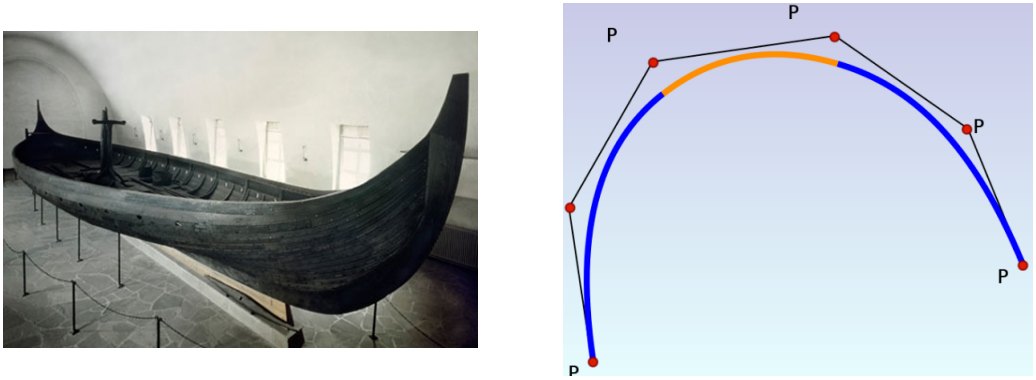


Figure 2.3: Picture on left shows *the Gokstad Viking ship*, a typical example of splines in shipbuilding [8]. On the right side, picture shows a spline used in computer aided design and computer graphics [9].

The parameters in a non-rigid transform are also called the control points. At these control points, the displacement is interpolated or approximated to map points between two images [23]. The control points are given by,

$$\mathbf{T}(\phi_i) = \phi_i' \qquad i = 1, 2, \cdots, n \tag{2.5}$$

Where ϕ is a control point, and n is a total number of control points. Freeform deformations (FFDs) is a nonlinear transformation which is based on B-spline. The key idea of FFD is to deform an object (in this case, an image) by manipulating underlying mesh of control points (i.e. control point grid) [23]. A deformation at any point (x, y, z) is given by [23],

$$\mathbf{u}(x, y, z) = \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} \theta_l(u)\theta_m(v)\theta_n(w)\phi_{i+l,j+m,k+n} \tag{2.6}$$

where $i = \left\lfloor \frac{x}{\delta} \right\rfloor - 1, j = \left\lfloor \frac{y}{\delta} \right\rfloor - 1, k = \left\lfloor \frac{z}{\delta} \right\rfloor - 1, u = \frac{x}{\delta} - \left\lfloor \frac{x}{\delta} \right\rfloor, v = \frac{y}{\delta} - \left\lfloor \frac{y}{\delta} \right\rfloor, w = \frac{z}{\delta} - \left\lfloor \frac{z}{\delta} \right\rfloor$

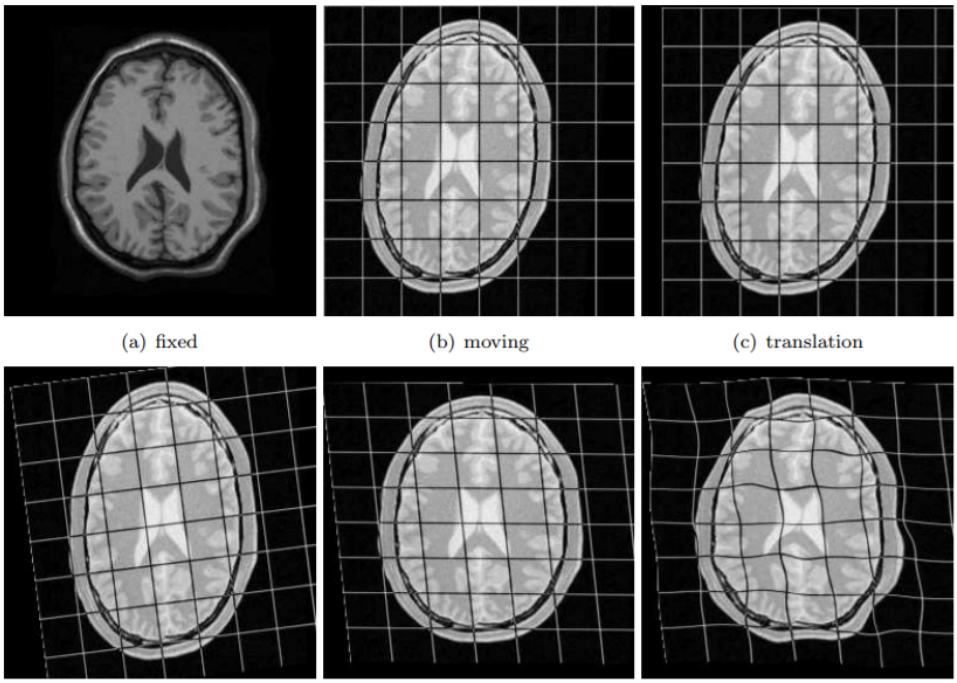


Figure 2.4: Figure shows the transformation models used in image registration [10]. (a) and (b) are two images to be aligned. Various transformations (from (c)-(f)) can be applied to the moving image (b) to make it align with the fixed image (a). In a typical registration application, an affine transformation (e) is followed by a non-rigid (f) transformation.

and δ is uniform spacing among x,y,z axes.[1] ϕ are control points and θ are B-spline function, which are defines as,

$$\begin{aligned}\theta_0(s) =&(1-s)^3/6\\ \theta_1(s) =&(3s^3-6s^2+4)/6\\ \theta_2(s) =&(-3s^3+3s^3-3s+1)/6\\ \theta_3(s) =&s^3/6\end{aligned} \tag{2.7}$$

2.2.5 Optimizer

Image registration is an iterative optimization problem that can be formulated as,

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \gamma_k \mathbf{g}_k, \tag{2.8}$$

where $\boldsymbol{\mu}$ represents the transformation parameters at iteration k, $\mathbf{g}$ is the search direction and γ_k the stepsize. The optimizer uses a search direction $\mathbf{g}_k$ based on the

[1] In general, δ is not same among all axes.

gradient to select the optimal step size γ_k. For the search direction, the gradient $\mathbf{g}_k := \partial C/\partial \boldsymbol{\mu}$ of the cost function C can be selected. A step size should be chosen wisely. Too small step size leads to a high number of iterations which leads to a higher computational time. On the other hand, a large step size leads to instability in the registration which may leads to incorrect optimization of parameters.

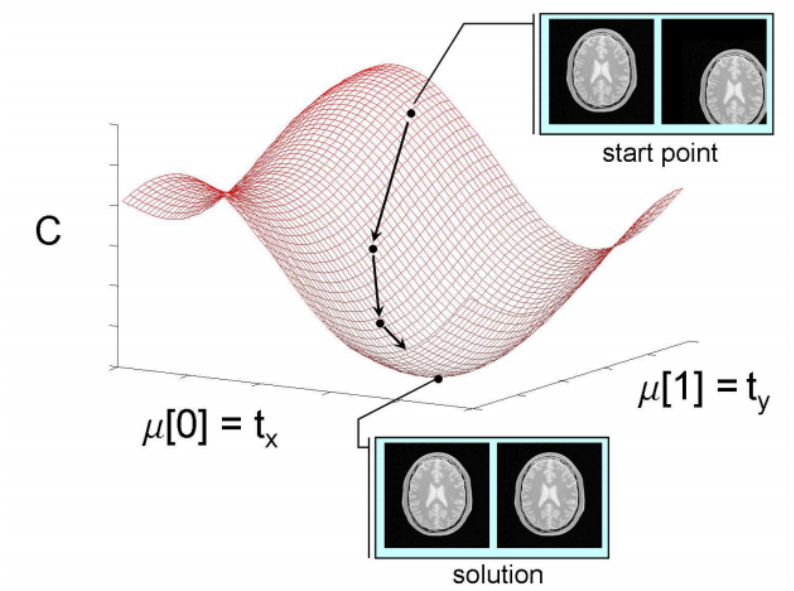


Figure 2.5: Figure shows a plot of cost function [7]. Here two traslation parameters along x and y directions are optimized. The cost function is minimum at the solution point where two images are seen to be aligned perfectly.

Stopping criteria for registration

Since registration is an iterative algorithm, typically a stopping criterion is used. One stopping criterion is a maximum number of iterations, k_{max}. In this criterion, a registration is computed till $k = k_{max}$ for equation 2.8. The second criterion is to use a line search. The key idea of a line search is to take few (e.g., ten iterations) steps in a direction till the optimum cost function is obtained. If optimum cost function is obtained before k_{max}, the registration is stopped. In general, conjugate gradient optimizer is used with the line search. Typically, a conjugate gradient optimizer has higher convergence rate than a simple gradient descent optimizer [24].

Multi-resolution registration

The registration can be done using the multiple resolutions (or levels), starting from the registration at the lower resolution to the registration at the higher resolution. This is done using downsampling with or without Gaussian smoothing [7]. This approach is

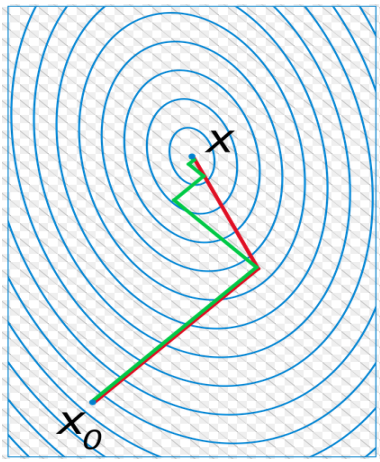


Figure 2.6: The figure shows gradient descent vs conjugate gradient [11]. Green path corresponds to the gradient descent, and red path represents the conjugate gradient. Starting from X_0, both optimizers converge at an optimum value X. The conjugate gradient takes two iterations while gradient descent takes five iterations.

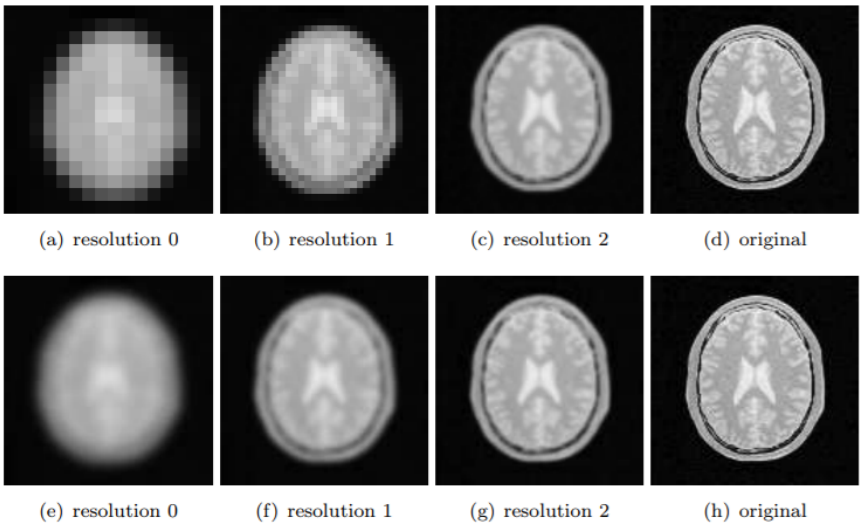
also called as pyramids (refer figure 2.2). The word pyramid indicates the data intensity increases as the level resolution increases.

Similarly, in case of control points of B-spline, pyramid strategy is applied to the number of parameters. At lower resolution, a coarse control point grid is used while, at higher resolution, a refined control point grid is used. The multi-resolution strategy not only reduces data and transformation complexity but also improves registration quality [7]. The main role of using pyramid (in case of non-rigid registration) is to align bigger structure faster with lower resolution (with less data) and then align smaller structure with higher resolution. In addition to this, pyramid also eliminates false minima.

2.3 Problem with non-rigid revisited

As the number of parameters used in non-rigid registration can be as high as 10^5 (term ϕ in equation 2.6), non-rigid registration becomes more arithmetic intensive with higher computation time as resolution level increases. Typically, non-rigid registration can take up from few minutes to more than 10 minutes. Due to this, non-rigid cannot be used in the time critical applications, such as image-guided surgery or the radiation therapy in case of prostate cancer.

The alignment can be more complex due patient repositioning and continuous organ movement [25]. Currently, non-rigid methods are applied with offline settings only which may obstruct an efficient therapy. Hence, non-rigid registration methods need to be revamped for the time-critical applications. In the next section, related work in the literature is discussed to address this issue.



(a) resolution 0 (b) resolution 1 (c) resolution 2 (d) original

(e) resolution 0 (f) resolution 1 (g) resolution 2 (h) original

Figure 2.7: Figure shows the multi-resolution strategy [7]. Here, registration is done using three resolutions. In first row, downsampling is applied with Guassian pyramid while in second row, only downsampling is applied. The number of voxels is doubled with each resolution level and the size of voxel is halved.

2.4 Related work

In simple words, registration is an iterative algorithm which has an objective to achieve optimum parameters to align two images. The methodological strategies like pyramid approach and conjugate gradient enable a registration to perform faster. But these strategies are not sufficient to speed up a non-rigid algorithm for time-critical applications. To address this problem, mainly two strategies have been proposed. First, to use a methodological technique called Stochastic Gradient Descent (SGD) [26]. Second, parallel computing using GPU or multithreading (openMP).

2.4.1 Stochastic gradient descent

The word stochastic means a random pattern or a random distribution. The key idea in SGD [26] is to compute a fast but noisy approximation of the search direction $\tilde{\mathbf{g}}_k := \partial\widetilde{C}/\partial\boldsymbol{\mu}$ (for equation 2.8), by selecting a small random subset from full fixed image sample space. At each iteration, a new random subset is drawn. This is done by the sampler called random sampler in figure 2.2. Thus, a full sample space is avoided to optimize parameters. In computation point of view, instead of using the data size of millions at each iteration, the data size of few thousands can be used. The random sampling reduces the data intensity of the algorithm drastically. But at the same time, this degrades the conversion rate, which means stochastic registration may take more iterations to reach a certain cost function value than a registration using full sampling.

2.4.2 GPGPU

Another solution is to use the high performance parallel computing. Since last decade, GPU and high performance computing (HPC) have been topics of research to address high computation problems. For a typical linear algebra algorithm, by using the GPU, a much higher throughput can be achieved than the CPU with the similar accuracy. Hence, GPU can be a problem solver for the non-rigid registration.

High performance computation approaches such as [27], Haghighi et al. [28] and Plastimatch [29] take advantages of the GPU architecture for parallel computing. In Haghighi et al., only cost function computation is implemented on GPU. In Shamonin et al. [30], CPU and GPU parallelization were implemented in elastix [31]. In this work, Gaussian pyramid and image sampling were implemented on GPU using OpenCL and cost function derivative calculation was optimized using CPU parallelism. In NiftyReg, almost all the registration framework, except the cast function, is implemented on GPU using CUDA.

These implementations are, however, based on gradient descent optimization using full image sampling. In the field of machine learning or neural networks, there exist implementations of SGD on the GPU [32]. However, these methods typically optimize over a large collection of data, and the term stochastic refers to random batches of data instead of random voxels within one image, which is a fundamental difference from a registration point of view. Therefore, an implementation of SGD on GPU is lacking for image registration.

2.5 Registration toolboxes

There are a lot of registration toolboxes with different approaches for registration. There are several open source registration softwares like Image Insight Segmentation and Registration Toolkit (ITK), elastix, NiftyReg, Plastimatch, etc. There are python packages like SimpleITK, SimpleElastix to compute registration in Python. Sometimes, for a particular medical therapy, a new registration algorithm is proposed. For example, a new penalty term is proposed to incorporate a missing structure in an alignment [33]. So, registration toolboxes have a wide diversity with each has its advantages and disadvantages. Recently, SuperElastix [34] is being developed to unify this diversity. The work under this thesis will be included in SuperElastix.

2.5.1 Elastix

Elastix is an open source image registration based toolbox based on widely known Image Insight Segmentation and Registration Toolkit (`ITK`) [35]. Being developed on the top of `ITK`, elastix has all functionality offered by `ITK` in addition of some special enhancements [10]. One of the enhancement is Fast Adaptive SGD (FASGD), which is developed to tackle the problem of nonrigid registration for a time critical application.

In SGD [26], a decaying step size (r_k) is used in the equation 2.8, and is given by,

$$\gamma_k = \frac{a}{(A+k)^{\alpha}}, \tag{2.9}$$

Where parameters a, A and α are constants that typically need to be tuned for the type of data. Since stochastic gradient descent does not have its own stopping criterion, the maximum number of iterations $k_{\max}$ needs to be set as well. Hence, this manual selection of r_k is difficult and can be time consuming which defeats the purpose of being used in a time critical application. Thus, adaptive SGD (ASGD) [22] and FASGD was proposed, where FASGD outperforms ASGD in the computational time. Although FASGD can speed up non-rigid registration, it is only tested in CPU hardware. Thus, FASGD can be further optimized using parallel computing with the GPU.

2.5.2 NiftyReg

In Modat et al [27], NiftyReg toolbox was proposed in which image registration implemented on both CPU and GPU architecture. NiftyReg uses .nii (nifty) format medical images only[2]. NiftyReg uses entire sample space to compute cost function gradient but lower computation time (within a minute) can be achieved with the help parallel architecture of GPU.

2.6 Summary

In this chapter, an image registration algorithm has been explained in brief. Further, two possible solutions, SGD and GPGPU were discussed. From this discussion, elastix or NiftyReg can be chosen as a foundation for this thesis. Next chapter discusses profiling and analysis of these toolboxes in details. Following table 2.1 shows the similarities and dissimilarities (in bold) between elastix and NiftyReg registration toolboxes.

Elastix	**NiftyReg**
• B-Spline Transform	• B-Spline Transform
• NMI measure	• NMI measure
• **FASGD**	• **Conjugate gradient ascent**
• **Runs predefined number of iterations**	• **Stopping criteria with line search**
• **Computation of 5000 samples per iteration(random sampling)**	• **Computation of Millions of samples per iteration(full sampling)**

Table 2.1: Comparison of Elastix and NiftyReg

[2]Using SimpleITK package in python, it is possible to change any medical image format.

3 Profiling and analysis

From literature survey, two possible implementations can accelerate the non-rigid registration. First, a GPU implementation of FASGD in elastix and second, a SGD implementation in NiftyReg. Due to the limited timeline of the thesis, only one implementation can be selected and therefore, profiling and further analysis are needed to evaluate both toolboxes. Both toolboxes were executed using a lung data set, and the accuracy was tested for both. In this chapter, experiment setup, registration pipeline and profiling are discussed. Based on the profiling and analysis, proposed implementations and optimizations are stated.

3.1 Amdahls law of speedup

To estimate the speedup of an application, Amdahl's law of speedup is often used and is given by,

$$S_{overall} = \frac{1}{(1-p) + \frac{p}{s}} \tag{3.1}$$

Where $S_{overall}$ is the overall speedup of the application, p is the portion of the execution time of a function eligible for optimization, and s is the expected speedup of that function. The key idea of this formula is to find out functions in a program which are more time consuming and can be accelerated using an optimization. Such a function (or a part of a function) is often called as a hotspot or a bottleneck. To gain significant overall speedups$s_{overall}$, the value of p should be more.

3.2 Proposed workflow

The code optimization is an iterative process and can be expressed as a workflow shown in diagram 3.1. The very first step is to profile and analyze the original codes to find out hotspots and bottlenecks. The next step is to optimize the code to remove the bottlenecks found in the first step. An optimization can be a software based, hardware based or using both. Sometimes a faster mathematical method can also be used to solve the bottleneck. For example, SGD is a faster mathematical method than standard gradient descent. While GPU computing, OpenMP, and Advanced Vector Extensions (AVX) represent both hardware and software based optimizations.

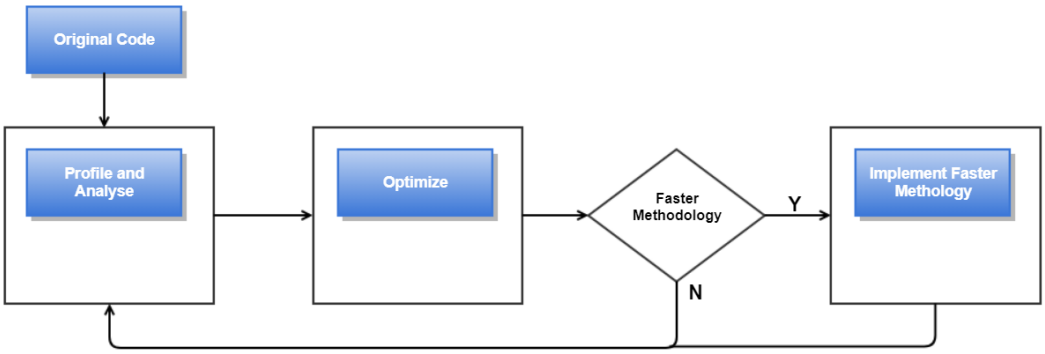


Figure 3.1: Proposed workflow for the thesis

3.3 Setup overview

Hardware

For this thesis, the experiments were run on Intel Xeon CPU E5-1620- 8 cores with 64 GB memory. GPU used in this work is Nvidia Tesla K40c GPU with 12 GB global memory, and peak performance of 4.29 Tflops for single precision[1].

SPREAD Database

For this analysis, computed tomography (CT) lung data from SRPEAD [36] studies has been used. It consists of 19 patient data with the baseline (fixed) and followup (moving) images. Each patient has 100 ground truth points to determine target registration error (TRE) to evaluate the accuracy of the registration.

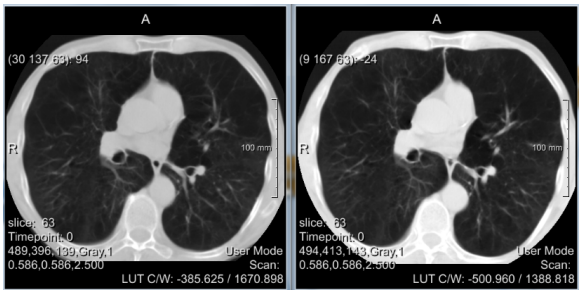


Figure 3.2: Figure shows CT scan for baseline (left image) and followup (right) images.

[1]See appendix for more CPU (A.1) and GPU (A.2) specifications

3.4 Registration pipeline

Figure 3.3 shows the registration pipeline used during this thesis, which includes implementation of a faster non-rigid registration and TRE evaluation of this implementation. As shown in figure 3.3, 100 selected points from a baseline image are transformed using optimized parameters (or control points grid) using the non-rigid registration. Euclidean distance (in mm) is calculated between transformed points and corresponding points in followup images. Such 100 euclidean distances are calculated, and boxplot of these distances represents the quality of the registration. For timing analysis, log files from non-rigid registration are used.

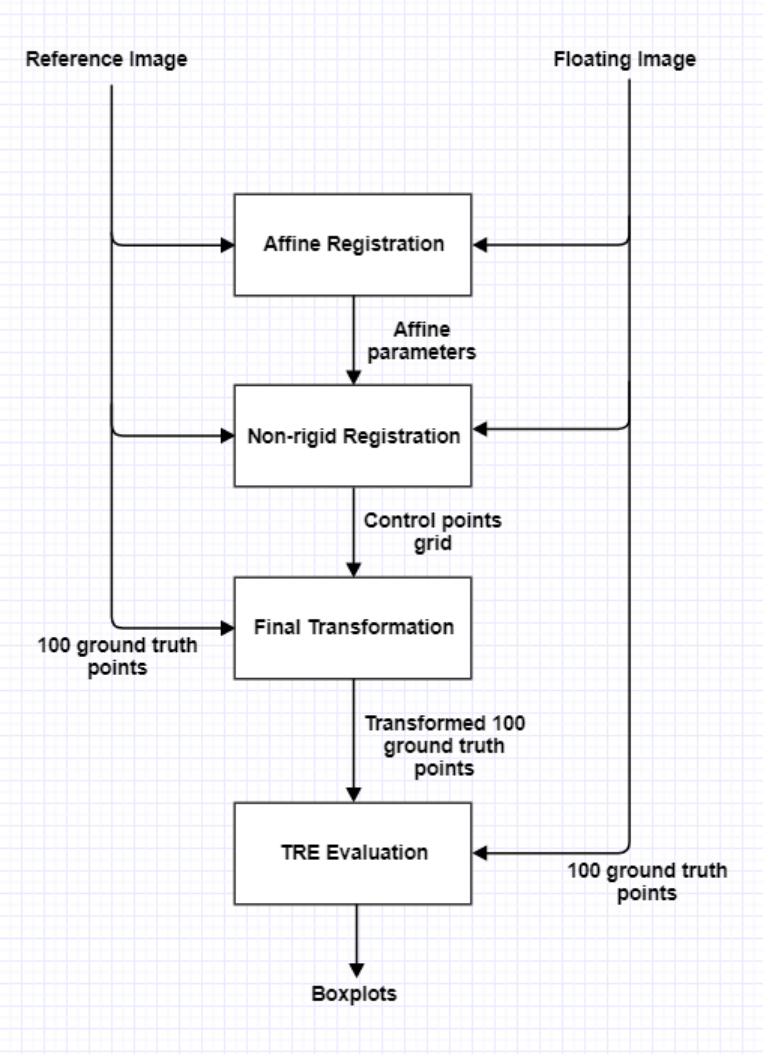


Figure 3.3: Figure shows the registration pipeline used in the thesis. The affine registration is followed by the non-rigid registration, and TRE evaluation is performed on 100 truth points by applying final registration. Default settings for affine registration for NiftyReg were used while for FASGD settings from paper [12] were used.

3.5 TRE evaluation

Both FASGD and NiftyReg were executed using the SPREAD dataset to observe registration quality. The registration settings for FASGD were taken from paper [12]. NiftyReg was executed with the default settings. TRE formula is given by equation 3.2 [23],

$$TRE = T(p) - q \tag{3.2}$$

Where, p and q are the corresponding points in two images to be aligned, and T is a transformation. The final transformation for FASGD was computed by **transformix**, which is a binary application in elastix. Similarly, **reg_transform**, an application in NiftyReg toolkit, is used to apply the final transformation to the image. After final transformation, the euclidean distances between 100 ground truth points were calculated. A ground truth point is a point whose position is known in images to be aligned. TRE is an error and ideally, it should be zero and practically, as small as possible. So a boxplot near to zero means the high accuracy. The figure 3.4 shows boxplots of the FASGD and NiftyReg. FASGD accuracy is slightly better than the NiftyReg (default setting) but not significantly different. This shows that both can be used as a foundation for the proposed implementations.

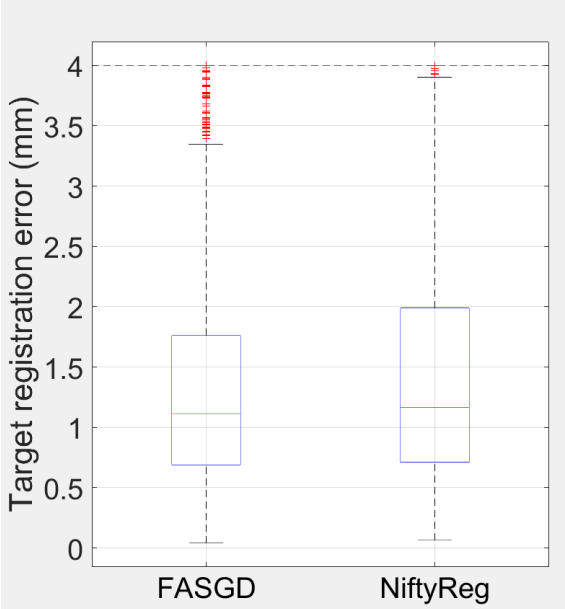


Figure 3.4: TRE evaluation of base codes

3.6 Timing analysis

Both toolboxes can generate log files during registration. These log files have total time taken for the computation[2]. From these log files, it is found out that FASGD takes an

[2]see appendix A.4 for more information about the log files

average 22 seconds while NiftyReg takes around 34 seconds on average for the non-rigid registration using the SPREAD dataset.

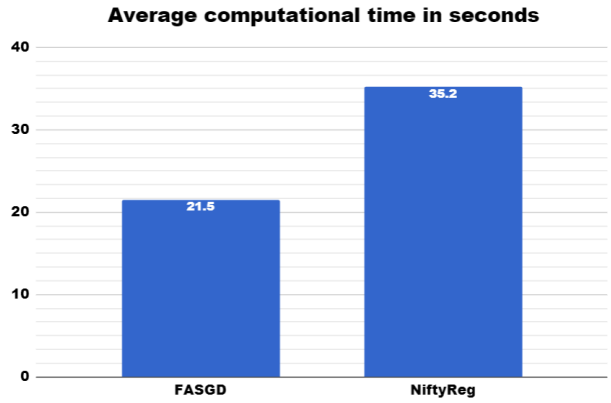


Figure 3.5: Average computational time for elastix and NiftyReg base codes

3.7 Profiling

To find out p in the equation 3.1, profiling is necessary. Without good profiling, it is difficult to optimize a given program, which consists of thousands of lines. Toolboxes like NiftyReg or elastix typically has a wide number of functions. To manually profile such high number of functions is a difficult task in a given time. Further, only a part of a function can be a bottleneck instead of the entire function. Therefore, the profiling tools are must. There are many open source and commercial C++ profiling tools available for CPU and GPU. For this thesis, Valgrind and Intel Vtune Amplifier were used to profile elastix and NiftyReg. Though Valgrind is open source, it takes significant time (few hours) to profile a program which runs around 15 seconds. On the other hand, Vtune profiles the program in the same time as program computational time. Thus, Vtune was chosen for profiling of the CPU in this thesis. Similarly, for the GPU, Nvidia visual profiler (**nvvp**) and **nvpof** were used because of their hardware compatibility.

3.7.1 Elastix-FASGD profiling

Advance Hotspots Analysis of Vtune was used for profiling. From the figure 3.6, it can be concluded that Elastix hotspots are functions from ITK libraries. From Amdahl's Law, if p is a portion of execution time equal to top three hotspots functions, then p is around 16% of the CPU time. For this value of p, overall speedup $S_{overall}$ will be insignificant.

`elastix` has dependency over ITK. This makes elastix more complex from software hierarchy point of view. Due to complex software hierarchy, CUDA (or OpenCL) framework is difficult to add. The optimization of elastix on GPU may be a difficult task given a limited timeline of the thesis as optimization requires a lot of software hierarchy changes.

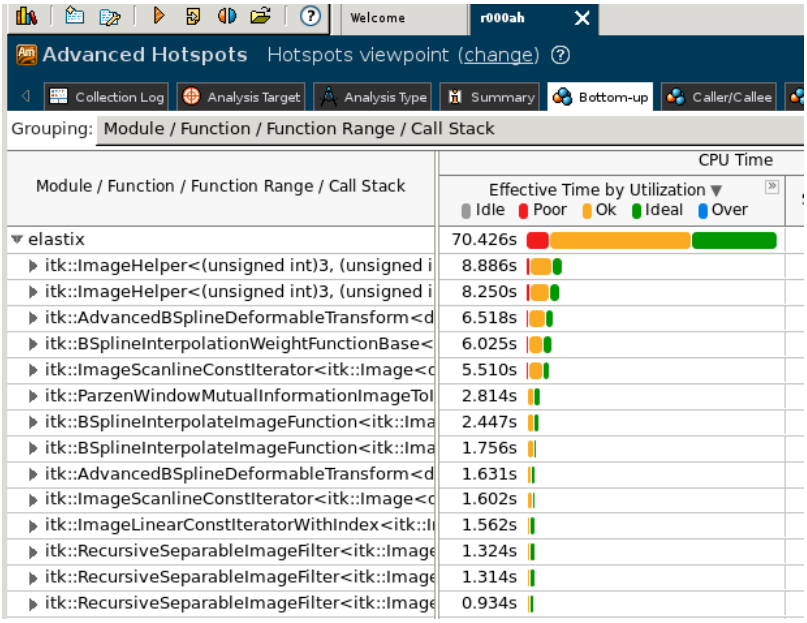


Figure 3.6: Figure shows Advaced Hotspot analysis report of Vtune for FASGD. From report, it can be concluded that ITK based functions are major bottlenecks.

3.7.2 NiftyReg profiling

Hotspots Analysis of NiftyReg (fig 3.7) indicates that 56% of total CPU time is taken by reg_getEntropies function. For such high value of p, $S_{overall}$ can be significant. Further analysis showed that joint histogram filling is the main bottleneck within this function. A GPU implementation can be a possible optimization for the joint histogram filling as proposed in paper [37]. NiftyReg already has pre-existing GPU (CUDA) framework which makes it more suitable than elastix to be used as a foundation of our implementations. Thus, NiftyReg was chosen for this thesis.

3.8 Proposed optimization - iteration 1

From the discussion in the previous section, NiftyReg was chosen as the foundation. Based on the profiling and a rule of thumb, two implementations were proposed as shown in figure 3.8. First, NiftyReg2, an optimization of NiftyReg by porting the joint histogram filling to the GPU. Second, NiftyRegSGD, a SGD implementation of NiftyReg2. Further implementations were proposed in the next optimization iteration.

Porting of the joint histogram filling

This optimization is important to remove the main hotspot present in the current NiftyReg version. From paper [37], two histogram strategies were chosen for this optimization. 1) CUDA global memory atomics, 2) Local histogram reduction, which uses faster shared memory atomics over slower global memory atomics. These methods are

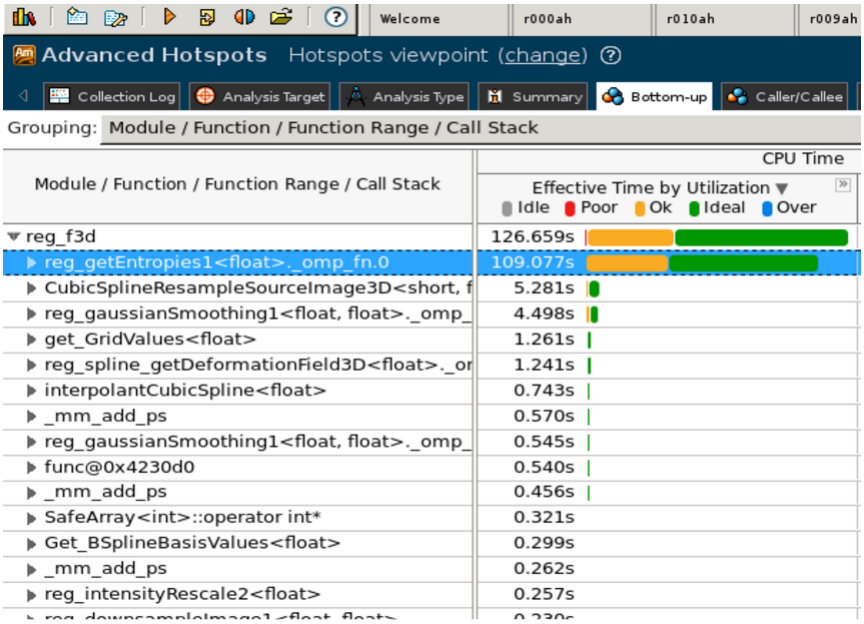


Figure 3.7: Figure shows Advaced Hotspot analysis report of Vtune for NiftyReg. From the report, it can be concluded that the function reg_getEntropies is a major bottleneck.

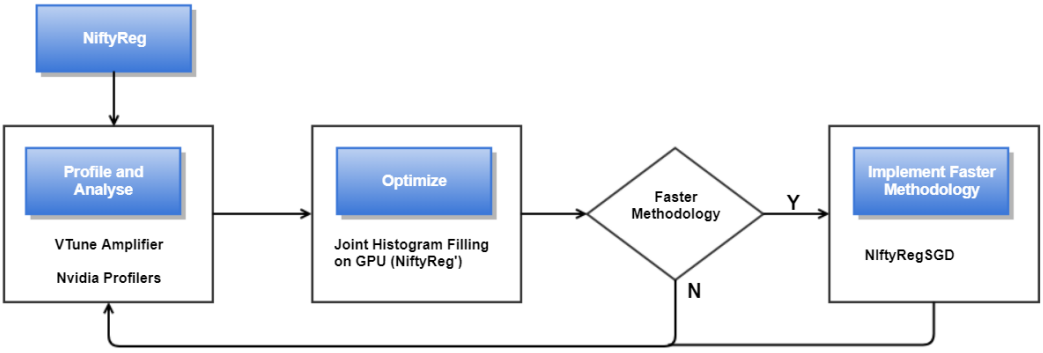


Figure 3.8: Proposed implementations for optimization iteration 1. First iteration involves profiling using profiling tools and, implementations of NiftyReg2 and NiftyRegSGD.

discussed in Chapter 4 in details.

NiftyRegSGD

Current NiftyReg uses full sampling. This means for the SPREAD data, around 13 million samples are used at each iteration. This data intensity can be minimized by using SGD [26] as discussed earlier in chapter 2. So, NiftyRegSGD represents an implementation of the faster methodology using SGD.

Optimization - Iteration 2

The previous optimizations will have mathematical and parallel computing advantages. Since optimization is an iterative process, further profiling and analysis are needed. NiftyRegSGD will be profiled again with Vtune and Nvidia visual profiler to find out new bottlenecks. Based on the profiling, further optimizations will be proposed.

3.9 Summary

This chapter concludes the selection of NiftyReg as the foundation for this thesis by profiling and analysis. Two optimization strategies were proposed to accelerate NiftyReg in the first iteration of optimization cycle. In next chapter, implementations of all proposed optimizations are discussed in details.

4 Implementations

4.1 GPGPU computing

Parallel and distributed computing are main topics of research in high performance computing. The computational problems for data intensive and arithmetic intensive applications like big data analytics and liner algebra has been tackled by parallel and distributed (or cluster) computing. Image registration is not a big data problem. The average size of a lung CT data is around 30 MegaBytes. So here, cluster computing may not be an optimal solution. On the other hand, a GPU with sufficient global memory and optimizations suitable to the GPU hardware, can provide an optimal solution. Following figure 4.1 describes CUDA programming and memory hierarchy for general purpose GPU (GPGPU) programming [38].

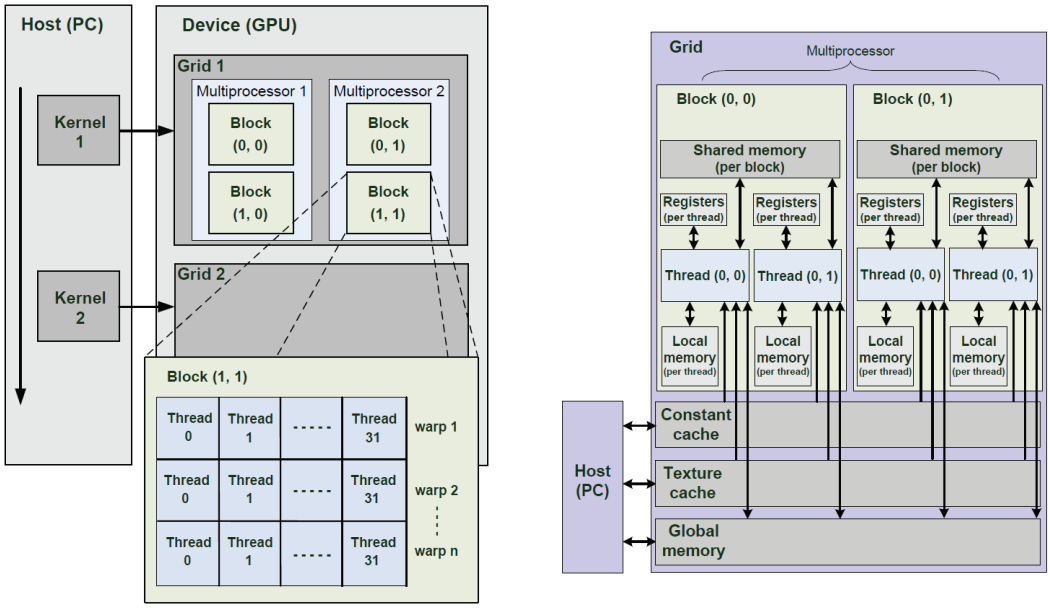


Figure 4.1: GPU memory and programming hierarchy [13].

CUDA programming model consists of grids, blocks, warps, and threads, where a grid resides on the top of the hierarchy and a thread resides at the bottom of the hierarchy, refer figure 4.1. A GPU hardware consists of streaming multiprocessors (SM), and SM consists of streaming processors (SP). So, CUDA blocks are computed on SM while threads are computed on SP. Similarly, GPU memory has a hierarchy. The local memory for a thread, the shared memory for a block and the global memory for a grid.

4.2 Mutual information and joint histogram

Mutual Information is most widely used cost function in the medical image registration. In NiftyReg, normalised mutual information (NMI) is used as the cost function as default and is given by,

$$NMI = \frac{H(F) + H(M(\mathbf{T}))}{H(F, M(\mathbf{T}))} \tag{4.1}$$

where $H(F)$ and $H(M(\mathbf{T}))$ are marginal entropies of the fixed and moving image respectively. $H(F, M(\mathbf{T}))$ is the joint entropy of two images. To calculate a joint entropy, a joint histogram is calculated, which is similar to a normal histogram. But a joint histogram has two dimensions with each dimension represents an image. Let us suppose at coordinate (25,30), a fixed image has voxel intensity 100 and a moving image has voxel intensity 150. Hence the value at position (100,150) of the joint histogram is incremented by one. Hence the joint histogram filling needs two read accesses for input values with an addition read access of the output value at (100,150) to increment that output by one.

For the GPU, above example at coordinate (25,30) is executed by a thread. Similarly, coordinate (25,31) is executed by next thread in the same warp in parallel. This is a basic approach for a CUDA kernel for the joint histogram filling.

4.3 Porting of the joint histogram filling

In the current version of NiftyReg, the joint histogram filling in mutual information has been identified as the major bottleneck. NiftyReg2 is a CUDA based acceleration of this bottleneck to improve performance. Different methods are present in the literature for GPU based histogram computation. The CUDA atomic operations and local histogram explained in Shams et al [37] are selected.

Typically for a GPU based histogram filling, CUDA atomic operations are used to avoid a race condition. A race condition is when two or more threads try to update a memory location. Atomic operations make threads to update a memory location sequentially. In histogram filling, CUDA atomic instruction called **atomicAdd(Addr, value)** is used, where Addr is a memory address to be updated with a value (in this case, value is one).

Local histogram Reduction

As shown in figure 4.3, first local histograms are filled using atomic operations on shared memory. Then local histograms are reduced into a final histogram in global memory. This is similar to the map-reduce algorithm used in Hadoop or distributed computing. This approach avoids slower atomic operations on the global memory. But histogram reduction depends on the size of shared memory, which depends on the GPU model. Tesla 40k has 48KB of shared memory. Typically, a joint histogram has dimensions like 64x64 or 128x128. The size of a joint histogram can depend on an application. For an integer of size 4 Bytes and a joint histogram of size 128x128, the shared memory of

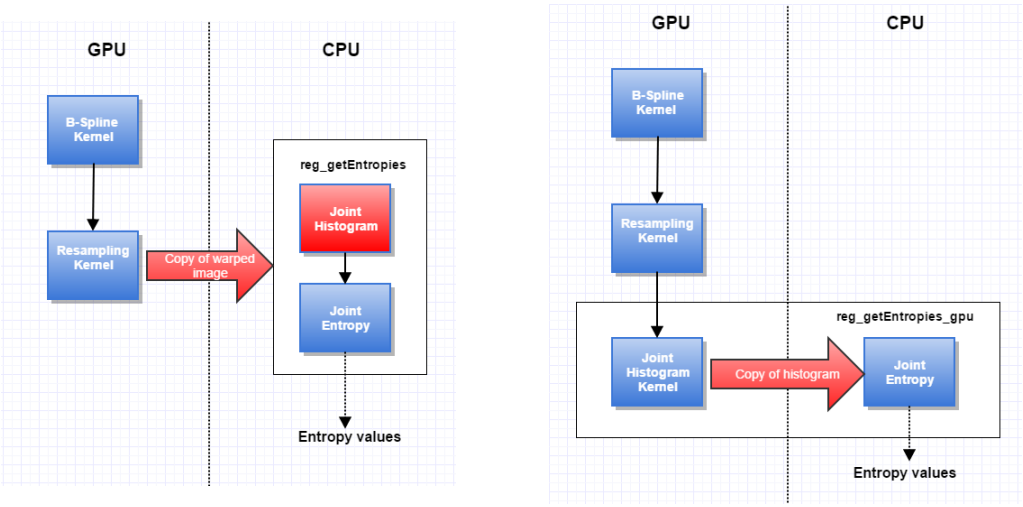


Figure 4.2: Left figure shows implementation of joint histogram in current NiftyReg version. Right figure shows proposed changes for NiftyReg2

$128 * 128 * 4 \approx 64KB$ is needed. For GPUs with lower compute capabilities, this is a limitation of this approach. Hence, in the final version, local histogram reduction is not used. For this implementation, a default 64x64 histogram is used.

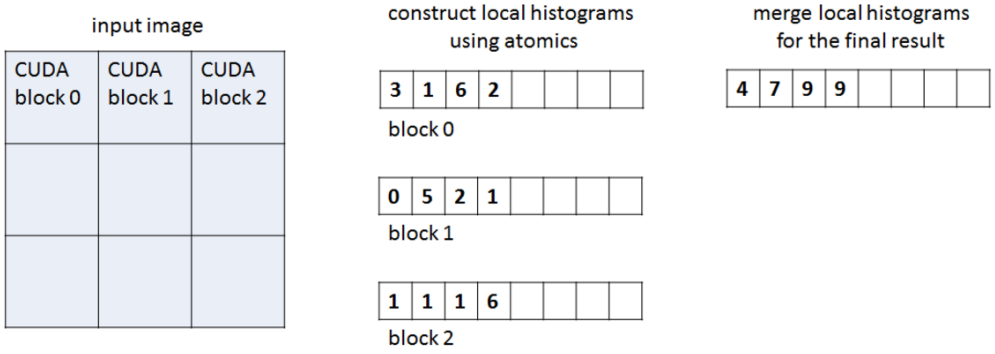


Figure 4.3: Figure histogram reduction method [14]

Global atomic add

The second method is to use atomic operations in the global memory. No shared memory is used in this method and hence, no restriction on the size of a joint histogram size. In addition to this advantage, the global atomic operations for the Kepler architecture (Tesla K40c) has been improved as compared to previous GPU architectures like Fermi, Maxwell [39]. Hence, global atomic operations have been selected over local histogram[1].

[1]Implementation of local histograms reduction is also included but it is turned off. Since this project is publicly available, developers can turn on any method for the joint histogram filling.

4.4 NiftyRegSGD

Once NiftyReg2 is implemented, it is followed by implementation of NiftyRegSGD with decaying step formula [26] given as,

$$Stepsize = \frac{\delta * a}{(A + k)^{\alpha}} \tag{4.2}$$

where δ is the maximum spacing among x,y and z axes. a, A and α are constants. k is the current iteration and k_{max} is the maximum number of iterations. The line search is removed from this implementation. In NiftyRegSGD, interpolation, cost function, transformations and voxel gradient calculations will be computed with a small subset of the data. Original NiftyReg has a line search which makes iterative registration to stop before the maximum number of iterations. This line search is removed for NiftyRegSGD. So k_{max} is the only stopping criteria for NiftyRegSGD.

Masking and sampling

In medical imaging, sometimes only an organ can be a point of interest. For example, in a CT scan of a thorax, only left lung is a point of interest. In such cases, a mask is used. A mask is a boolean image having the same size as the input image. In a mask, only valid co-ordinates have boolean value **true** while others have **false** value. Thus, only the left lung is considered in the image registration or similar medical image processing.

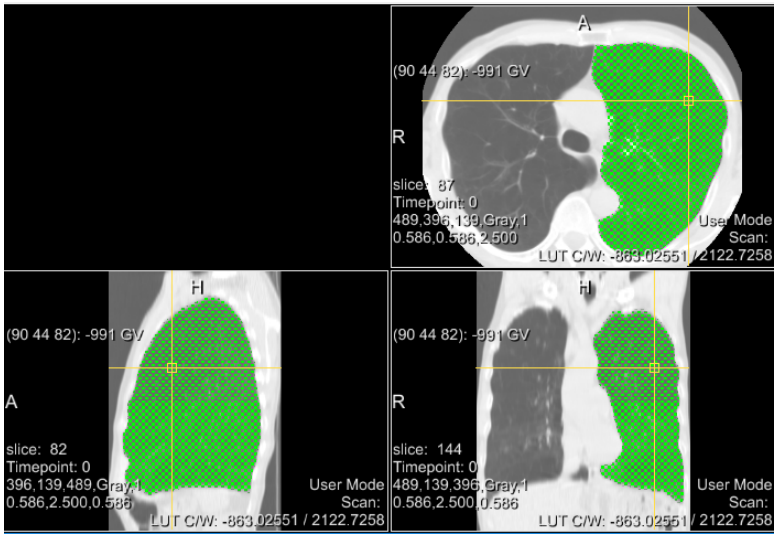


Figure 4.4: Figure shows a mask for the left lung. Co-ordinates at green colour are valid or true. Thus only green co-ordinates are used in the registration. This is similar to using a subset of an image.

Thus, this concept of a mask is extended in order to create a random subset (or a random mask). Hence a random sampler is implemented in NiftyReg to create a random mask at each iteration. This is implemented as shown in figure 4.5.

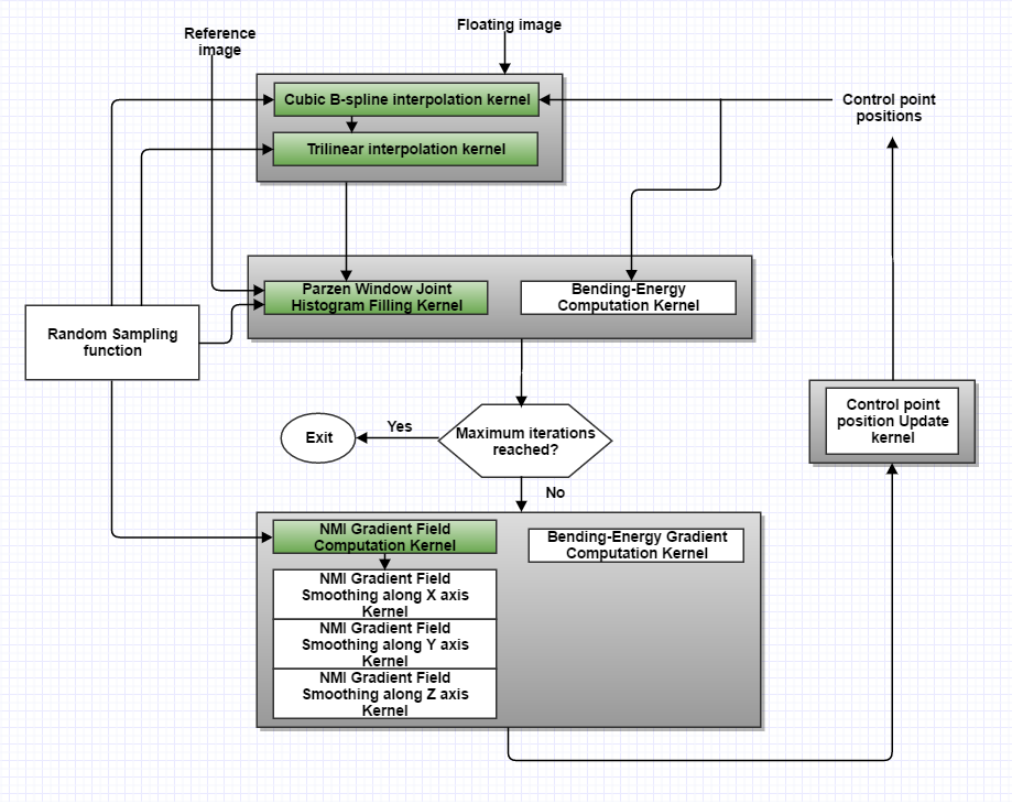


Figure 4.5: Figure shows a NiftyRegSGD changes in NiftyReg. A new CPU based random sampling function is introduced and modules in green colours are now using a subset of input data.

As shown in figure 4.5, the random sampling function will create a fresh random subset of data at each iteration for SGD. Original NiftyReg has two different implementations of the mask, one each for CPU and GPU. For the CPU, a mask has the same size as the fixed image and has boolean values as discussed earlier. For the GPU, the size of a mask is equal to the number of valid coordinates and have values same as the position of valid coordinates. For example, a fixed image of size ten is used, and only 0,5 and 7 are valid coordinates. So mask for CPU will be $\{1, 0, 0, 0, 0, 1, 0, 1, 0, 0\}$. But GPU mask will be $\{0, 5, 7\}$ of size 3. If no mask is used, then CPU and GPU mask will have the same size. In this case, GPU mask will be $0, 1, 2, \cdots, \Omega_F$, where Ω_F is the image space of the fixed image. Since we are using the GPU version of NiftyReg, the GPU mask will be our point of interest. Creating a different mask for GPU reduces the size of the mask which utilizes less GPU memory. It only computes for a valid point, this avoids the computation for a invalid point. If a random subset is applied to the cost function equation 2.1, then $x_i = \Omega_r$ and $\Omega_r \subset \Omega_F$, where Ω_r is a random mask created by the random sampling function. Such a random mask is used as an input to CUDA kernels shown in green in figure 4.5. A random GPU mask is implemented using the read-only texture memory. The advantage of the texture memory is to have a faster read access

than the global memory.

4.5 Random chunk sampling

Although, the computational intensity is reduced due to SGD, there is still a memory bottleneck with the random sampling. Suppose, a GPU mask has value $\{100, 598, 12, 1111, 15634, 359, \cdots\}$. If only first four values are considered, 4 CUDA threads in a warp will access 100, 598, 12 and 1111 co-ordinate positions. These are random memory locations to be accessed by parallel threads, and these memory locations will not be accessed in parallel as shown in figure 4.6. Instead, threads will access them sequentially, and memory bandwidth will be wasted. GPU has 128-bit cache line, which means in case of a coalesced memory access, 32 floating values (4 bytes each) can be accessed in parallel for a warp. But since only one value (out of 32) is used, other 124 bytes are wasted for each warp in a worst-case scenario.

To solve this, a novel random chunk sampling is proposed. In this strategy, every first sample out of 32 samples is created randomly, followed by 31 samples adjacent to this first sample. So, the GPU issues only a single 128-byte load. This enables 32 threads in a warp to have a coalesced memory access, which results in faster memory access and increase in GPU throughput. Refer figure 4.6.

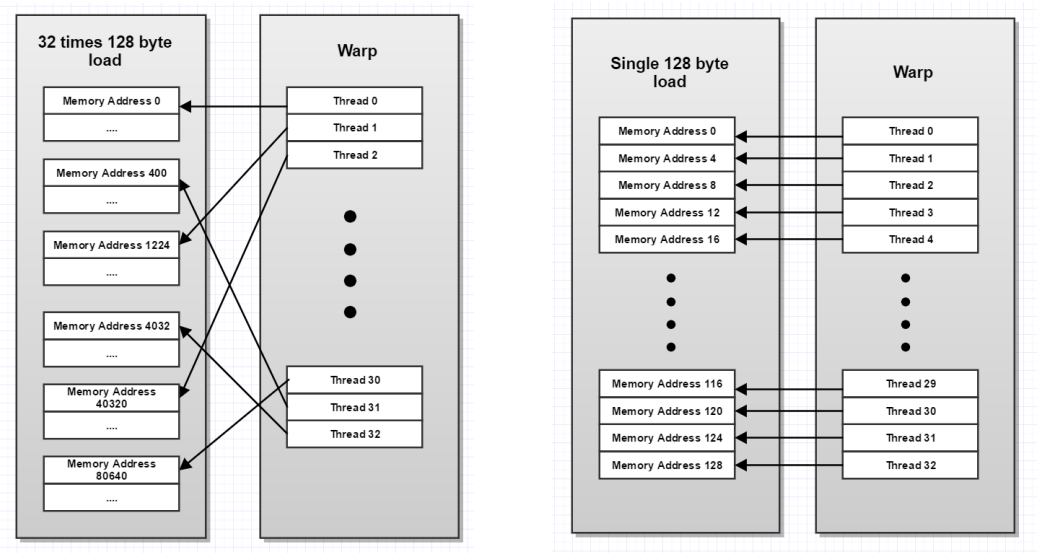


Figure 4.6: Non coalesced(left) and coalesced (right) memory access

This novel random chunk sampling may reduce the randomness property of the subset but improves the memory access for the GPU. To visualize a set of random samples, a GPU mask is stored at an iteration, and a nifty mask image is created using NiBabel python package [40]. This mask image is visualized in MeVisLab [41] and shown in figure 4.7 for both sampling strategies.

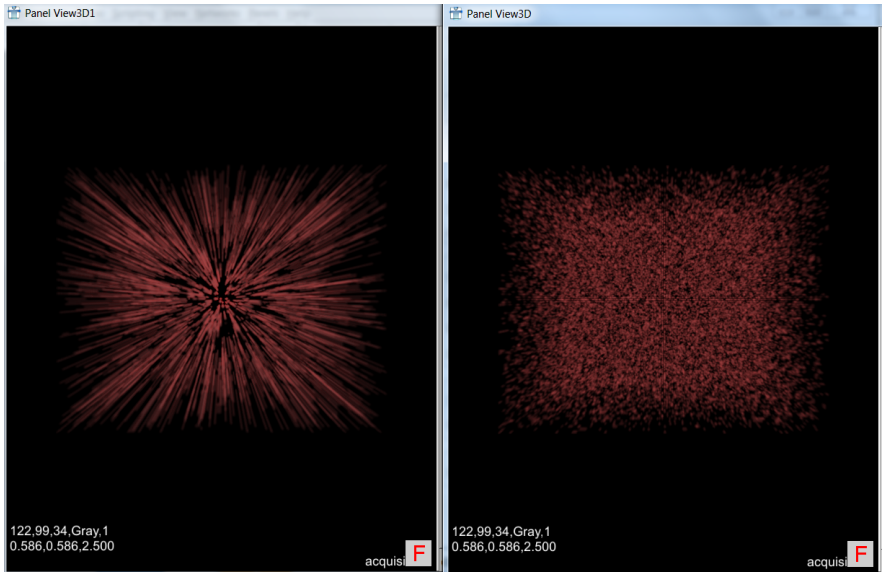


Figure 4.7: Left figure shows the random chunk sampling for GPU hardware and right figure shows the naive random sampling.

4.6 Summary

In this chapter, NiftyReg2 and, NiftyRegSGD with a naive random sampler and random chunk sampler are proposed. Experiments and results will be discussed in the next chapter in details. For NiftyRegSGD, a tuning is required. A tuning in registration is to select the user-defined parameters to get better registration quality. This tuning is also explained in next chapter.

5 Results and discussion

This chapter discusses the performance of all proposed methods using the SPREAD dataset which was used for the profiling. Boxplots are used to evaluate registration quality using 100 truth points, and timing analysis is done using the log files.

5.1 NiftyReg2

The first optimization is to accelerate the joint histogram filling on GPU as shown in figure 4.2. In original NiftyReg, a warped image is computed on the GPU, but for a joint histogram filling, this warped image is transferred to the CPU. In proposed method, transfer of a warped image (of size around 13 millions) is avoided and transfer of a joint histogram (of size around 4 thousands) is used. This joint histogram is further needed to calculate entropy values using the CPU. So, this method not only accelerates a histogram filling but also uses smaller and faster data transfer (cuda memcpy). This implementation speeds up NiftyReg 2.7 times using default settings. The average computational time for original NiftyReg is 36 seconds and for NiftyReg2 is 13.02 seconds.

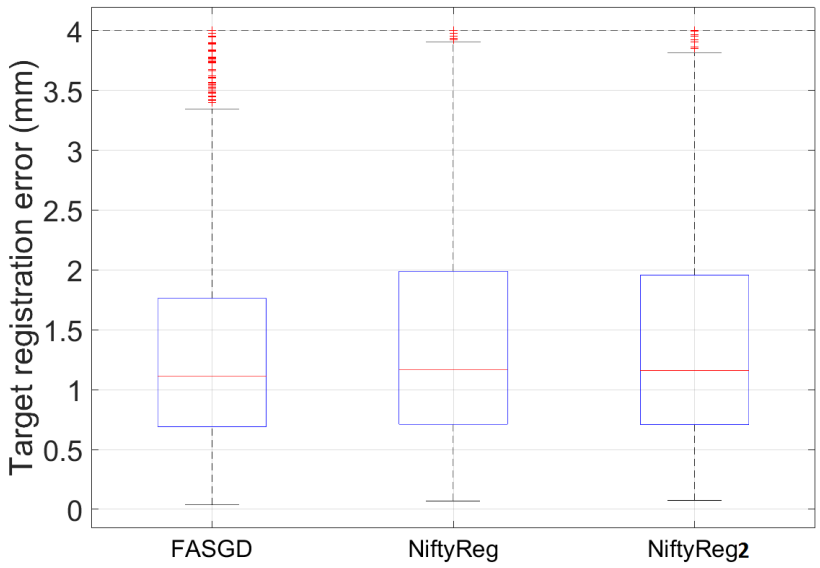


Figure 5.1: Boxplots of FASGD, NiftyReg and NiftyReg2.

Above figure 5.1 shows the TRE evaluation of the NiftyReg2. From the boxplots, NiftyReg2 has similar registration quality as compared to the original NiftyReg while NiftyReg2 runs 2.7 times faster than original and FASGD.

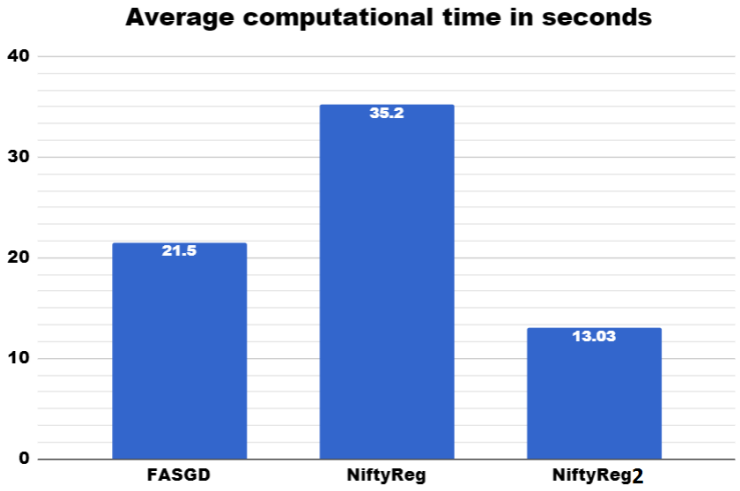


Figure 5.2: Average computational time of FASGD, NiftyReg and NiftyReg2 in seconds.

5.2 NiftyRegSGD

NiftyReg2 is extended using decaying step size given by equation 2.9 using a random subset of data generated by the random sampler. Random sampling function is implemented with user-defined random sampling percentage, s. Also in equation 2.9, a, A, and α are user defined. The stopping criteria k_{max} is also user defined, where $k = \{1, 2, 3, \cdots, k_{max}\}$. Hence, the selection of these parameters is important to get optimal registration accuracy. This selection can be called as registration tuning.

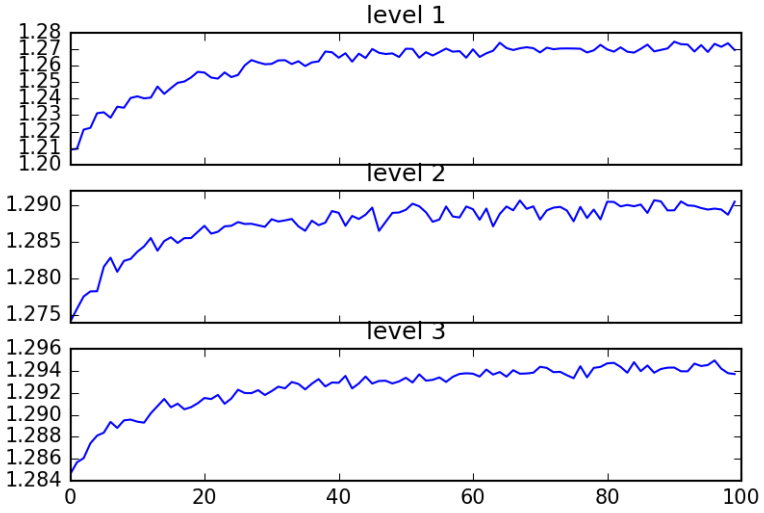


Figure 5.3: Figure shows cost metric reaches optimum value after 100 iterations for multiresolution approach. Due to the stochastic nature, the cost plot becomes noisy.

The values for A and α were chosen as 20 and 0.90 as stated in this paper [22]. Before the tuning, NiftyRegSGD was validated with cost plots, shown in figure 5.3 using the decaying step size, shown in figure 5.4. The noisy nature of the cost function represents a stochastic property of the optimizer. The expected cost plots of NiftyRegSGD were obtained. Before evaluating NiftyRegSGD further, random chunk sampling is evaluated first.

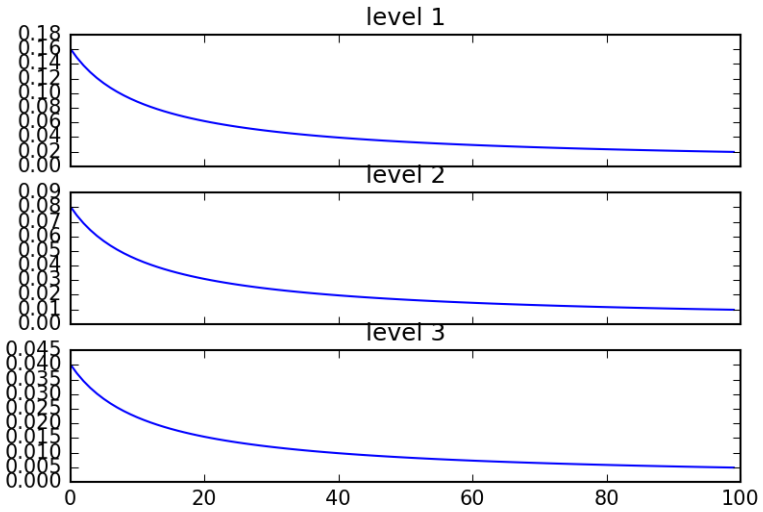


Figure 5.4: Figure shows the decaying step size used for cost plot in figure 5.3.

5.3 Random chunk sampling

The plots of throughput (giga voxel per seconds) for B-spline kernel (figure 5.5) and resampler kernel (figure 5.6) were plotted for both sampling strategies for sampling percentage, $s = [15, 35, 65]$. Typically, the modern CPUs and GPUs have a roof line model for their performance. From figure 5.5, for the naive random sampling, throughput is lowest for all levels at $s = 65$. This shows the memory bottleneck, which is discussed in 4.5, which introduces more delay at a higher sampling percentage than a lower sampling percentage. The throughput for B-spline using the random chunk sampling is increased as compared to naive random sampling. For the random chunk sampling, increasing throughput is observed at $s = [15, 35]$ with higher resolution. But for $s = 65$, throughput is same for all three levels. The reason for this can be 1) GPU is operating at peak performance for this kernel, 2) there may be some other bottleneck present in this kernel.

Similarly for resampler kernel, the increase in GPU throughput is observed in case of random chunk sampling, refer figure 5.6. The performance of GPU for resample kernel is as expected in the roof line model, which means the throughput increases as size of the data increases.

Regarding the runtime, the speedup is different for different sampling percentage.

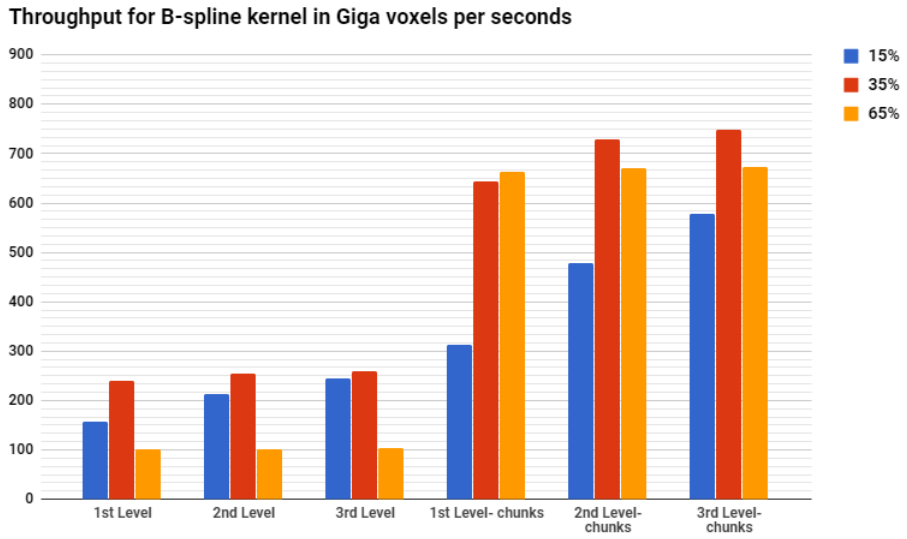


Figure 5.5: Figure shows the average throughput for B-spline kernel for different levels. '-chunks' represents the throughput for the random chunk sampling.

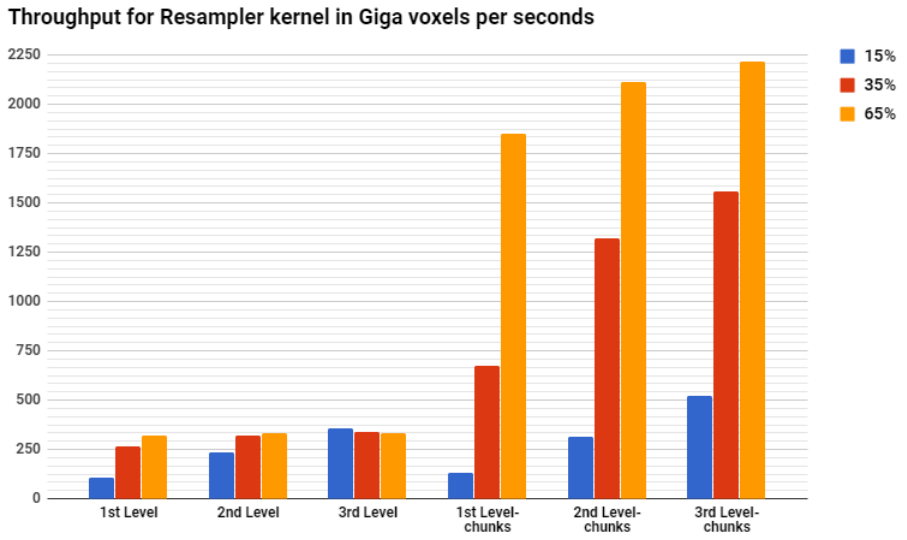


Figure 5.6: Figure shows the average throughput for resampler kernel at different levels for different sampling percentage. '-chunks' represents the throughput for the random chunk sampling.

The average computation times are plotted for both kernels for all levels using both sampling strategies, as shown in figure 5.7 and 5.8. From both figures, it is observed that the speedup increases with higher resolution and higher sampling percentage. For the analysis, only b-spline and resampler kernels were profiled because these two are the most time-consuming kernels which are dependent on the random sampling.

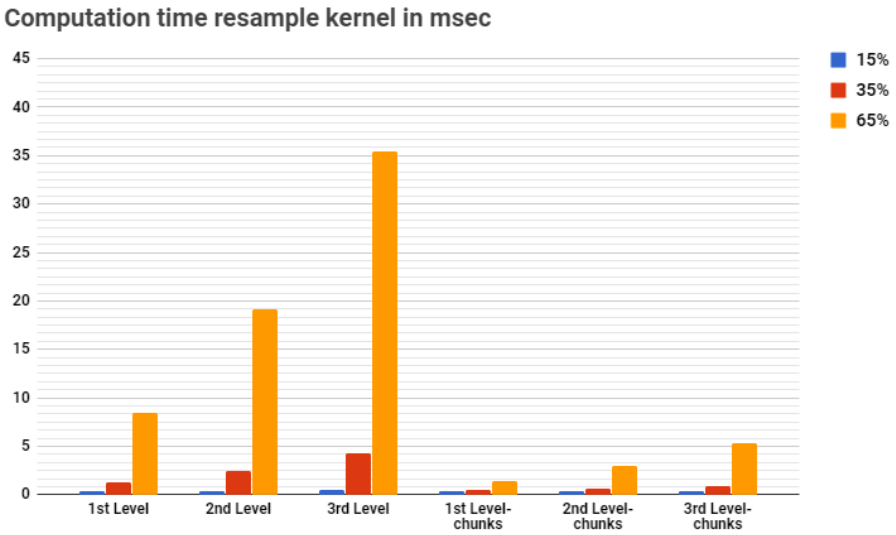


Figure 5.7: Figure shows the average computation time for B-spline kernel at different levels for different sampling percentage for both sampling strategies. '-chunks' represents the computational time for the random chunk sampling.

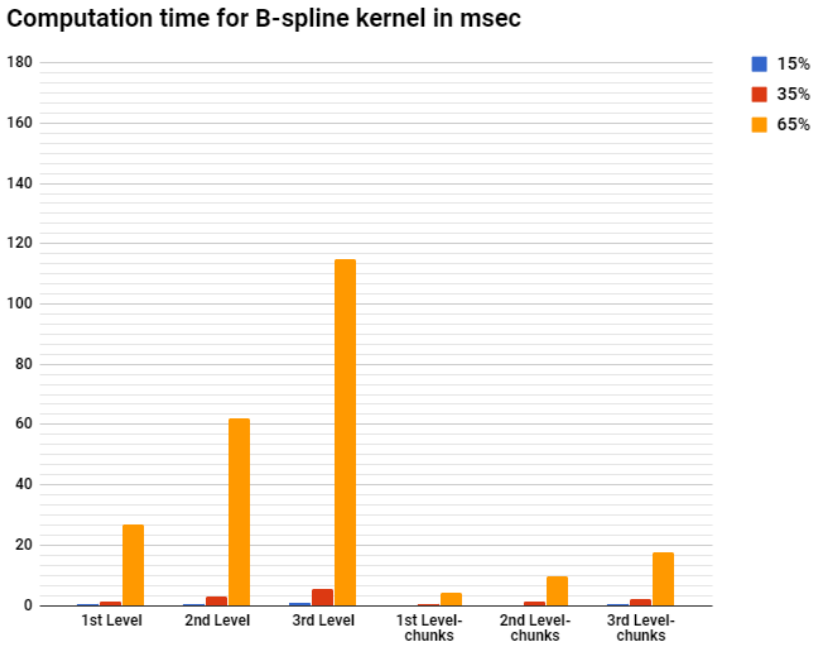


Figure 5.8: Figure shows the average computation time for B-spline kernel at different levels for different sampling percentage for both sampling strategies. '-chunks' represents the computational time for the random chunk sampling.

5.4 Tuning

The registration tuning was done using the random chunk sampling. As explained in the section 5.2, tuning is the selection of s, a and k to obtain better registration quality. So experiments were performed for the different values of user-defined parameters. These ranges of values are shown in the table5.1. For the tuning, SPREAD data (19 patients) is divided into two datasets, train data (10 patients) and test data (9 patients). The tuning was done using the train data.

Parameters	Values
a	[0.15,0.25,0.35]
s	[10,20,30.....,300]
k	[5,10,15,20,.....80]

Table 5.1: Table depicts the range of values for parameters used for tuning.

The tuning took more than 48 hours. The figure 5.9 shows the plots of median TRE values for the train data against the sampling percentage and computational time. Each black dot represents a median TRE value at an iteration value k and values for k are shown in table 5.1.

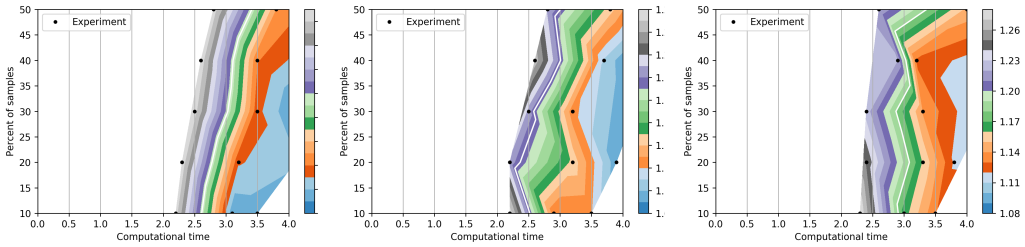


Figure 5.9: The median of TRE are plotted for $a = [0.15, 0.25, 0.35]$ (from left to right) using the train data. For $a = 0.25$, NiftyRegSGD performs fastest with similar accuracy as FASGD.

The another purpose of this tuning is to find the fastest setting for NiftyRegSGD to match the accuracy of elastix-FASGD . So, the white curve in the above plots represents FASGD accuracy (i.e., the median of TRE) for the same train data. From the plots, it is observed that for $a = 0.25$, NiftyRegSGD performs fastest to match FASGD accuracy. The higher values of a are avoided. Because for these values registration may become unstable.

Hence, a similar contour plot for $a = 0.25$ is shown in figure 5.10. In this plot, the leftmost point on the white curve was chosen as the final setting for the registration. Because at this point, NiftyRegSGD is performing fastest to match FASGD accuracy. The table 5.2 shows the fastest setting for the registration.

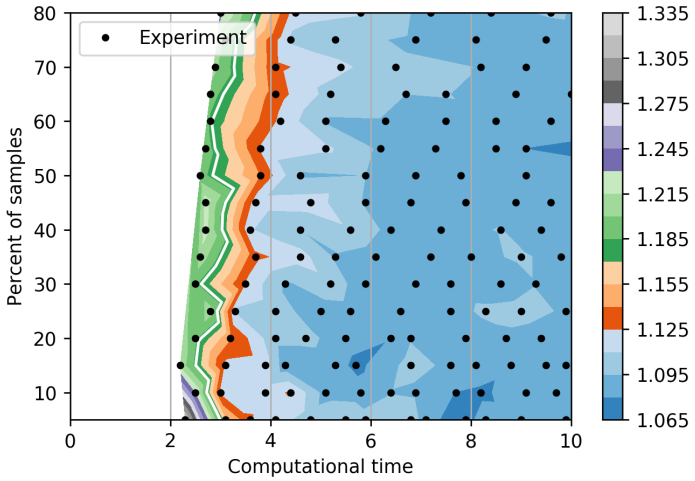


Figure 5.10: Contour plot of the median TRE is plotted for $a = 0.25$.

5.5 NiftyRegSGD with fastest settings

The final fastest settings which are shown by table 5.2 are applied to the test data to find out registration performance of NiftyRegSGD.

Parameters	Values
a	0.25
s	15
k	20
A	20
α	0.90

Table 5.2: Table shows the fastest registration setting for NiftyRegSGD to match FASGD accuracy.

The figure 5.11 shows TRE boxplots for all the methods using train and test data. The train data accuracy is plotted for NiftyRegSGD with and without random chunk sampling.

From the figure 5.11, the train data has slightly better accuracy than test data in case of NiftyRegSGD with or without random chunk sampling. The accuracy of both NiftyRegSGD has found to be better than NiftyReg and NiftyReg2 for both data sets. On FASGD point of view, accuracy for NiftyRegSGD is better in case of train data while FASGD has better accuracy for test data than both NiftyRegSGD.

The figure 5.12 shows accuracies for all methods using overall SPREAD dataset. The accuracies for both NiftyRegSGD are same as the accuracy of FASGD and, are better than NiftyReg and NiftyReg2.

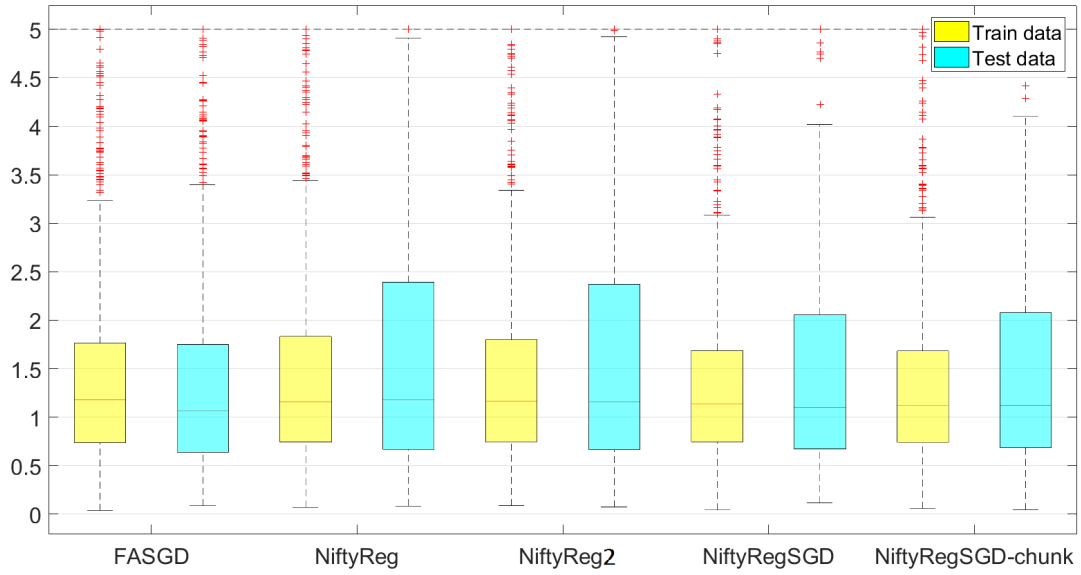


Figure 5.11: Figure shows boxplots for all methods using train and test data sets.

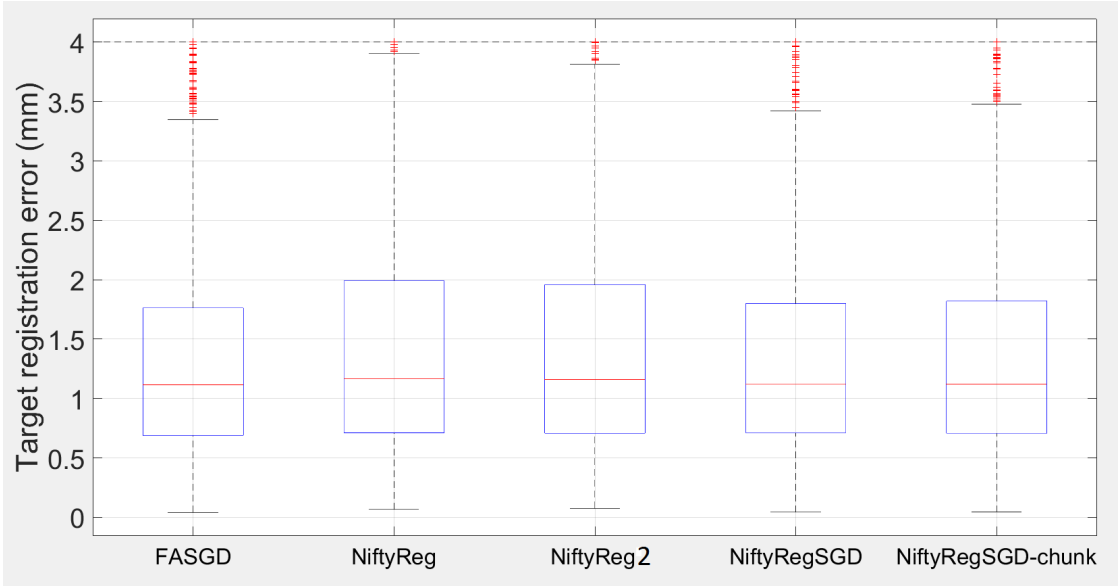


Figure 5.12: Figure shows boxplots for all methods using SPREAD data sets.

Signed rank test and final timings

To measure the performance of TRE with respect to FASGD, Wilcoxon signed rank test is performed in Matlab for train and test data separately (table 5.3) and on the whole dataset (table 5.4). Wilcoxon signed-rank test is used for the null hypothesis. The null hypothesis is a statistical test to check if there is no significant difference between two sets (here, TRE). This test returns two values p and h, where p is probabilities in

the tail of the distribution and h is a logical value. If $p < 0.05$, the test rejects null hypothesis and for $p >= 0.05$, the test fails to reject the null hypothesis. In short, for values $p >= 0.05$, there is no significant difference between two datasets.

	NiftyReg		NiftyReg2		NiftyRegSGD		NiftyRegSGD-chunk	
	Train	Test	Train	Test	Train	Test	Train	Test
p-value	0.06	$9 * 10^{-6}$	0.062	$2 * 10^{-5}$	0.26	0.10	0.15	0.03
Hypothesis	✓	✗	✓	✗	✓	✓	✓	✗

Table 5.3: Table shows Wilcoxon signed rank test result for each methods as compared to FASGD for train and test dataset.

Table 5.3 shows the result for the signed rank test. For the train data, the null hypothesis holds, which means there is no significant difference in accuracy between all NiftyReg implementations and FASGD. For test data, NiftyRegSGD with naive random sampling has no significant difference in accuracy. NiftyRegSGD with chunk sampling has a marginal difference wrt FASGD, but has a much higher p value than original NiftyReg and NiftyReg2. This means NiftyRegSGD with chunk sampling is not as significantly different as original NiftyReg and NiftyReg2.

	NiftyReg	NiftyReg2	NiftyRegSGD	NiftyRegSGD-chunk
p-value	$1.1 * 10^{05}$	$2.3 * 10^{05}$	0.88	0.68
Hypothesis	✗	✗	✓	✓

Table 5.4: Table shows Wilcoxon signed rank test result for each methods as compared to FASGD for overall dataset.

Table 5.4 shows signed rank test results for all methods using overall dataset. Original NiftyReg and NiftyReg2 have lower p values and null hypothesis does not hold in these cases. Both NiftyRegSGD have higher p values and hold the null hypothesis. So using fastest settings, our implementation matches the accuracy of FASGD. The figure 5.13 shows the computational time for all methods. NiftyRegSGD with random chunk sampling performs fastest with 2.8 seconds with similar accuracy as FASGD.

5.6 Discussion

With the fastest settings from table 5.1, NiftyRegSGD performs almost ten times faster than FASGD with similar accuracy for the overall dataset. Using random chunk sampling, 1.6 times speedup is observed in NiftyRegSGD using fastest settings. The accuracy for the test data is slightly less according to the signed rank test results from table 5.3. This can be improved by using some extra iterations. From figure 5.10, accuracy increases with number of iterations. So, instead of using the leftmost point on the white curve, a leftmost point with the highest accuracy (in the whole contour) can be chosen. This setting will increase computation time (which is still faster than FASGD), but it can improve the accuracy even better than FASGD.

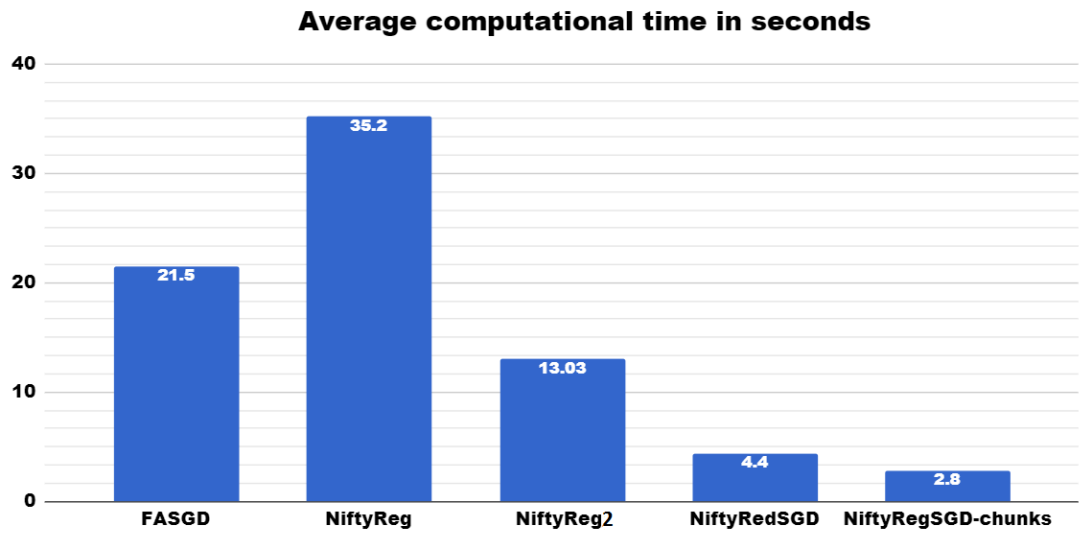


Figure 5.13: Figure shows the plot for computational time for all methods.

5.7 Summary

In this section, results and evaluations of all proposed methods are discussed. In general, boxplots and timings represent the main evaluation criteria. The contour plots were plotted to justify the selection of optimal registration parameters using tuning experiment. CUDA kernels were evaluated using the throughput plots. In the next chapter, the conclusion is stated using the result discussed in this section. Also, the future recommendations are proposed in the next section.

6 Conclusions and future work

In the previous chapter, results and performance of all implementations were discussed. This chapter discusses the conclusion and future recommendations for this work. In this thesis, a high performance non-rigid registration, NiftyRegSGD is implemented using stochastic gradient optimizer. The thesis started with the optimization of original NiftyReg to remove the present bottlenecks which are identified by the profiling. The next step was to implement stochastic gradient descent, NiftyRegSGD. Further, NiftyRegSGD was improved with optimized memory access using the random chunk sampling.

6.1 Conclusion

For our first method NiftyReg2, 2.7 speedup was observed with similar accuracy as the original NiftyReg. Typically, speed up and throughput for a registration depends on a number of iterations k and sampling percentage s. From the plots from 5.5 to 5.8, it can be concluded that the higher value of s results into the higher throughput and speedup using our random chunk sampling. Our implementation is thus following a typical roofline model for a modern CPUs or GPUs using the random chunk sampling.

	FASGD	**NiftyReg2**	**NiftyRegSGD (chunk)**
Resolutions	3	3	3
Transform	B-spline	B-spline	B-spline
3D Parameters	$\approx 1k/6k/35k$	$\approx 6k/35k/230k$	$\approx 6k/35k/230k$
Metric	NMI	NMI	NMI
Optimizer	FASGD	Conjugate gradient	SGD
Step size	adaptive	line search	Equation 4.2
Iterations	500/500/500	$\approx 54/64/169$	20/20/20
Sampler	random	full	random (chunk)
Samples	0.04%	100%	15%
Total time in sec	21.2	13.02	4.4 (2.8)

Table 6.1: Algorithmic and settings overview of the various registration methods.

The table 6.1 compares all registration methods used in this thesis. The main differences lie in optimization technique and number of samples used in the registration. CPU based FASGD uses around 5000 (0.04%) samples at each resolution. On the other hand, NiftyReg2 uses full 12 million (100%) sampling. Our SGD implementation NiftyRegSGD uses 15% samples. As compared to FASGD, this sampling percentage is much higher. But, a GPU can afford such a high sampling percentage. This high percentage has

two advantages, one, GPU performs at higher throughput, and second, a cost function converges faster with a higher percentage.

Our random chunk sampling has improved the memory access. GPU CUDA kernels have higher throughput as compared to naive random sampling. The improvement in throughput is found to be different with different resolution and sampling percentage. The throughput plots for the random chunk sampling is following the roofline model for modern computer and GPU architecture.

Thus, both mathematical method and parallel computing have been implemented to achieve a fast non-rigid medical image registration NiftyRegSGD, which performs almost ten times faster than recently proposed FASGD while maintaining a similar accuracy.

6.2 Future recommendations

Extensions

To have the better accuracy, NiftyRegSGD depends on a tuning of the registration parameters. Such tuning is difficult and time consuming. The tuning done during this thesis took more than two days to complete. Even though NiftyRegSGD performs within few seconds, this tuning defeats the main purpose of being near real time. FASGD is proposed to tackle this problem. Hence, the next step can be NiftyRegFASGD, which is an implementation of FASGD in NiftyReg to avoid time consuming tuning.

Currently, elastix-FASGD and NiftyRegSGD use k_{max} as a stopping criterion, which means tuning is still needed to chose k_{max}. The registration should take more iterations when there are more deformations between two images. Thus, k_{max} depends on the input data. The images of a certain patient can take a longer time to align than images of another patient. On the other hand, the original NiftyReg and our NiftyReg2 use line search as a stopping criterion. Using a line search, registration can be terminated smartly without tuning k_{max}. The word smartly means the line search lets a registration run until the cost function converges. Therefore, an extension of a line search in FASGD is highly recommended in both elastix and NiftyRegSGD to use a fast non-rigid registration in online therapy.

NiftyRegSGD only supports one to one registration. In some medical application, many to one registration is needed, in which there are one reference image and many moving images. Our implementation also does not support masking. This means an extension can be implemented to create random sampling from a given mask. These can be extensions to our implementation.

Further validations

Our implementation has been evaluated using only one dataset and TRE. It is recommended to evaluate present NiftyRegSGD using different dataset and different evaluation criteria. For example, Hammer data [42] can be used for registration where an image segmentation is mapped. This quality of mapping can be measured by the dice overlap. NiftyRegSGD can also be validated using different cost functions as well. In this thesis, only NMI is used as the cost function.

The accuracy of random chunk sampling is tested using TRE only. This thesis does not answer the validity of the random chunk sampling in details. Further statistical testings can be used to compare random chunk sampling to naive random sampling to validate our new sampling strategy.

Optimization iteration-3

By further analysis and profiling on NiftyRegSGD, we have found that voxel gradient smoothing is performed on whole image space and only a few the voxel gradients are used to compute the node gradients. For example, at highest resolution, around 13 million voxel gradients are smoothed but, only 230k gradients are used for the node gradient computation. This approach is wasting the GPU memory and computational time. This can be solved by smoothing with downsampling. In downsampling, only 230k voxels are smoothed instead of 13 million. In programming point of view, this strategy can combine two CUDA kernels (voxel gradient smoothing and node gradient calculation) into a single kernel and may decrease the computation time.

In our implementation, only joint histogram is implemented on GPU while other kernels were used as they are. These kernels were developed using NVidia 8800 GTX GPU [27] which has 1 compute capability. In this thesis, we have used Tesla 40k, which has 3.5 compute capability. The design of these kernels needs to be tailored for the latest GPU architecture. Thus, NiftyRegSGD with above recommendations can be a possible solution for the non-rigid registration for a time critical application. Our paper abstract related to this thesis has been accepted by the SPIE medical imaging conference and the final paper will be published in February 2018.

Bibliography

[1] "X-ray wikipedia," https://en.wikipedia.org/wiki/X-ray.

[2] "Tabulating machine wikipedia," https://en.wikipedia.org/wiki/Tabulating_machine.

[3] "CT scan sketch by godfrey hounsfield," https://en.wikipedia.org/wiki/Godfrey_Hounsfield.

[4] "Philips website," https://www.usa.philips.com/healthcare/product.

[5] "ADAPTNOW-high-precision cancer treatment by online adaptive proton therapy," https://www.lumc.nl/org/radiologie/research/LKEB/969723/ADAPTNOW/.

[6] L. Ibanez, W. Schroeder, L. Ng, and J. Cates, "The itk software guide," 2005.

[7] S. Klein and M. Staring, "Elastix manual," pp. 1–65, 2015.

[8] "Museum of cultural history, oslo," http://www.khm.uio.no/english/visit-us/viking-ship-museum/exhibitions/gokstad/.

[9] "Spline wikipedia," https://en.wikipedia.org/wiki/Spline_(mathematics).

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

[11] "Conjugate gradient wikipedia," https://en.wikipedia.org/wiki/Conjugate_gradient_method.

[12] Y. Qiao, B. van Lew, B. P. F. Lelieveldt, and M. Staring, "Fast automatic step size estimation for gradient descent optimization of image registration," *IEEE Transactions on Medical Imaging*, vol. 35, no. 2, pp. 391–403, Feb 2016.

[13] L. Hasan, M. Kentie, and Z. Al-Ars, "Dopa: Gpu-based protein alignment using database and memory access optimizations," *BMC Research Notes*, vol. 4, no. 1, p. 261, Jul 2011. [Online]. Available: http://dx.doi.org/10.1186/1756-0500-4-261

[14] "Nvidia online blog on fast histogram computation," https://devblogs.nvidia.com/parallelforall/gpu-pro-tip-fast-histograms-using-shared-atomics-maxwell/#more-4175.

[15] S. T. Ratliff, "Webb's physics of medical imaging, second edition." *Medical Physics*, vol. 40, no. 9, pp. 097 301–n/a, 2013, 097301. [Online]. Available: http://dx.doi.org/10.1118/1.4818282

[16] G. O'Regan, *Pillars of Computing: A Compendium of Select, Pivotal Technology Firms*. Springer International Publishing, 2015. [Online]. Available: https://books.google.nl/books?id=RBKcCgAAQBAJ

[17] H. A. D. Nguyen, Z. Al-Ars, G. Smaragdos, and C. Strydis, "Accelerating complex brain-model simulations on GPU platforms," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 974–979.

[18] G. Smaragdos, G. Chatzikonstantis, R. Kukreja, H. Sidiropoulos, D. Rodopoulos, I. Sourdis, Z. Al-Ars, C. Kachris, D. Soudris, C. D. Zeeuw, and C. Strydis, "Brainframe: A node-level heterogeneous accelerator platform for neuron simulations," *Journal of Neural Engineering*, vol. 14, November 2017.

[19] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars, "GPU-accelerated bwa-mem genomic mapping algorithm using adaptive load balancing," in *Proceedings of the 29th International Conference on Architecture of Computing Systems – ARCS 2016 - Volume 9637*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 130–142. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-30695-7_10

[20] S. Ren, K. Bertel, and Z. Al-Ars, "Exploration of alternative GPU implementations of the pair-hmms forward algorithm," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Dec 2016, pp. 902–909.

[21] P. Bhosale, M. Staring, Z. Al-Ars, and F. F. Berendsen, "GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications," *SPIE on Medical Imaging*, to appear in Feb 2018.

[22] S. Klein, J. P. Pluim, M. Staring, and M. A. Viergever, "Adaptive stochastic gradient descent optimisation for image registration," *International journal of computer vision*, vol. 81, no. 3, p. 227, 2009.

[23] 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.

[24] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," vol. 6, pp. 163 – 168, 08 1963.

[25] "Lumc image registration," https://www.lumc.nl/org/radiologie/research/LKEB/969723/1301100528142222/.

[26] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[27] M. Modat, G. R. Ridgway, Z. A. Taylor, M. Lehmann, J. Barnes, D. J. Hawkes, N. C. Fox, and S. Ourselin, "Fast free-form deformation using graphics processing units," *Comput. Methods Prog. Biomed.*, vol. 98, no. 3, pp. 278–284, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.cmpb.2009.09.002

[28] B. Haghighi, N. D. Ellingwood, Y. Yin, E. A. Hoffman, and C.-L. Lin, "A gpu-based symmetric non-rigid image registration method in human lung," *Medical & Biological Engineering & Computing*, Aug 2017. [Online]. Available: https://doi.org/10.1007/s11517-017-1690-2

[29] G. C. Sharp, N. Kandasamy, H. Singh, and M. Folkert, "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," *Phys Med Biol*, vol. 52, no. 19, pp. 5771–5783, Oct 2007.

[30] D. Shamonin, E. Bron, B. Lelieveldt, M. Smits, S. Klein, and M. Staring, "Fast parallel image registration on cpu and gpu for diagnostic classification of alzheimer's disease," *Frontiers in Neuroinformatics*, vol. 7, p. 50, 2014. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fninf.2013.00050

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

[32] D. Zastrau and S. Edelkamp, *Stochastic Gradient Descent with GPGPU*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 193–204. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33347-7_17

[33] F. F. Berendsen, A. N. Kotte, A. A. de Leeuw, M. A. Viergever, and J. P. Pluim, "Free-form registration involving disappearing structures: application to brachytherapy mri," in *International MICCAI Workshop on Computational and Clinical Challenges in Abdominal Imaging*. Springer, Berlin, Heidelberg, 2013, pp. 136–144.

[34] F. F. Berendsen, K. Marstal, S. Klein, and M. Staring, "The design of superelastix 2014; a unifying framework for a wide range of image registration methodologies," in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2016, pp. 498–506.

[35] "Insight segmentation and registration toolkit," http://www.itk.org, accessed: 2010-09-30.

[36] J. Stolk, H. Putter, E. M. Bakker, S. B. Shaker, D. G. Parr, E. Piitulainen, E. W. Russi, E. Grebski, A. Dirksen, R. A. Stockley, J. H. Reiber, and B. C. Stoel, "Progression parameters for emphysema: A clinical investigation," *Respiratory Medicine*, vol. 101, no. 9, pp. 1924 – 1930, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0954611107001862

[37] R. Shams and R. Kennedy, "Efficient histogram algorithms for nvidia cuda compatible devices," in *Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS)*, 2007, pp. 418–422.

[38] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[39] "Kepler tunning guide nvidia," http://docs.nvidia.com/cuda/kepler-tuning-guide/#axzz4lJ5wP6Tw, accessed: 2010-09-30.

[40] "Nibabel python package," http://nipy.org/nibabel/.

[41] "Mevislab-a powerful modular framework for image processing research and development," https://www.mevislab.de/.

[42] A. Hammers, R. Allom, M. Koepp, S. L Free, R. Myers, L. Lemieux, T. N Mitchell, D. Brooks, and J. S Duncan, "Three-dimensional maximum probability atlas of the human brain, with particular reference to the temporal lobe," vol. 19, pp. 224–47, 08 2003.

A

A.1 Intel E5-1620 CPU specifications

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz
Stepping:              1
CPU MHz:               1199.980
CPU max MHz:           3800.0000
CPU min MHz:           1200.0000
BogoMIPS:              6984.19
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              10240K
NUMA node0 CPU(s):     0-7
```

A.2 GPU specifications

```
Device 0: "Tesla K40c"
  CUDA Driver Version / Runtime Version          8.0 / 8.0
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 11440 MBytes
  (15) Multiprocessors, (192) CUDA Cores/MP:     2880 CUDA Cores
  GPU Max Clock rate:                            745 MHz (0.75 GHz)
```

```
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536),2D=(65536, 65536),
                                                 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384),2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 2 / 0
```

A.3 NiftyRegSGD command line arguments

```
reg_f3d -ref /srv/2-lkeb-16-reg1/pbhosale/SPREAD/nifty/p000_Aard/baseline_1_crop.nii
-flo /srv/2-lkeb-16-reg1/pbhosale/SPREAD/nifty/p000_Aard/followup_1_crop.nii
-res /srv/2-lkeb-16-reg1/pbhosale/SPREAD/Results_lap/p000_Aard_out.nii
-aff /srv/2-lkeb-16-reg1/pbhosale/SPREAD/param/affine_matrix_p000_Aard_1.txt
-cpp /srv/2-lkeb-16-reg1/pbhosale/SPREAD/Results_lap/p000_Aard_cpp.nii
-rdmsam 1 -maxit 10
-SGD 0.25 20 0.9
-noConj
-ln 3
-gpu
```

Above snippet is an example of NiftyRegSGD command. To profile with Nvidia profiler, nvpp is added at the start. For Vtune profiler, binary reg_f3d is selected (compiled with -g flag) and command line arguments are provided. The following table explains each

command line inputs.

-ref	fixed or reference file location
-flo	floting or moving image
-res	ouput aligned image
-aff	input affine parameters
-res	output aligned image
-cpp	output control point image
-rdmsam	sampling percentage
-SGD	$\langle a \rangle \langle A \rangle \langle \alpha \rangle$from equation 2.9
-noConj	not to use conjugate gradient
-ln	number of resolutions
-gpu	use GPGPU

A.4 reg_f3d log file

Following snippet shows a log file from reg_f3d. At the start of the log file, command line and, details about input images, and control point grid are shown. Then, at each level respective resolution images are shown starting with lowest resolution. Cost function value is defined by **Current objective function** at each iteration after adjusting with bending energy penalty. This value is used to plot cost plots in the result chapter. The decreasing step size is obtained by the value followed by $\lfloor +$ after cost function. The time taken by B-spline and resampler are also stated, which are used to calculate for throughput discussion. The final timing is logged by **Transformation performed time** at the end of the file. All the plots are generated using python and Matlab, and the scripts can be found here.

```
[NiftyReg F3D] Command line:
 /home/pbhosale/tools/git/implement/build/install/bin/reg_f3d -ref
/srv/2-lkeb-16-reg1/pbhosale/SPREAD/nifty/p000_Aard/baseline_1_crop.nii
-flo /srv/2-lkeb-16-reg1/pbhosale/SPREAD/nifty/p000_Aard/followup_1_crop.nii
-res /srv/2-lkeb-16-reg1/pbhosale/SPREAD/Results_lap/p000_Aard_out.nii
-aff /srv/2-lkeb-16-reg1/pbhosale/SPREAD/param/affine_matrix_p000_Aard_1.txt
-cpp /srv/2-lkeb-16-reg1/pbhosale/SPREAD/Results_lap/p000_Aard_cpp.nii
-rdmsam 1 -maxit 10 -SGD 0.25 20 0.9 -noConj -ln 3 -gpu


[NiftyReg F3D] GPU implementation is used
[NiftyReg F3D] OpenMP is used with 8 thread(s)
[NiftyReg F3D] ***************************************************
[NiftyReg F3D] INPUT PARAMETERS
[NiftyReg F3D] ***************************************************
[NiftyReg F3D] Reference image:
[NiftyReg F3D]  * name: /srv/2-lkeb-16-reg1/pbhosale/SPREAD/nifty/p000_Aard/
baseline_1_crop.nii
[NiftyReg F3D]  * image dimension: 446 x 315 x 129 x 1
```

```
[NiftyReg F3D]  * image spacing: 0.683 x 0.683 x 2.5 mm
[NiftyReg F3D]  * intensity threshold for timepoint 1/1: [-3.4e+38
3.4e+38]
[NiftyReg F3D]  * binnining size for timepoint 1/1: 64
[NiftyReg F3D]  * gaussian smoothing sigma: 0
[NiftyReg F3D]
[NiftyReg F3D] Floating image:
[NiftyReg F3D]  * name: /srv/2-lkeb-16-reg1/pbhosale/SPREAD/nifty/p000_Aard/
followup_1_crop.nii
[NiftyReg F3D]  * image dimension: 458 x 332 x 131 x 1
[NiftyReg F3D]  * image spacing: 0.683 x 0.683 x 2.5 mm
[NiftyReg F3D]  * intensity threshold for timepoint 1/1: [-3.4e+38
3.4e+38]
[NiftyReg F3D]  * binnining size for timepoint 1/1: 64
[NiftyReg F3D]  * gaussian smoothing sigma: 0
[NiftyReg F3D]
[NiftyReg F3D] Warped image padding value: nan
[NiftyReg F3D]
[NiftyReg F3D] Level number: 3
[NiftyReg F3D]
[NiftyReg F3D] Maximum iteration number per level: 10
[NiftyReg F3D]
[NiftyReg F3D] Final spacing in mm: -5 -5 -5
[NiftyReg F3D]
[NiftyReg F3D] The NMI is used as a similarity measure.
[NiftyReg F3D] The Parzen window joint histogram filling is approximated
[NiftyReg F3D] Similarity measure term weight: 0.995
[NiftyReg F3D]
[NiftyReg F3D] Bending energy penalty term weight: 0.005
[NiftyReg F3D]
[NiftyReg F3D] Linear energy penalty term weights: 0 0
[NiftyReg F3D]
[NiftyReg F3D] L2 norm of the displacement penalty term weights: 0
[NiftyReg F3D]
[NiftyReg F3D] Jacobian-based penalty term weight: 0
[NiftyReg F3D] ************************************************
[NiftyReg F3D] Current level: 1 / 3
[NiftyReg F3D] Current reference image
[NiftyReg F3D]  * image dimension: 111 x 78 x 32 x 1
[NiftyReg F3D]  * image spacing: 2.732 x 2.732 x 10 mm
[NiftyReg F3D] Current floating image
[NiftyReg F3D]  * image dimension: 114 x 83 x 32 x 1
[NiftyReg F3D]  * image spacing: 2.732 x 2.732 x 10 mm
[NiftyReg F3D] Current control point image
[NiftyReg F3D]  * image dimension: 27 x 20 x 11
```

```
[NiftyReg F3D]  * image spacing: 13.66 x 13.66 x 50 mm
[NiftyReg F3D] Total samples=277056 Percents=1.000000 Number of random
samples=2770
[NiftyReg F3D] MaxStepsize=10.000000
[NiftyReg F3D] reg_bspline_getDeformationField time =0.115000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.312000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.080000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.296000 msec
[NiftyReg F3D] [2] Current objective function: -1.96343 = (wNMI)1.08816
- (wBE)3.05e+00 [+ 0.161415 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.081000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.295000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.084000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.297000 msec
[NiftyReg F3D] [4] Current objective function: -1.76927 = (wNMI)1.11638
- (wBE)2.89e+00 [+ 0.148725 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.085000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.312000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.086000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.298000 msec
[NiftyReg F3D] [6] Current objective function: -1.44237 = (wNMI)1.10499
- (wBE)2.55e+00 [+ 0.137973 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.084000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.297000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.085000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.298000 msec
[NiftyReg F3D] [8] Current objective function: -1.52079 = (wNMI)1.12316
- (wBE)2.64e+00 [+ 0.12874 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.084000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.297000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.086000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.297000 msec
[NiftyReg F3D] [10] Current objective function: -3.20975 = (wNMI)1.11812
- (wBE)4.33e+00 [+ 0.120721 mm]
[NiftyReg F3D] Current registration level done
[NiftyReg F3D] ------------------------------------------------
[NiftyReg F3D] ************************************************
[NiftyReg F3D] Current level: 2 / 3
[NiftyReg F3D] Current reference image
[NiftyReg F3D]  * image dimension: 223 x 157 x 64 x 1
[NiftyReg F3D]  * image spacing: 1.366 x 1.366 x 5 mm
[NiftyReg F3D] Current floating image
[NiftyReg F3D]  * image dimension: 229 x 166 x 65 x 1
[NiftyReg F3D]  * image spacing: 1.366 x 1.366 x 5 mm
[NiftyReg F3D] Current control point image
```

```
[NiftyReg F3D]  * image dimension: 49 x 36 x 17
[NiftyReg F3D]  * image spacing: 6.83 x 6.83 x 25 mm
[NiftyReg F3D] Total samples=2240704 Percents=1.000000 Number of random
samples=22407
[NiftyReg F3D] MaxStepsize=5.000000
[NiftyReg F3D] reg_bspline_getDeformationField time =0.125000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.332000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.106000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.305000 msec
[NiftyReg F3D] [2] Current objective function: 0.82634 = (wNMI)1.1099 -
(wBE)2.84e-01 [+ 0.0807073 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.104000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.304000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.106000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.304000 msec
[NiftyReg F3D] [4] Current objective function: 0.871169 = (wNMI)1.12887
- (wBE)2.58e-01 [+ 0.0743627 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.104000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.311000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.105000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.312000 msec
[NiftyReg F3D] [6] Current objective function: 0.8868 = (wNMI)1.12412 -
(wBE)2.37e-01 [+ 0.0689865 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.104000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.309000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.106000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.308000 msec
[NiftyReg F3D] [8] Current objective function: 0.902789 = (wNMI)1.12849
- (wBE)2.26e-01 [+ 0.0643699 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.104000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.308000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.107000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.318000 msec
[NiftyReg F3D] [10] Current objective function: 0.908478 = (wNMI)1.12253
- (wBE)2.14e-01 [+ 0.0603604 mm]
[NiftyReg F3D] Current registration level done
[NiftyReg F3D] ---------------------------------------------
[NiftyReg F3D] *********************************************
[NiftyReg F3D] Current level: 3 / 3
[NiftyReg F3D] Current reference image
[NiftyReg F3D]  * image dimension: 446 x 315 x 129 x 1
[NiftyReg F3D]  * image spacing: 0.683 x 0.683 x 2.5 mm
[NiftyReg F3D] Current floating image
[NiftyReg F3D]  * image dimension: 458 x 332 x 131 x 1
[NiftyReg F3D]  * image spacing: 0.683 x 0.683 x 2.5 mm
```

```
[NiftyReg F3D] Current control point image
[NiftyReg F3D]  * image dimension: 94 x 67 x 30
[NiftyReg F3D]  * image spacing: 3.415 x 3.415 x 12.5 mm
[NiftyReg F3D] Total samples=18123210 Percents=1.000000 Number of random
samples=181232
[NiftyReg F3D] MaxStepsize=2.500000
[NiftyReg F3D] reg_bspline_getDeformationField time =0.373000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.398000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.351000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.387000 msec
[NiftyReg F3D] [2] Current objective function: 1.10396 = (wNMI)1.11966 -
(wBE)1.57e-02 [+ 0.0403536 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.351000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.387000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.351000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.387000 msec
[NiftyReg F3D] [4] Current objective function: 1.10694 = (wNMI)1.12339 -
(wBE)1.65e-02 [+ 0.0371813 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.351000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.390000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.351000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.389000 msec
[NiftyReg F3D] [6] Current objective function: 1.10705 = (wNMI)1.12388 -
(wBE)1.68e-02 [+ 0.0344932 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.355000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.388000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.346000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.365000 msec
[NiftyReg F3D] [8] Current objective function: 1.11203 = (wNMI)1.12919 -
(wBE)1.72e-02 [+ 0.0321849 mm]
[NiftyReg F3D] reg_bspline_getDeformationField time =0.346000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.364000 msec
[NiftyReg F3D] reg_bspline_getDeformationField time =0.346000 msec
[NiftyReg F3D] reg_resampleSourceImage_kernel time =0.365000 msec
[NiftyReg F3D] [10] Current objective function: 1.1146 = (wNMI)1.13197 -
(wBE)1.74e-02 [+ 0.0301802 mm]
[NiftyReg F3D] Current registration level done
[NiftyReg F3D] ------------------------------------------------
[NiftyReg F3D] Transformation performed time is 2.039961 sec
[NiftyReg F3D] Transformation performed in 0 min 2 sec
[NiftyReg F3D] Registration Performed in 0 min 3 sec
[NiftyReg F3D] Have a good day !
```

A.5 GitHub link

The work done during the thesis is publicly available on github. To download NiftyRegSGD, please download the branch random_sampling.

```
git clone -b random_sampling https://github.com/SuperElastix/NiftyRegSGD.git
```

A.6 SPIE paper abstract

An abstract of the paper has been accepted by the medical imaging conference by the SPIE which will be held in February 2018 at Houston, Texas. The main paper is under development right now.

GPU-based stochastic-gradient optimization for non-rigid medical image registration in time-critical applications

Parag Bhosale[a,b], Marius Staring[b,c], Zaid Al-Ars[a], and Floris F. Berendsen[b]

[a]Computer Engineering, Delft University of Technology, Delft, The Netherlands
[b]Division of Image Processing, Leiden University Medical Center, Leiden, The Netherlands
[c]Department of Intelligent Systems, Delft University of Technology, Delft, The Netherlands

ABSTRACT

Currently, non-rigid image registration algorithms are too computationally intensive to use in time-critical applications. Existing implementations that focus on speed typically address this by either parallelization on GPU-hardware, or by introducing methodically novel techniques into CPU-oriented algorithms. Stochastic gradient descent (SGD) optimization and variations thereof have proven to drastically reduce the computational burden for CPU-based image registration, but have not been successfully applied in GPU hardware due to its stochastic nature. This paper proposes 1) NiftyRegSGD, a SGD optimization for the GPU-based image registration tool NiftyReg, 2) random chunk sampler, a new random sampling strategy that better utilizes the memory bandwidth of GPU hardware. Experiments have been performed on 3D lung CT data of 19 patients, which compared NiftyRegSGD (with and without random chunk sampler) with CPU-based elastix Fast Adaptive SGD (FASGD) and NiftyReg. The registration runtime was 21.5s, 4.4s and 2.8s for elastix-FASGD, NiftyRegSGD without, and NiftyRegSGD with random chunk sampling, respectively, while similar accuracy was obtained. Our method is publicly available at `https://github.com/SuperElastix/NiftyRegSGD`.

Keywords: Non-rigid image registration, stochastic gradient descent, GPGPU, memory access optimization, random chunk sampling

1. DESCRIPTION OF PURPOSE

Image registration is widely used in clinical applications. The high number of parameters in non-rigid registration (up to 100k in some cases) makes this approach computationally intensive, resulting in a rather high computation time. This limits non-rigid registration from being used in real-time critical applications, such as image-guided surgery or online adaptive radiotherapy.

To address this problem, FASGD[1] was proposed recently, which speeds up the adaptive[2] version of stochastic gradient descent optimization (SGD).[3] In SGD, the cost function gradients are computed using a random subset of samples (i.e. voxels) instead of the entire set of sample space (i.e. full image). A random subset is generated each iteration. Hence, this approach reduces data intensity which results in less computation time as compared to conventional gradient descent methods. High performance computation approaches such as NiftyReg[4] and Plastimatch[5] take advantages of the GPU architecture for parallel computing. These implementations are, however, based on gradient descent optimization using full image sampling. In Shamonin et al.,[6] CPU and GPU parallelization were implemented in `elastix`.[7] In the field of machine learning or neural networks, there exist implementations of SGD on the GPU.[8] However, these methods typically optimize over a large collection of data and the term stochastic refers to random batches of data instead of random voxels within one image, which is a fundamental difference from a registration point of view.

This paper presents the implementation of SGD on the GPU architecture for image registration to take advantage of both stochastic optimization as well as parallel computing. For the proposed work, coined NiftyRegSGD, NiftyReg was chosen as a foundation of our implementation for its public access and pre-existing use of the GPU. The accuracies and timings of lung CT registrations are compared with `elastix`-FASGD, which has been evaluated using the same data and is the fastest publicly available method to our knowledge.

Further author information: (Send correspondence to Parag Bhosale)
Parag Bhosale: E-mail: P.S.Bhosale@student.tudelft.nl, Telephone: +31 65 15 11725
Floris Berendsen: E-mail: F.Berendsen@lumc.nl, Telephone: +31 71 52 66206

2. METHOD

Our implementation NiftyRegSGD is based on NiftyReg. In this section we discuss the several changes that have been carried out to implement a stochastic gradient descent method and we introduce the random chunk sampler. Image registration may be formulated as an iterative optimization problem, using:

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k - \gamma_k \mathbf{g}_k, \tag{1}$$

where $\boldsymbol{\mu}$ represents the transformation parameters at iteration k, $\mathbf{g}$ is the search direction and γ_k the stepsize.

The optimizer of NiftyReg uses a search direction $\mathbf{g}_k$ based on the gradient in combination with a line search strategy to select the optimal stepsize γ_k. The line search of NiftyReg inherently imposes a stopping criterion for k. For the search direction either the gradient $\mathbf{g}_k := \partial C/\partial \boldsymbol{\mu}$, or the conjugate gradient of the cost function C can be selected. The cost function in image registration typically takes the following form:

$$C(\mu) = \Psi\left(\frac{1}{|\Omega_F|} \sum_{x_i \in \Omega_F} \xi(F(x_i), M(\mathbf{T}(x_i, \mu)))\right), \tag{2}$$

where $\Psi(u)$ and $\xi(u, v)$ are continuous and differentiable functions, and where Ω_F is a discrete set of voxel coordinates from the fixed image. The key idea in SGD[3] is to compute a fast but noisy approximation of the search direction $\tilde{\mathbf{g}}_k := \partial \widetilde{C}/\partial \boldsymbol{\mu}$, by randomly selecting a small subset of all fixed image coordinates. In each iteration a new random subset is drawn. An exponentially decaying stepsize γ_k is typically used:

$$\gamma_k = \frac{\delta a}{(A + k)^\alpha}, \tag{3}$$

where δ is the maximum voxel spacing among the x, y and z axes. Parameters a, A and α are constants that typically need to be tuned for the type of data. Since stochastic gradient descent does not have a trivial stopping criterion the maximum number of iterations $k_{\max}$ needs to be set as well.

The architectural changes applied to NiftyReg allow for the implementation of a random sampler, enabling the calculation of the approximate gradient and replacing the line search strategy with a stopping criterion by the decaying step size function using a fixed number of iterations. We implemented a naive sampler that randomly picks samples from the fixed image. This randomness, however, prevents the GPU from accessing memory in parallel, forcing sequential global memory reads, which lead to higher memory access time and wastage of memory bandwidth. Therefore, we propose a new sampling strategy that is better tailored to GPU hardware, coined random chunk sampling. In this strategy, every first sample out of 32 samples is created randomly, followed by 31 samples adjacent to this first sample. This enables 32 threads on a GPU to have a coalesced memory access, which results in faster memory access and increased GPU throughput.

3. EXPERIMENTS AND RESULTS

The proposed NiftyRegSGD method with naive random sampling as well as with random chunk sampling is compared with `elastix`-FASGD and the original NiftyReg.

For the experiments, CT lung data from the SPREAD study[9] have been used. It consists of 19 patients with baseline (fixed) and follow-up (moving) images. Each patient has 100 corresponding landmarks that serve as a ground truth to evaluate the target registration error (TRE). The data was divided into a training set of 10 patients, to find the optimal registration settings, and a testing set of 9 patients to perform the final evaluation. All experiments were run on an Intel Xeon E5-1620 CPU with a Tesla K40c GPU.

The registration settings we used for `elastix`-FASGD where proposed by its authors[1] and were determined for the same SPREAD data set. To have a fair comparison based on speed we tuned NiftyRegSGD with the random chunk sampler to match the median TRE accuracy to that of `elastix`-FASGD. NiftyReg could not be tuned to achieve a sufficient median TRE, therefore we report using the default settings. For NiftyRegSGD we used the same B-spline grid spacing as NiftyReg. The values $\alpha = 0.90$ and $A = 20$ were chosen from the literature.[2] We optimized the parameters gain factor $a \in [0.05, 0.65]$, sampling percentage $s_{\%} \in [10, 80]$ and

maximum number of iteration $k_{\max} \in [10, 300]$, over the training set. Keeping computation time in mind, the optimal value of a was found to be 0.25. For $a = 0.25$, optimal values for $s_{\%} = 15\%$ and $k_{\max} = 20$ were found, see Figure 1. All settings are summarized in Table 1.

Profiling the most time consuming GPU kernels, i.e. the B-spline transform and the resampler, shows a ten and six fold improvement of the throughput from the naive random sampler to the random chunk sampler for the B-spline and resampler kernel respectively.

	elastix-FASGD	**Original NiftyReg**	**NiftyRegSGD (both)**
Resolutions	3	3	3
Transform	B-spline	B-spline	B-spline
3D Parameters	$\approx 1k/6k/35k$	$\approx 6k/35k/230k$	$\approx 6k/35k/230k$
Metric	NMI	NMI	NMI
Optimizer	FASGD	Conjugate gradient	SGD
Step size	adaptive	line search	Equation (3)
Iterations	500/500/500	$\approx 54/64/169$	20/20/20
Sampler	random	full	random or random chunk
Samples	5000/5000/5000 (0.04%)	$\approx 4 \cdot 10^5/3 \cdot 10^6/2 \cdot 10^7$ (100%)	$\approx 6 \cdot 10^4/5 \cdot 10^5/4 \cdot 10^6$ (15%)

Table 1: Algorithmic and settings overview of the various registration methods.

The accuracies and average timings of the four methods are compared in Figure 2 for both the training and the test set. For the training data, Wilcoxon signed rank tests indicated that there were no significant differences in median accuracies of both NiftyRegSGD methods with respect to `elastix`-FASGD ($p > 0.05$). For the test data, the differences with respect to `elastix`-FASGD were found to be just significant for the NiftyRegSGD with random chunk sampling and not for the NiftyRegSGD with naive random sampling.

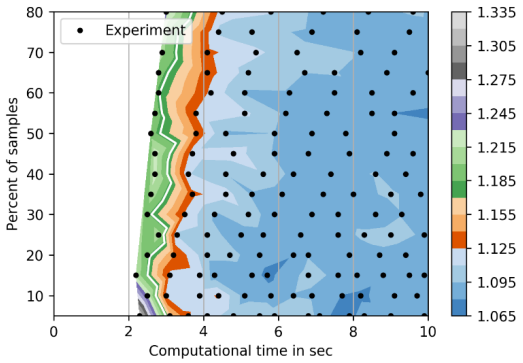


Figure 1: The median TRE for NiftyRegSGD with random chunk sampling is plotted against $s_{\%}$ in steps of 5% and $k_{\max}$ in steps of 10 for $a = 0.25$ for the training set. The background color indicates the median TRE. The white curve equals the median TRE of `elastix`-FASGD. The left-most point on this white curve indicates the fastest setting equally accurate as FASGD. The optimal settings are $s_{\%} = 15\%$ and $k_{\max} = 20$.

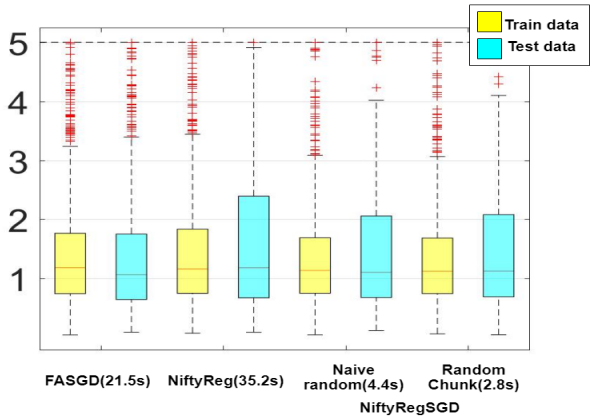


Figure 2: Target registration error (TRE) in mm and average runtime (indicated in the labels) using settings of Table 1 for the training and test data set. Proposed NiftyRegSGD with random chunk sampling performs fastest with better accuracy for the training data and less accuracy for the test data compared to `elastix`-FASGD.

4. DISCUSSION AND CONCLUSION

We presented our image registration method NiftyRegSGD which introduces stochastic gradient descent (SGD) optimization in a high performance GPU implementation. Experiments have been performed on follow-up lung

CT scans. The use of SGD drastically speeds up registration time (to 2.8s) compared to the current GPU-based registration method (NiftyReg, 35.2s) that uses fully deterministic gradients. At an equal median target registration error, our GPU implementation also outperforms elastix-FASGD (21.5s), to our knowledge the currently fastest method on this dataset.

We introduced a random chunk sampler as part of our SGD, which shows a speed improvement over a naive random sampler (4.4s). Due to its coalesce memory access, the throughput of GPU kernels is drastically increased. Sacrificing some of the randomness-properties only harmed the convergence rate a little (as observed in the test set), which might easily be compensated for by an extra iteration.

Experiments from Figure 1 have shown that the stochastic sub-sampling percentage of NiftyRegSGD was found to be optimal at 15%, in contrast to the CPU-based SGD of elastix-FASGD of around 0.04%. A smaller percentage of data reduces the computation time per iteration, but typically degrades the convergence rate. On the GPU, this trade-off favors a higher sampling percentage due to its parallel computation.

The small difference in accuracy for training and test data shows that accuracy depends on manual tuning, that is currently needed for NiftyRegSGD. For future work we may adopt the automatic parameter estimation methods from FASGD to remedy this. We may accelerate this estimation procedure on the GPU as well. The current results however still yield sub-voxel accuracy within 2.8s for a non-rigid registration procedure. We therefore conclude that the proposed methods open up possibilities to embed online registration in the clinical workflow, and consequently may benefit e.g. image-guided surgery and adaptive radiotherapy. Our fork of NiftyReg is publicly available at https://github.com/SuperElastix/NiftyRegSGD.

Acknowledgements

The Tesla K40c used for this research was donated by the NVIDIA Corporation.

REFERENCES

[1] Qiao, Y., van Lew, B., Lelieveldt, B. P. F., and Staring, M., "Fast automatic step size estimation for gradient descent optimization of image registration," *IEEE Transactions on Medical Imaging* **35**, 391–403 (Feb 2016).

[2] Klein, S., Pluim, J. P., Staring, M., and Viergever, M. A., "Adaptive stochastic gradient descent optimisation for image registration," *International journal of computer vision* **81**(3), 227 (2009).

[3] Robbins, H. and Monro, S., "A stochastic approximation method," *The annals of mathematical statistics* , 400–407 (1951).

[4] Modat, M., Ridgway, G. R., Taylor, Z. A., Lehmann, M., Barnes, J., Hawkes, D. J., Fox, N. C., and Ourselin, S., "Fast free-form deformation using graphics processing units," *Comput. Methods Prog. Biomed.* **98**, 278–284 (June 2010).

[5] Sharp, G. C., Kandasamy, N., Singh, H., and Folkert, M., "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," *Phys Med Biol* **52**, 5771–5783 (Oct 2007).

[6] Shamonin, D., Bron, E., Lelieveldt, B., Smits, M., Klein, S., and Staring, M., "Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease," *Frontiers in Neuroinformatics* **7**, 50 (2014).

[7] Klein, S., Staring, M., Murphy, K., Viergever, M. A., and Pluim, J. P., "Elastix: a toolbox for intensity-based medical image registration," *IEEE Transactions on Medical Imaging* **29**(1), 196–205 (2010).

[8] Zastrau, D. and Edelkamp, S., [*Stochastic Gradient Descent with GPGPU*], 193–204, Springer Berlin Heidelberg, Berlin, Heidelberg (2012).

[9] Stolk, J., Putter, H., Bakker, E. M., Shaker, S. B., Parr, D. G., Piitulainen, E., Russi, E. W., Grebski, E., Dirksen, A., Stockley, R. A., Reiber, J. H., and Stoel, B. C., "Progression parameters for emphysema: A clinical investigation," *Respiratory Medicine* **101**(9), 1924 – 1930 (2007).