ABSTRACT

Title of Document: PARALLEL COMPUTATION OF NONRIGID IMAGE REGISTRATION

Frances Kimpik Leung, Master of Science, 2010

Directed By: Professor Raj Shekhar, Dept of Diagnostic Radiology (University of Maryland, Baltimore) and Dept of Electrical and Computer Engineering

Automatic intensity-based nonrigid image registration brings significant impact in medical applications such as multimodality fusion of images, serial comparison for monitoring disease progression or regression, and minimally invasive image-guided interventions. However, due to memory and compute intensive nature of the operations, intensity-based image registration has remained too slow to be practical for clinical adoption, with its use limited primarily to as a pre-operative too. Efficient registration methods can lead to new possibilities for development of improved and interactive intraoperative tools and capabilities.

In this thesis, we propose an efficient parallel implementation for intensity-based three-dimensional nonrigid image registration on a commodity graphics processing unit. Optimization techniques are developed to accelerate the compute-intensive mutual information computation. The study is performed on the hierarchical volume subdivision-based algorithm, which is inherently faster than other nonrigid registration algorithms and structurally well-suited for data-parallel

computation platforms. The proposed implementation achieves more than 50-fold runtime improvement over a standard implementation on a CPU. The execution time of nonrigid image registration is reduced from hours to minutes while retaining the same level of registration accuracy.

PARALLEL COMPUTATION OF NON-RIGID IMAGE REGISTRATION

By

Frances Kimpik Leung

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2010

Advisory Committee:
Professor Raj Shekhar, Chair
Professor Shuvra Bhattacharyya
Professor Uzi Vishkin

Acknowledgements

I would like to express my sincere utmost gratitude to Dr. Raj Shekhar for his guidance throughout this thesis. Without his supervision and direction, this thesis would not have been possible. I would also like to thank my thesis committee members, Dr. Shuvra Bhattacharryya and Dr. Uzi Vishkin, for their cooperation and support amidst their busy schedules.

I am also thankful to my colleagues at the Imaging Technologies Laboratory, in particular, Dr. William Plishker and Dr. Reza Rad. They have been providing valuable help and inputs at various times during my research. Many insightful discussions and feedback have sharpened this work on to completion.

Finally, I would like to thank my grandparents and parents for their support and encouragement. They have always been patient to me and challenged me in achieving higher goals in my career. I am also grateful to my siblings, Helen, Kimmy, Cece, and Kim, for their words of encouragement. I would like to especially mention my long time friends Jessica and Chanson for always been there with me. Many people have been part of my graduate education and I take this opportunity to thank them all.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1. Introduction and Motivation

Image registration, the process of spatially aligning two images, is an essential need in a number of medical procedures. Medical image registration has historically been used for multimodality fusion of images providing complementary information; comparison of images from different time points for qualifying disease progression or regression; and recently emerging to applications in fusion of pre- and intraoperative images in minimally invasive image-guided interventions (IGIs). These applications have the potential to improve the quality of patient care by improving the efficiency and effectiveness of the associated medical procedures.

The success of these novel medical capabilities is critically dependent on accurate and precise target identification and localization. Previous research has recognized intensity-based registration by maximizing mutual information (MI) between two images as the most accurate, robust, versatile and fully automatic approach to image registration. Meyer et al. [1] demonstrated the accuracy and clinical versatility of MI for automatic multimodality thoracic and abdominal image registration. Rueckert et al. [2] developed an MI-based nonrigid registration algorithm for deformation correction in three-dimensional (3D) magnetic resonance (MR) breast images. Hill et al. [3] applied nonrigid registration to pre- and post-resection interventional MR brain images to quantify intraoperative brain deformation.

Whereas these proven advantages have led to intensity-based registration becoming the approach of choice, the lengthy execution of MI computation continues to discourage clinical adoption. Fast automatic image registration will open new possibilities for development of interactive intraoperative tools.

1.2. Contribution of this Thesis

The goal of this thesis work is to present a multiprocessor implementation for nonrigid image registration applications by utilizing parallelism in MI computation. The hierarchical volume subdivision-based algorithm reported by Walimbe and Shekhar [4] is the focus in this work. The parallelizable nature of this registration algorithm would be exploited.

In this thesis, a complete implementation of the graphics processing unit (GPU)-based nonrigid image registration algorithm is presented. The implementation is scalable for the ever evolving generations of GPU or other massively parallel architectures in the future. A warp-aware sort and merge technique is presented to target the dominated runtime bottleneck in compiling the mutual histograms for MI computation.

The final implementation is validated with five sets of 256 x 256 x 256 computed tomography (CT)-CT test cases. A GPU with the latest NVIDIA hardware and software architecture is used to benchmark the resulting performance and accuracy. The overall performance of rigid and nonrigid registrations implementations is compared with a CPU and an FPGA-based implementation.

1.3. Outline of this Thesis

The thesis is organized as follows: Chapter 2 provides the background on image registration and the associated techniques used within the registration framework. The concept of intensity-based image registration, specifically MI-based algorithm, are discussed. The latest GPU architecture and recently reported GPU-based image registration results follow next. In Chapter 3, a GPU-based rigid registration implementation is presented. Numbers of critical design considerations will be discussed in the content of rigid registration and further expanded while discussing nonrigid registration. In Chapter 4, a GPU-based nonrigid registration implementation is presented for the hierarchical volume subdivision-based image registration algorithm. Chapter 5 describes an optimized multiprocessor implementation in detail. Finally, in Chapter 6, conclusions and future work are presented.

Chapter 2: Background and Related Work

Three-dimensional (3D) image registration is fundamental to various medical procedures, including image-based longitudinal comparison, multimodality image fusion, and population based atlas creation. It is a prerequisite when *before* and *after* images are to be compared (subtracted) in longitudinal comparison studies for quantifying disease progression or regression, often in response to a treatment. It is also a necessary first step in multimodality image fusion before images from one or more modalities with complementary information can be meaningfully overlaid. In these intra-patient instances, images are misregistered (misaligned) because they are usually acquired at two different times separated by hours to months with the patient in different body orientations. In the inter-patient instance of population-based atlas creation, images are misaligned also because they come from different individuals with different body types and thus necessitate image registration.

One of the growing applications of image registration that requires real-time performance is image-guided interventions (IGIs). The success of IGIs is critically dependent on accurate and precise target identification. Diagnostic-quality pre-treatment images are often used for treatment and navigation planning, while intra-treatment images available to provide accurate spatial information to navigate interventional devices are generally lower-resolution and less information-rich. Rapidly merging these two types of images with complementary strengths (clear target definition and up-to-date patient anatomy) can help clearly

visualize targets during an IGI, provide real-time quantitative feedback on organ motion and deformation, and permit real-time treatment monitoring. Achieving high-speed image registration is a fundamental need in not only interventional applications but virtually all applications.

Decades of research has led to significant success in developing accurate, reliable and fully automated image registration algorithms and techniques. The most recognized technique among these is intensity-based registration by maximizing MI (also referred to as MI-based image registration) between the two images to be registered [5-12]. However, due to the large size of 3D images and the computation intensive nature of this search-based technique, intensity-based image registration has remained too slow to be practical for clinical adoption. The slow execution has also prevented large-scale validation studies and clinical trials evaluating the quality and benefits of image registration.

Image Registration

Image registration is the iterative process of spatially aligning two or more images taken at different times, from different modality, or from different viewing angles. Maintz and Viergever [13] and Hill etal. [14] have presented a detailed summary of the medical image registration domain. Generally medical image registration can be classified into two main categories; extrinsic registration and intrinsic registration. Extrinsic methods based on other foreign objects that are not natively

a part of the imaged space; whereas intrinsic registration is based purely on the image information from the patients.

Extrinsic methods rely on artificial objects which may be attached to the patient or placed within the field of view of the image. These objects are specially designed to be well visible and accurately detectable in all pertinent modalities. The registration of the acquired images involves simply determining the corresponding translation between the external objects, which can be computed explicitly without the need for complex optimization algorithm. Therefore, this type of registration is comparatively easy, fast, and possible to be automated. However, extrinsic methods generally require advanced planning; provisions must be made at the time of preprocedural imaging. The marker objects are often invasive to the patients, while non-invasive marker options are generally less accurate. As extrinsic methods do not include patient-related image information, the nature of the registration transformation is mostly restricted to rigid transformation model only.

Intrinsic methods, on the contrary, rely on patient-generated image content only. Registration may be based on a limited set of identified salient points (landmarks), the alignment of segmented binary anatomical structures (segmentation based), or directly based on the image intensity values (voxel property based).

Landmark-based registration uses predefined salient points (landmarks) from the different images and determines the spatial transformation of the images with these paired points. Landmarks can be anatomical; accurately locatable points of the visible anatomy, which are usually identified interactively by the users. Landmark-based methods are often used to find rigid or affine transformations. Given a large enough set of points, they can also be used for more complex nonrigid transformation. The optimization procedure of landmark-based registration is relatively fast as the set of identified points is sparse compared with the original image content. However, this approach cannot be fully automated as user interaction is usually required for the identification of the landmarks.

Segmentation-based image registration methods aligns images based on the same anatomical structures (mostly surfaces and curves) extracted from the images to be registered. The alignment between the structures can be either rigid model based or deformable model based. The rigid model based approaches are the most popular methods in clinical use due to the success of the 'head-hat' method introduced for multimodal images. The nonrigid model based approaches elastically deforms the extracted structure from an image to fit the second image. Segmentation-based techniques are computationally efficient and they support multi-modal registration. However, the accuracy of registration highly depends on the segmentation accuracy. These methods cannot be fully automated as the segmentation step is often performed semi-automatically.

Voxel property-based methods operate directly on the image grey value without prior data reduction by the user or segmentation. The image grey value content is either reduced to a representative set of scalars and orientations, or the full image content is used. Theoretically, voxel property-based methods using the full image content are the most flexible methods which become the most interesting methods of current research. Voxel property based methods can be fully automatic, they also support multi-modal registration and are proven to be accurate. However, these methods are still limited from 3D clinical applications by the considerable computational costs. This thesis work will address this aspect through the use of the latest graphic processor units.

Intensity-Based Image Registration

Intensity based image registration is an automatic approach to spatially align two images based on their voxel grey levels. This method consists of numbers of optimization iterations with the aim to find the transformation parameters T_{opt} which optimally aligns the reference image *(RI)*, with the floating image *(FI)* by maximizing the similarity measure. The following equation summaries this process, where S is the similarity function to be maximized, T is the transformation operator applied to the reference image coordinates.

$$T_{opt} = argmax\ S(\ RI(x,y,z),\ FI\ (T(x,y,z))\ ) \qquad Eq.\ 2.1$$

Registration starts with an initial transformation *T*, which can be rigid model or nonrigid model, and maps the reference image voxels into the floating image space. The similarity measure *S* quantitatively determines the degree of misalignment between the images based on the voxel intensities (without considering any abstracted representation of the images). The optimization algorithm then updates the parameters of the transformation *T* based on the similarity measure result. This iterative process allows the optimization algorithm to search for the best transformation parameters that achieve the most optimal align between the two images. The major components of the image registration operation are depicted in Figure 2.1. Each component will be discussed in detail in the following sections.

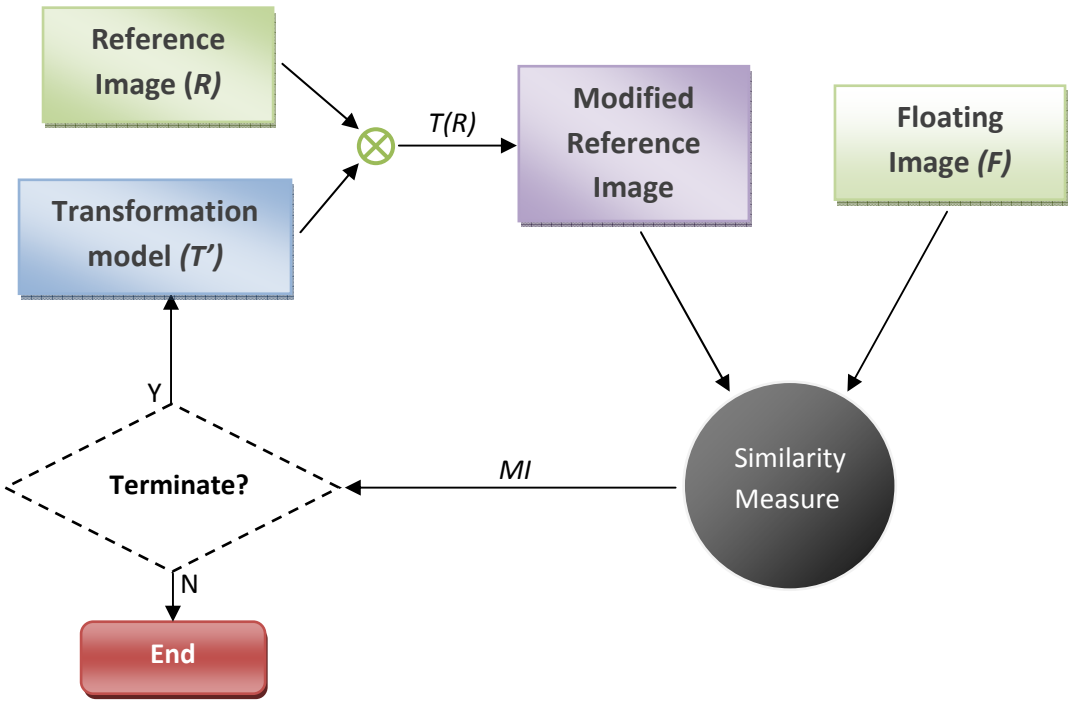

Figure 2.1: Image Registration Flow Diagram

Transformation Model

$$\boldsymbol{T}: (x,y,z) \rightarrow (x',y',z') \qquad \textit{Eq. 2.2}$$

Transformation model describes the spatial relationship between the reference and the floating images. These models are categorized according to their degrees of freedom. Rigid transformation includes only translations and rotations. Affine transformation advances rigid models by also including scaling and shearing. Perspective transformation is similar to affine transformation; however, the parallelism of lines need not be preserved [15].

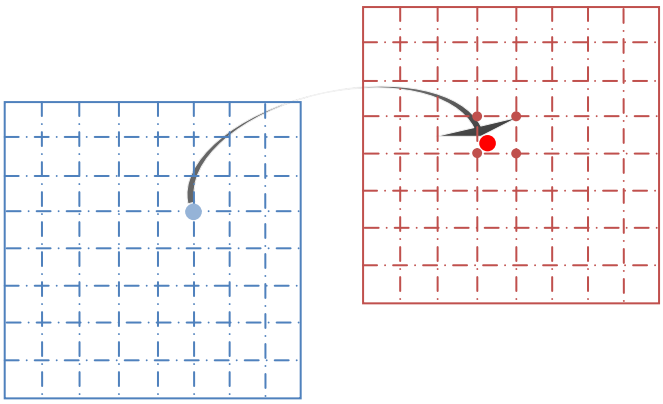

Figure 2.2: Transformation of Reference Image Voxels to Floating Image

Rigid Models

Translations and rotations suffice to register images of rigid objects, for example, bone or brain. For 3D image registration, rigid models can be described using a single constant 4x4 transformation matrix *(Eq 2.3, 2.4)*. Where *t* is an arbitrary translation vector and *r* is a 3x3 rotation matrix.

$$y = T_{rigid}\, x \qquad \textit{Eq. 2.3}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ 1 \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix} \qquad \textit{Eq. 2.4}$$

As rigid registration has reported numbers of successes [12], the limited degrees of freedom is not sufficient for applications where nonrigid transformations are required, such as modeling soft tissue movement and deformation.

Nonrigid (Deformable) Models

While rigid registration limiting the deformation to rotations and translations, nonrigid transformation models offers higher degrees of freedom to represent the misalignment between images. This provides the mean to model local deformations of the images which results high accuracy in registration result. One of the applications is intrasubject registration, when nonrigid transformations are required to accommodate any tissue deformation due to interventions or changes over time.

One of the well recognized nonrigid transformation models is Free Form Deformation (FFD) [2, 16]. FFD deforms an object by manipulating an underlying mesh of control points. The resulting deformation controls the shape of the 3D object and produces a smooth and continuous transformation. The set of control points controls the degree of nonrigid deformation which can be advanced by the resolution of the mesh of control points. A large spacing of control points

allows modeling of global nonrigid deformations, while a small spacing of control points allows modeling of highly local nonrigid deformations. However, this modeling approach suffers from high computational complexity. The resolution of control point mesh not only controls the finest of the local deformation field, but also defines the number of degrees of freedom, consequently, the computational complexity. For the case of 5x5x5 mesh of control points, the FFD model yields 375 degrees of freedom.

Another class of nonrigid registration algorithms is based on the concept of hierarchical image subdivision. These algorithms divide the image into numbers of sub-images (subvolumes) and perform local rigid (linear) transformation on individual subvolumes. Interpolation technique is applied to the result of individual transformed subvolumes to obtain the final smooth deformation field. Finer the resolution of the subvolumes, finer the local deformable registration can be achieved. These algorithms are computationally efficient and inherently suited well in the parallel computing framework as each subvolume can be processed independently from other subvolumes of the same hierarchical level. In this thesis, we have considered one of the hierarchical volume subdivision deformable registration algorithms proposed by Walimbe and Shekhar [4]. The details of this algorithm will be discussed in the later session.

Similarity Measure

Automatic registration requires a metric for measuring the degrees of similarity between the images to be registered at each iteration of the optimization process. The metric is used to guide the optimization engine to approach the optimal alignment. Ideally, the similarity measure attains its maximum (or minimum) when the images are perfectly aligned and deviates as the images are less overlaid. Similarity measure for intensity-based registration is computed directly from the voxel intensity values of the images rather than from geometrical structures. Some of the commonly used metrics are sum of squared difference (SSD), normalized cross correlation (NCC), correlation ratio (CR), and mutual information (MI) [17, 18]. One of the decision factors on selecting the proper similarity measure is the use of image modality; whether the images are taken with the same or different type of imaging modalities.

For intramodality images, registration is to compare the images of a subject taken at different time with the same modality. If there is no change in the subject, or the images are properly aligned, the difference image, subtraction of the reference image and the transformed image, will result no structural difference except for noise. The amount of residue in the difference image corresponds to the amount of registration error. The iterative optimization process will calculate the optimal transformation T to minimize the residue in the difference image. The computation of similarity measure techniques for intramodality images is more straightforward and highly parallelizable.

For intermodality registration, there is no simple relationship between the intensities of the two images to be registered. The registration error cannot be quantified simply by deriving a difference image. Mutual information (MI) based approach introduced by Collignon et al. and Wells, et al. provides a sufficient similarity metric applicable for both intramodality and intermodality registration.

Mutual Information

Mutual information (MI) is a basic concept from information theory based on the concept of entropy. MI measures the statistical dependency between two random variables or the amount of information that one variable contains about the other. In the case of intensity-based image registration, the MI of the intensity of the voxel pairs is maximal when the two images are properly aligned.

Entropy is a measure of information of message developed from communication theory. This concept can be interpreted as a measure of the amount of information an event gives, the uncertainty about the outcome of an event, and the dispersion of the probabilities. Shannon introduced an entropy measure in 1948 [19], which weights the information per outcome by the probability of that outcome. Given events $e_1, ..., e_m$ occurring with probabilities $p_1, ..., pm$, the Shannon entropy is defined as

$$H = -\sum p(x) \cdot \ln(p(x)) \qquad \textit{Eq. 2.5}$$

When the concept of entropy applied to images, the distribution of the grey values of the image is concerned. The probability distribution of the grey values is the number of occurring of each grey value in the image divided by the total number of occurrences. An image with small variance in grey values has a low entropy value; it contains very little information. Whereas an image contains a lot of information, with even distribution of different grey values, the entropy value will be high. Entropy also describes the dispersion of a probability distribution. The entropy value is low when the distribution has a few dominant peaks and it is maximal when all outcomes have an equal chance of occurring.

Image registration adopts this concept as similar measure criterion by measuring the information of the joint probability distribution of the images to be registered. Woods et al. [20-21] first introduced the idea of using the grey values ratio for similarity measure back in the early 1990s. Hill et al. [22] later adapted Woods' measure and proposed the technique of constructing a feature space which is a two-dimensional plot showing the combination of grey values in each of the two images for all corresponding points. As the alignment of the two images changes, the feature space (joint histogram) also changes. When the images are optimally registered, the joint histogram will show certain clusters of grey values. As the degree of misregistration increases, the joint histogram shows increasing dispersion. Figure 2.3 shows an example of a joint histogram of an MR image registered with itself. When the images are perfectly registered, the distribution of the joint histogram forms on the diagonal as the images are identical (Figure 2.3a).

When one of the images rotates, the resulting histogram starts to disperse (Figure 2.3b). As the degree of rotation increases, the amount of dispersion of the joint distribution increases (Figure 2.3c-d).

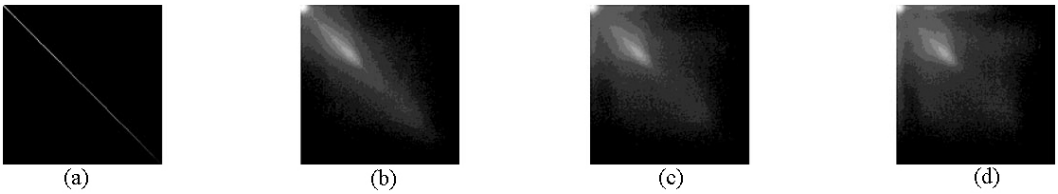

Figure 2.3: Joint Histogram of an MR image with itself
with different degree of image alignment. [12]

Collignon etal. [23] and Studholme et al. [24] suggested using entropy in image registration by measuring the dispersion of the joint probability distribution. When the joint distribution has a few dominant peaks (better registered), the entropy value is low. As the images deviate away from the proper alignment, the entropy of the joint distribution increases. The Shannon entropy definition is derived as *Eq. 2.6* for a joint distribution. Proper registration of images is achieved by obtaining the transformation which minimizes their joint entropy.

$$H = -\sum\sum p_{RI,FI}(x,y)\cdot \ln\left(p_{RI,FI}(x,y)\right). \qquad \textit{Eq. 2.6}$$

Shortly after the knowledge of using joint entropy in image registration was proposed, Collignon et al. [25,26] and Viola and Wells [27-29] introduced the use of mutual information for image registration.

The definition of MI is frequently presented in the following forms, where *H(.)* denotes as entropy (Shannon's entropy).

$$MI(RI,FI) = H(RI) - H(RI|FI), \quad \textit{Eq. 2.7}$$

$$MI(RI,FI) = H(FI) - H(FI|RI), \quad \textit{Eq. 2.8}$$

$$MI(RI,FI) = H(RI) + H(FI) - H(RI,FI), \quad \textit{Eq. 2.9}$$

where the individual and mutual entropies are calculated as:

$$H(RI) = -\sum p_{RI}(x) \cdot \ln\left(p_{RI}(x)\right), \quad \textit{Eq. 2.10}$$

$$H(FI) = -\sum p_{FI}(x) \cdot \ln\left(p_{FI}(x)\right), \quad \textit{Eq. 2.11}$$

$$H(RI,FI) = -\sum\sum p_{RI,FI}(x,y) \cdot \ln\left(p_{RI,FI}(x,y)\right). \quad \textit{Eq. 2.12}$$

In this thesis, the third definition of MI *(Eq. 2.9)* is chosen as this definition is most closely related to joint entropy and most applicable for image registration application. This form of representation consists of the term *H(RI,FI)*, which means that maximizing mutual information is related to minimizing joint entropy. As recalled in our earlier discussion, the joint histogram of two images' grey values disperses when they are misaligned which result increases of entropy and decreases of MI.

To further improve the stability of the measure criterion, Studholme et al. [30] proposed a normalized measure of mutual information. The size of overlapping part of the images influences the mutual information measures in two ways. As the area of overlap decreases and the number of samples decreases, the statistical power of the probability distribution estimation is reduced. Also, Studholme et al. [9, 30] have shown that the mutual information measure might increase with

increasing misregistration. This occurs when the relative areas of object and background even out and the sum of the marginal entropies increases, faster than the joint entropy. Normalized mutual information (NMI) has addressed these issues as it is less sensitive to change in overlap.

$$NMI(RI, FI) = \frac{H(RI) + H(FI)}{H(RI, FI)} \qquad Eq.\ 2.13$$

Mutual information similarity measure enables full automatic registration on a large variety of applications as prior segmentation (manual marker identification) is not required. It advances other similarity measure particularly in intermodality registration, as no assumption is made regarding the nature of relation between the image intensities in both modalities. Holden et al. [31] have demonstrated that mutual information-based techniques are, in general, superior to other techniques for deformable image registration. A comprehensive survey of MI-based registration was presented by Pluim et al. [12].

Interpolation

The joint image intensity histogram of the image volume is constructed by binning the image intensity pairs *(RI(x,y,z), FI (T(x,y,z)))* for all overlapping voxels. In general, $T(x,y,z)$ will not coincide with a grid point (integer coordinate), interpolation techniques would be needed to obtain the corresponding image intensity value. Nearest neighbor (NN), trilinear (TRI), and partial volume (PV)

interpolation schemes have been traditionally used for this purpose. Figure 2.4 shows the differences of these three schemes.

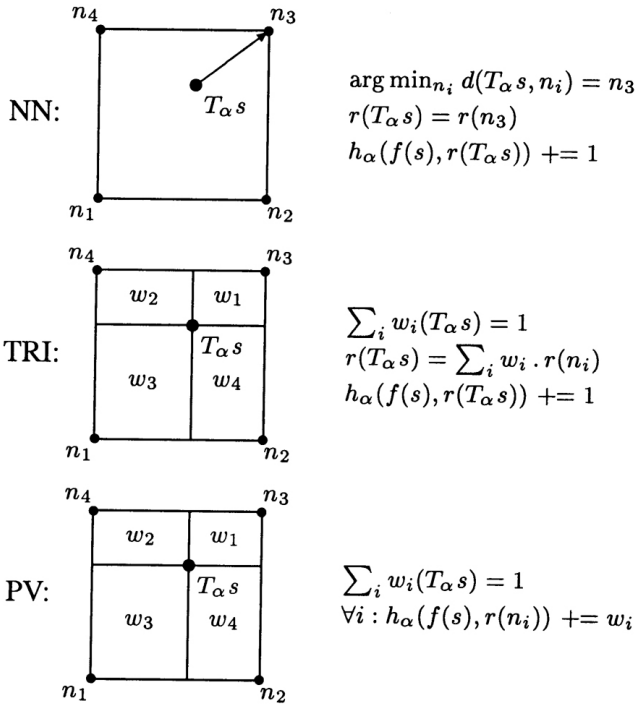

Figure 2.4: Different Interpolation Schemes (NN, TRI, PV) [8]

Nearest neighbor interpolation is the most straightforward approach, but it is insufficient to guarantee subvoxel accuracy. Trilinear interpolation may introduce new intensity values which are originally not present in the floating image. This effect would lead to unpredictable changes in the marginal distribution of the floating image. Partial Volume interpolation, proposed by Collignon [17], was specifically designed for creating joint histograms. Instead of computing a weighted intensity value and updating a single histogram entry, it uses the same weights of trilinear interpolation for fractional updates of the histogram entries corresponding to the neighbors of a transformed point. The contribution of the

image intensity $R(x,y,z)$ to the joint histogram is distributed over the intensity values of all eight nearest neighbors (for 3D case) of $T(x,y,z)$ on the grid of the floating image. PV interpolation creates smoother changes of the joint histogram for varying transformations and hence a smoother registration function. This scheme has been reported to produce the least interpolation error for MI-based registration [8, 32].

Consequently, PV interpolation scheme increases the number of memory accesses of the MH accumulation operation by approximately eight times, and floating point implementation is required to represent the fractional updates of the histogram entries. These differences of the interpolation schemes have direct impact on the performance, therefore, special techniques would be considered to ease the burden in memory bandwidth and compute resources.

Optimization Algorithm

Optimization algorithms are used to navigate the search space of transformation parameters. They identify the optimal combination of transformation parameters that best aligns a pair of images. In the case of multimodality intensity-based image registration, the voxel similarity function of the mutual information is the objective function to be optimized. The algorithm chosen in our implementation is the downhill simplex method.

The downhill simplex method is a multidimensional nonlinear optimization technique first introduced by Nelder and Mead [33]. This method uses the concept of a simplex, which is a special polytope of *N+1* vertices in an *N*-dimensional space. The simplex method places an initial simplex in the solution space and takes a series of steps to move the vertices towards the local optimum. Shekhar et al. [34, 35] and Walimbe et al. [4, 36] have reported successful use of this optimization technique for voxel similarity–based image registration.

Hierarchical Volume Subdivision Based Algorithm

Hierarchical volume subdivision based nonrigid image registration algorithms are inherently faster compared to most of the intensity based nonrigid registration algorithms. The framework of these algorithms is well adaptable to the architecture of parallel computing, which significantly advances the performance in computation. This class of nonrigid registration algorithms involves modeling the elastic transformation between images as an interpolation of multiple local rigid-body registrations [37]. One or both of the images to be registered are divided into subimages (subvolumes) which will be registered independently. The final non-linear transformation field is generated from the independent registration solutions from the subimages using various interpolation techniques.

Most of the earlier proposed volume subdivision based algorithms allows only translation-based model for the subvolumes due to the complexity in direct interpolation of 3D rotation. With only three degrees of freedom locally, very

small subvolumes are needed in order to recover extremely complex misalignment. However, similarity measures like MI lack the statistical power for robust registration as the subvolume size decreases. Walimbe and Shekhar [4] suggested to enhance this model by using the six-parameter rigid body transformation model throughout for registration of individual subvolumes and incorporating a quaternion-based scheme for direct interpolation of the resulting rigid body transformations for generation of the deformation field.

In the case of 3D image registration between two images, the references image *(RI)* and the floating image *(FI)*, this hierarchical subdivision based algorithm first recovers the global mismatch between the two images, followed by a series of refinement of the local matching. The reference image is divided by performing hierarchical octree-based subdivision. The subvolumes at each hierarchical level are registered to the undivided floating image. Localized misalignments are captured by using the six-parameter translation and rotation transformation models at the global (traditional rigid registration) as well as the local levels. The concept of the hierarchical registration scheme is presented in Figure 2.5.

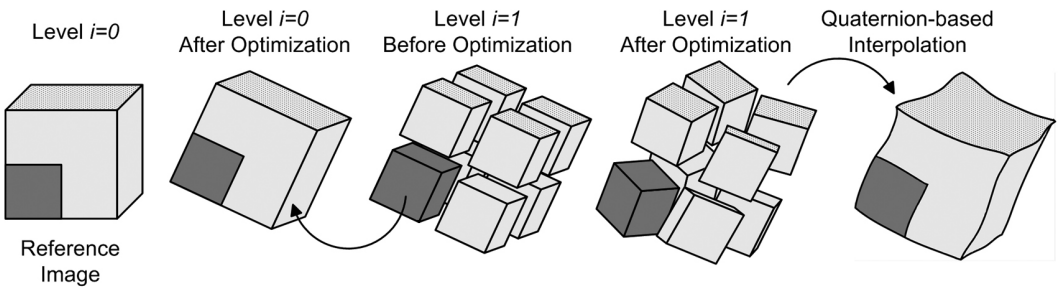

Figure 2.5: Hierarchical volume subdivion based nonrigid registration

Volume subdivision and subvolume registration steps continue until the voxel count of an individual subvolume reaches the predefined limit. The final deformation field obtained by quaternion-based interpolation of the individual subvolume transformations at the final hierarchical level is used to deform the floating image to match the reference image.

Technique in Calculating MI for Subvolumes

As the voxel count of an individual subvolume decreases, the accumulated mutual histogram for similarity measure becomes sparse, thus resulting unreliable mutual information for robust registration. A concept of MH_{rest} (MH-rest) is introduced to resolve this issue by taking information from all image voxels into the local registration problem of a given subvolume. The mutual histogram is compiled not only with the voxels of the given subvolume, but with the sum of two mutual histograms; $MH_{subvlume}$ and MH_{rest}. $MH_{subvlume}$ is compiled with the voxels of the subvolume being registered, and MH_{rest} is calculated with all the remaining voxels of the image based on the transformations obtained from the preceding hierarchical level. For a given subvolume registration, $MH_{subvlume}$ will be evolved as a function of the subvolume transformation model while MH_{rest} remains constant during the iterative optimization process as it is independent from the current hierarchical level. The resulting MI from the combined MH is computed over the entire image with local variations corresponding to the subvolume under optimization. This approach increases the statistical power of the calculated MI

and provides additional guidance to the registration process. *Eq. 2.14* summarizes this process. The contribution of current subvolume p at level i to the MH is computed based on the candidate transformation T_p^i. The contribution from the rest of the subvolumes to the MH remains constant during the optimization process.

$$MH^i{}_{Total_p} = MH^i{}_{Subvolume_p} + MH^i{}_{\mathrm{Re}\,st_p} \qquad Eq.\ 2.14$$

$$MH^i{}_{Subvolume_p} = Accumulate\left(T_p^{\,i}\right) \qquad Eq.\ 2.15$$

$$MH^i{}_{\mathrm{Re}\,st_p} = Accumulate\left(T_{\bar{p}}^{\,i-1}\right) \qquad Eq.\ 2.16$$

Graphic Processing Unit (GPU)

A graphics processing unit (GPU) is best known for its compute-intensive and high parallel computation capability for computer graphics applications. Fueled by the desire for real time, high-definition 3D graphics, GPUs have evolved into a highly parallel, multithreaded, multicore processor with tremendous computational power and very high memory bandwidth [38]. With the latest improvement in floating-point performance and increases in programmability flexibility, GPUs have also become the processor of choice for accelerating many non-graphics data parallel applications; especially favorable for numerically intensive scientific applications. Nieuwpoort and Romein introduce GPUs to radio astronomy signals correlation applications [39]. Levine et al [40] expand the GPU compute capacity to modular dynamic simulation acceleration applications and attain twenty times performance improvement compared to the use of single CPU. In this thesis, we focus on the GPUs developed by NVDIA and build our

knowledge of CPU-based image registration algorithm on the latest NVDIA GPU architecture.

GPU outperforms CPU in compute capability by devoting more transistors to its arithmetic logic units (ALUs) for data processing at the expense of reduced data caching and flow control (Figure 2.6). This makes the GPU architecture especially well suited for addressing problems that can be expressed as data-parallel programming model. The high arithmetic intensity in data-parallel computations can effectively hide the memory access latency even though the capacity of data caching is comparatively less sufficient in GPU architecture.

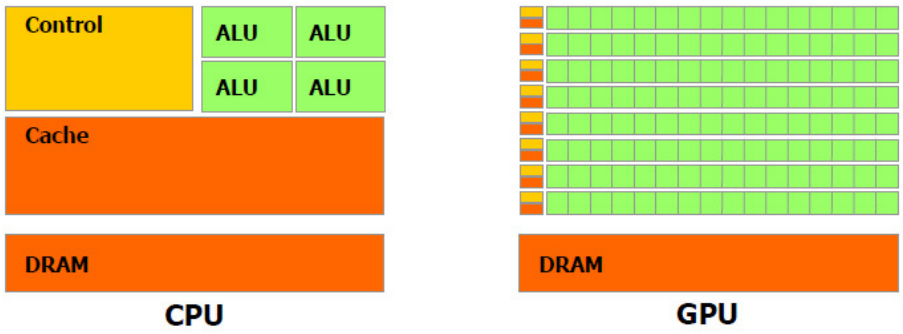

Figure 2.6: The GPU Devotes More Transistors to Data Processing [38]

CUDA Overview

Compute Unified Device Architecture (CUDA) is a general-purpose parallel computing hardware and software architecture developed by NVIDIA. This architecture introduces a new parallel programming model and instruction set architecture, which leverages the parallel compute engine to solve many complex computation problems in a more efficient way on a GPU than on a CPU. CUDA

programming model introduces abstractions, such as a hierarchy of thread groups, shared memories, and barrier synchronization, to cover fine-grained parallelism (thread parallelism) to coarse-grained parallelism (task parallelism). These abstractions guide programmers to partition their problems into coarse sub-problems. Each sub-problem can be solved in parallel by groups of threads (thread blocks). Each block of threads can be scheduled on any of the available processor cores with no specific constraint on execution sequence. Multiple sub-problems can be computed in series or in parallel depending on the version of the GPU architecture. A CUDA program therefore can be seemingly executed on any number of processor cores. This allows applications to transparently scale their parallelism to leverage the increasing number of processor cores.

CUDA abstracts the parallel programs to be run on the GPU as kernels. A kernel executes in parallel across a set of parallel threads which are organized in a grid of user-defined 1D/2D array of thread blocks. A thread block consists of a set of concurrently executing threads which can cooperate among themselves through per-block shared memory space and barrier synchronization. A grid of thread blocks shares results in the global memory space. Figure 2.7 shows the structure of the thread hierarchy.

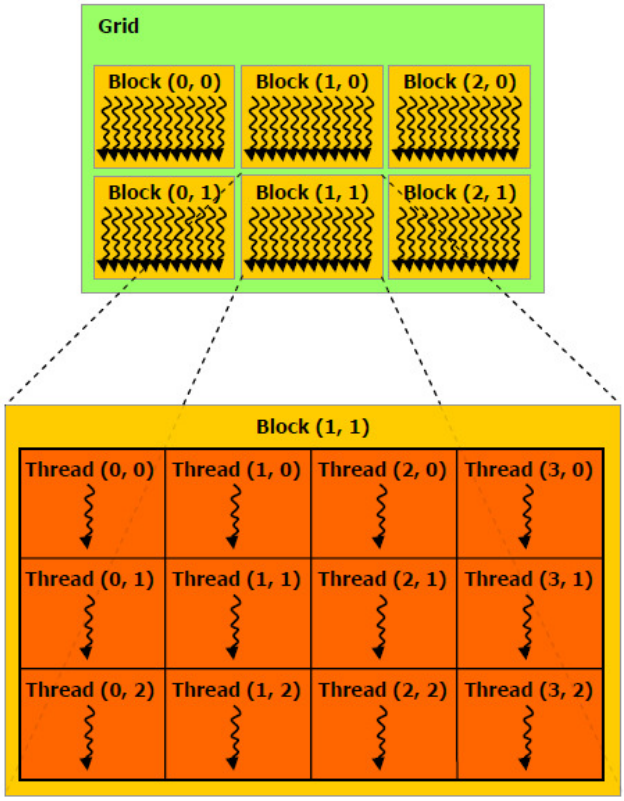

Figure 2.7: Thread Hierarchy [38]

The CUDA hardware architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). Each SM consists of 8 to 32 Scalar Processors (SPs), 2 to 4 special function units (SPU) for transcendental instructions, an instruction unit and on-chip shared memory. Each CUDA processor has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU). The total number of scalar processors and on-chip shared memory varies among generations of GPU architectures. Figure 2.8 illustrates the hardware model of the multiprocessor.

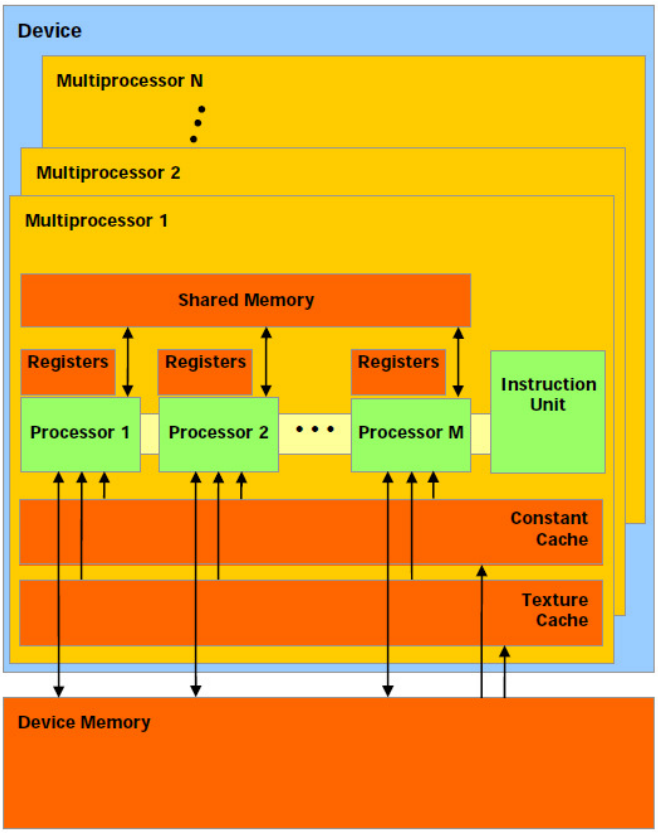

Figure 2.8: CUDA Hardware Model [38]

Programming Model

A typical CUDA implementation consists of the following steps (Figure 2.9). First, memory is allocated on the GPU (device). The CPU (host) then transfers the data from the host to the device and initialized the device memory if required. Next, the host determines the execution configure, i.e. the number of thread blocks and block size, and invokes the kernels. The device executes the kernels and stores the result in the device memory. When the computation is completed, the resulting data is transferred from the device to the host.

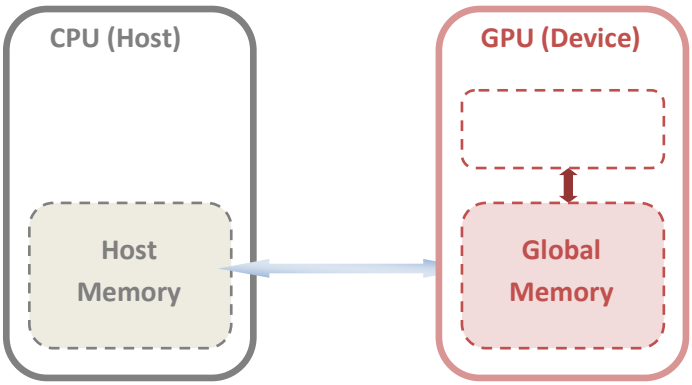

Figure 2.9: CUDA Programming Model

When a CUDA program on the host CPU invokes one or more kernel grids, the thread blocks of the kernel grids are distributed to the available streaming multiprocessors with sufficient memory resources. The threads of a thread block execute concurrently on scalar processors of a single steaming multiprocessor, and multiple thread blocks can execute on one multiprocessor if resource is available. Figure 2.10 shows the relationship between threads and processors in the CUDA architecture.

A multiprocessor is designed to execute hundreds of threads concurrently. To manage such a large number of threads running several different programs, the multiprocessor employs a new architecture called SIMT (Single-Instruction, Multiple-Thread) architecture. The multiprocessor maps each thread to one scalar processor, and each thread executes independently with its own instruction address and register state. SIMT enables programmers to write thread-level

parallel code for independent, scalar threads, as well as data-parallel code for coordinated threads.

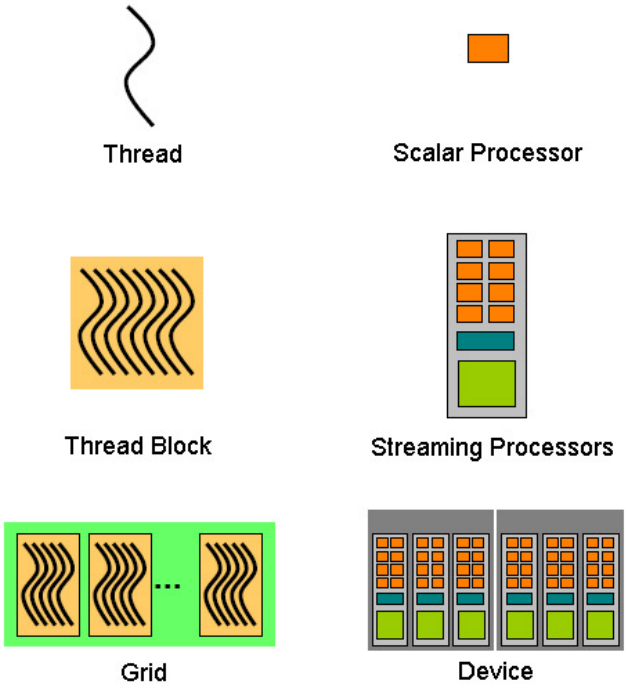

Figure 2.10: CUDA Software and Hardware Architecture Relationship

The multiprocessor manages, schedules, and executes threads in groups of 32 parallel threads called warps. A half-warp refers either to the first or the second half of a warp. When one or more thread blocks are assigned to a multiprocessor, the processor partitions the threads into warps, which will be scheduled by the warp scheduler for execution. A warp executes one common instruction at a time; therefore it reaches the most efficient performance when all 32 threads of a warp follow the same execution path and access memory in nearby addresses. When some of the threads of a warp diverge at a conditional branch, the warp serially

executes the different branch paths, then converges back to the same execution path when all the paths complete.

The number of blocks and warps that can reside and be processed together on the multiprocessor depends on the number of registers and shared memory available on the multiprocessor and the amount of registers and shared memory used by the kernel. The multiprocessor also has a limit on the number of resident blocks and the number of resident warps. These limitations and the amount of memory resources vary among generations of CUDA architecture. In this thesis, our discussion will focus on the latest CUDA compute architecture code named *Fermi*™.

Fermi Architecture

The Fermi architecture is NVIDIA's latest generation of CUDA architecture. Building upon the knowledge from the prior generations of processors, the new architecture has taken new approaches in design and achieved significant improvement in compute power through architectural innovations. With the increases in programmability and compute efficiency, the Fermi architecture is known to be the world's first computational GPU [41].

To target the performance in programmability and compute efficiency, Fermi architecture introduces the third generation streaming multiprocessor. The Fermi based GPUs consist of up to 16 Streaming Multiprocessors (SMs). Each SM consists of 32 Scalar Processors (SPs); four fold increases compared to the prior generation, and 4 special function units (SFU). This features a total of 512 SPs per GPU (Figure 2.11). To achieve near peak hardware performance, two warp schedulers and two instruction dispatch units are introduced to allow two warps to

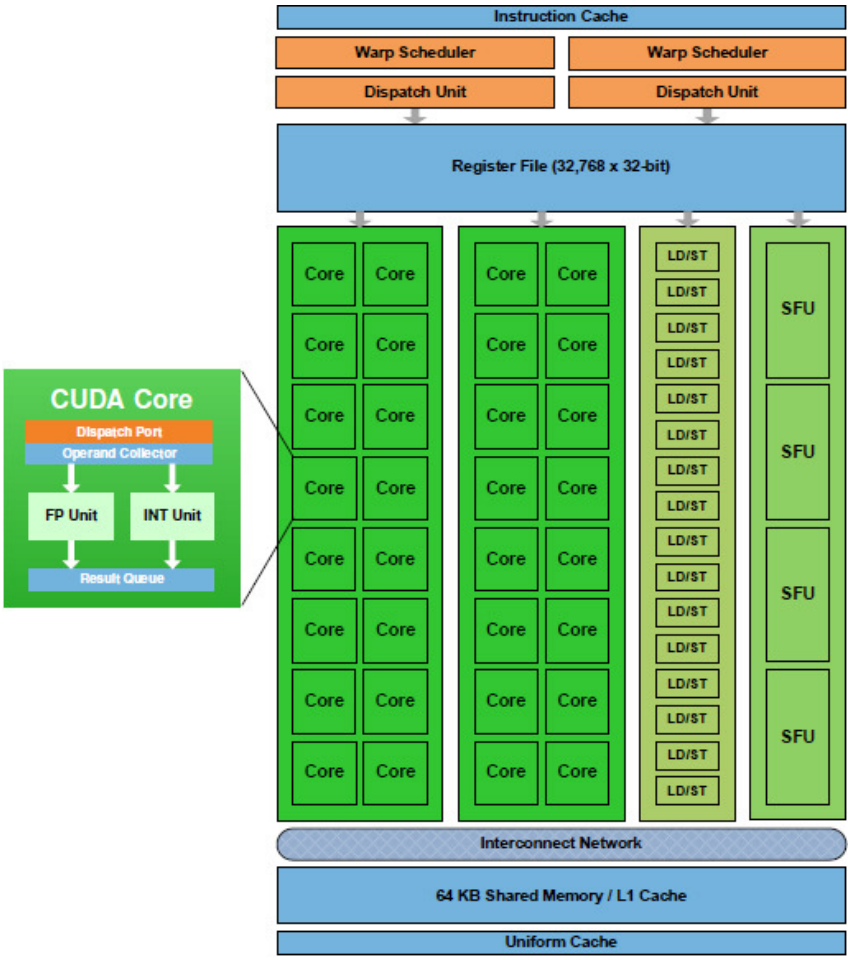

Figure 2.11: Fermi Streaming Multiprocessor [41]

be issued and executed concurrently. Double precision floating point arithmetic performances have also advanced up to 16 double precision multiply-add operations per SM, per clock, translated to 8x performance over the previous generation. 64KB of RAM with configurable partitioning of shared memory and L1 cache further extends the data sharing capacity among threads of the same thread block to greatly reduce off-chip traffic.

Memory subsystem of the Fermi architecture also demonstrates major improvements on memory hierarchy, called Parallel DataCache hierarchy, and atomic memory operation performance. The Parallel DataCache hierarchy, which consists of per-SM configurable L1 caches and unified L2 cache, provides a single unified memory request path for loads and stores and greatly improves performance over direct access to DRAM. Atomic memory operations allows concurrent threads to correctly perform read-modify-write operations on shared data structure which is one of the important elements in parallel programming. In the new architecture, more atomic units are available in hardware. Along with the addition of the L2 cache, atomic operations performance is up to 20x faster compared to the prior generation architecture.

In this thesis, our implementation and analysis result is based on the NVIDIA GeForce GTX 480 device. Table 2.1 highlights the hardware specification of this device along with the previous generation of GTX 285.

	GeForce GTX 285	GeForce GTX 480
Transistor count	1.4B	3.0B
Process node	55 nm @ TSMC	40 nm @ TSMC
Core clock	648 MHz	700 MHz
Memory clock	1300 MHz	924 MHz
Memory transfer rate	2600 MT/s	3696 MT/s
Memory bus width	512 bits	384 bits
Memory bandwidth	166.4 GB/s	177.4 GB/s
CUDA processors	240	480
SM count	30	15
Special Function Units (per SM)	2	4
Shared Memory (per SM)	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	Configurable 16 KB or 48 KB
L2 Cache	None	768 KB
Global Memory	1024 MB	1536 MB
Peak single-precision FLOPS	0.708 Tflops	1.35 Tflops
Peak double-precision FLOPS	88.5 Gflops	168 Gflops

Table 2.1: GeForce GTX285 and GTX480 Hardware Specification

Related Work

The increasing programmability and compute efficiency of the GPU architecture has attracted many researchers in adapting this computing model to registration algorithms. Earlier work in this area was restricted in mapping the programs (non-graphic applications) to graphics processing pipeline in terms of vertex and fragment shaders. This approach suffers from substantial programming overhead and results in modest performance improvements over CPU-based implementations.

Latest software platforms for GPU programming, NVIDIA's CUDA and AMD/ATI's Brook+, have drastically changed the programming paradigm to resolve this limitation by increasing the programmability. The programming environment is C/C++-like and it is easy to be upgraded for future generations of hardware. These programming model and architectural changes enable GPUs to general-purpose programming for various non-graphic applications. As NVIDIA's CUDA has been exclusively adopted by the research community, the papers referenced in this thesis are all developed based on CUDA.

GPUs are equipped for speeding up geometric transformations part of the image registration process, so the computation efficiency of similarity measures becomes a critical knob of the overall performance. For single-modality similarity measure implementation, the algorithm falls naturally onto the parallel architectures and the entire registration process can be efficiently parallelized. The input data can be processed independently, ideal for the SIMT architecture, with only a final reduction step to generate the result. Plishker et al. [42] have reported a CUDA implementation of SSD based rigid and nonrigid registration. Muyan-Ozcelik et al. [43] have reported a CUDA implementation of Demons deformable registration algorithm.

Compared to single modality image registration, similarity measures for multi-modality image registration require statistical measure like mutual information

(MI) is not a trivial translation on GPUs. Efficient mutual histogram compilation scheme on GPUs previously proposed have involved special handling on data distribution, data alignment, subsampling, memory/cache hierarchy, etc.. However, performance improvement usually comes in the cost of reduced accuracy by using smaller sample size (image voxels subsampling) or fewer histogram bins (less precise image intensity). Lin and Medioni [44] presented an implementation of Viola's [18] MI approximation method based on stochastic sampling of image intensities and Parzen windowing. Shams et al. proposed an approximate histogram computation method to speed up MI computation [45]. Without tolerating the lost in registration accuracy, Sham et al [46] later presented the *'sort and count'* method for histogram computation on the entire data set. This method sorts blocks of data with a parallel sort algorithm before writing to the memory to ease the need for synchronization or atomic operations.

A comprehensive survey of MI-based registration on GPU was presented by Shams et al. [37]. Table 2.2 highlights the recently reported results of rigid and nonrigid MI-based image registration implementations. As the GPU architecture has been rapidly evolving in the past few years, inter-architecture performance comparison of reported results becomes challenging. Research groups generally benchmark their implementations with the latest available GPU architecture compared with the CPU implementation, but rarely compared across GPU platforms. Hardware, software, and compiler improvement in different versions of the GPU architectures would significantly alter the performance results. Besides,

most groups report their speedups for the entire registration algorithm and for specific data sets. Comparison of different results is further complicated as other techniques might have been implemented for further speed up; for example multi-resolution scheme, specific convergence criteria for optimization algorithm, etc.. To better present the performance results, normalized results are given in terms of average execution time in milliseconds per mega-voxel per iteration of the optimization algorithm (ms/MVoxel/itr).

Group	Hardware	Pref.	Techniques
SHAMS [45]	GTX 8800 (16 MP/ 128 CORES)	6.17	MI ESTIMATED BY BIN SAMPLING
LIN [44]	GTX 8800 (16 MP/ 128 CORES)	–	MI ESTIMATED BY SAMPLING
SHAMS [46]	GTX 280 (30 MP/ 240 CORES)	4.06	MI COMPUTED USING BITONIC SORT AND COUNT

(a)

Group	Hardward	Pref	Technique
VETTER [48]	GTX 7800	2860	COMBINED MI AND KULLBACK-LEIBLER MEASURE
FAN[49]	GTX 8800 ULTRA (16 MP/128 CORES)	324	COMBINED MI AND KULLBACK-LEIBLER MEASURE

(b)

Table 2.2: Summary of Recently Reported GPU/MI-based Image Registration (a) Rigid (b) Nonrigid Image Registration [47]

For rigid image registration, the *sort and count* technique proposed by Sham et al has presented the most significant performance improvement. While it performs well on registrations with nearest-neighbor interpolation (each input data point results in one output data), the performance gains degrades for registrations with

partial volume interpolation (each input data point results in eight output data) which is a crucial component for MI-based nonrigid registration. The execution time of the *sort and count* operation on the larger output data set dominates the overall registration runtime. For nonrigid registration, the solutions proposed by Vetter et al [48] and Fan et al [49] reported noticeable speedup over the CPU implements, however, the performances is still far from acceptable for real-time applications.

In addition, the latest GPU architecture, for example NIVIDA *Fermi*, continues to report improvements in both hardware and software critical for general purpose computing. The new memory hierarchy in particular allows efficient data caching which shows significant performance improvement in atomic update operations. These technological improvements do not only scaling the throughput of the existing implementations; they also give researchers the flexibility in exploring innovative approaches for better designs and implementations of parallel image registration algorithms.

Validation

The GPU-based implementation presented in this thesis will be compared with a CPU and a FPGA implementation in terms of execution time and registration accuracy. Table 2.3 shows the hardware used for validation. To better present the speedup without the dependence on the size of images involved and the number

of optimization iterations, the execution time is normalized and averaged in milliseconds for a single iteration for processing 1,000,000 voxel pairs (ms/MVoxel/itr).

GPU	*GTX480, 1GB Memory, 15 SMs, 480 SPs, 48KB*
CPU (host)	*Intel Xeon 2.33GHz, 4GB RAM*
FPGA	*Altera Stratix II FPGA 200MHz , 1GB RAM*

Table 2.3: Hardware Specification

Five artificially deformed CT-CT image pairs are used as test cases. The known deformation fields of these testsets are used as reference. The RMS of the resulting deformation fields from different implementations will be computed against the reference data.

Chapter 3: MI-based Image Registration on GPU

This chapter presents a GPU-based implementation for mutual information based rigid image registration. First, we discuss previously reported works and their limitations. Next, we present various considerations of our implementation to show how our implementation maps rigid registration algorithm in general to the GPU architecture. This implementation approach not only can fully exercise the GPU compute capacity, but also retain the scalability. Finally, we compare the performance of this implementation with earlier reported result.

Motivation

The maximization of mutual information is the core computation in both rigid and nonrigid intensity-based image registration. The most common approach to computing MI is mutual histogram-based approach. The mutual histogram accumulation process requires intensive computation power and excessive memory accesses. Various multiprocessor solutions (multicore processor, CPU cluster and GPU) have been proposed over the years to accelerate the parallelizable computation; however, these solutions have not fully addressed the limitations on memory access and thus provide moderate acceleration at best.

Mutual information measures the similarity in a pair of images by first constructing the mutual (joint) histogram and then determining the joint and individual probability distribution functions (pdfs) and entropies. In the maximization process, the result forms the input to the optimization engine that

computes the next set of candidate registration parameters. The process repeats until the maximum of mutual information is reached.

In every iteration of the optimization algorithm, every voxel in the reference image is mapped to one voxel or a group of voxels, depending on the interpolation scheme, in the floating image. Based on the intensity of voxels, one or several bins of the mutual histogram are updated. For an image of size n, the computation time is on the order of $O(n)$, which is also the order of the number of memory accesses.

Previous multiprocessor solutions, CPU cluster in particular, accelerate this process by dividing the images into a number of subimages (subvolumes) and distributing the data across the cluster (Figure 3.1). Each processor is responsible for processing one part of the image while managing its own partial mutual histogram. These individual partial mutual histograms are sent back to the host machine at the end of the process and combined into a single mutual histogram. The speedup of this approach is generally limited by the number of processors in the cluster as the number of parallelizable subimages is constrained by the number of processors and each processor processes the subvolumes voxels in series. As the number of subvolumes goes up, the number of partial mutual histograms, which needed to be consolidated at the end of the process, also goes up.

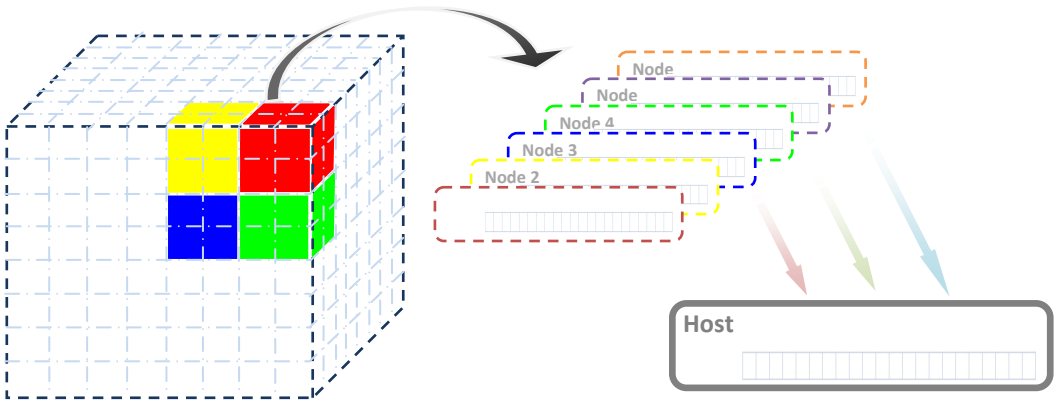

Figure 3.1: Block distribution scheme for CPU cluster implementation with distributed partial mutual histograms

With the parallel nature of the voxel coordinate transformation computation, GPU, which offers massive parallel compute power, appears to be a suitable platform for the MI problem. Ideally, with the GPU architecture, thread blocks can be visualized as CPUs in the cluster. The image is again divided into a number of subvolumes and each thread block is assigned a specific subvolume (Figure 3.2). Voxels in each subvolume are processed in parallel by the threads of the assigned thread block. Depending on the number of threads available in each block, each thread processes one or more voxels until all voxel in the subvolume are covered. Results are updated onto the partial mutual histogram residing at each block (shared memory). The partial mutual histograms will eventually be combined into a single mutual histogram in the main memory (global memory).

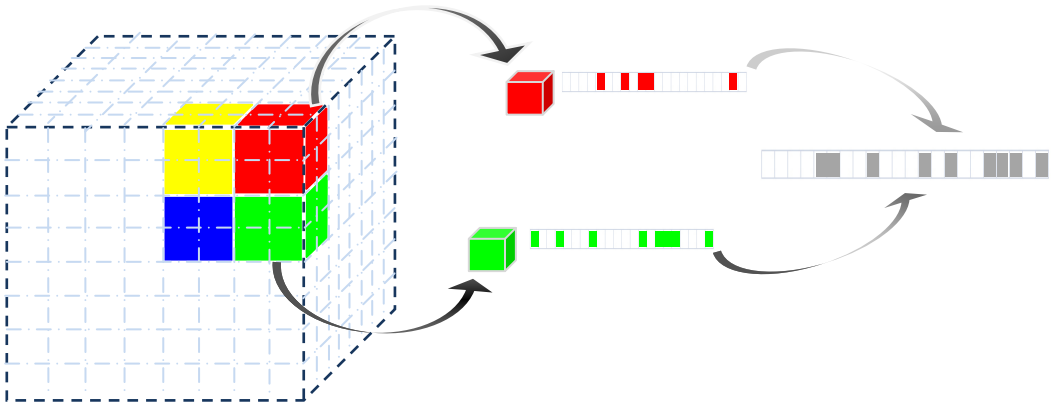

Figure 3.2: Block assignment scheme for GPU implementation with distributed partial mutual histograms in shared memory

Although the number of threads available is virtually unlimited, the amount of shared memory available for each thread block is constrained by the current technology. Medical images are either natively 8-bit or converted down to 8 bits for intensity-based image registration. This translates to a mutual histogram size of 256 x 256 bins, which far exceeds the available shared memory size offered by the current GPU architecture for accumulating partial mutual histogram in a thread block, regardless of the data type.

An obvious alternative is to store the partial histograms directly in the global memory instead of the shared memory (Figure 3.3). The drawback is the global memory access latency penalty ranges from 200 to 300 cycles. Besides, every block has a dedicated partial mutual histogram in the global memory; the number of blocks is now limited by the global memory size which hinders the scalability of this approach.

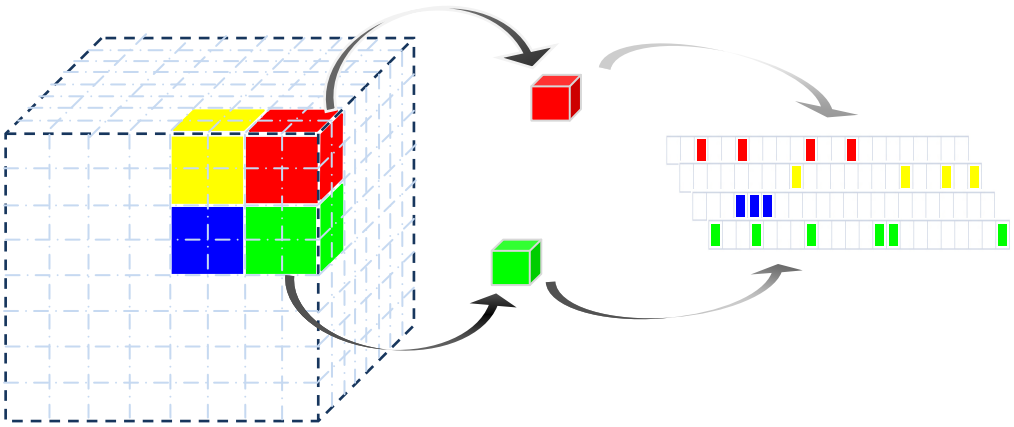

Figure 3.3: Block assignment scheme for GPU implementation with partial mutual histograms in global memory

Depending on the GPU architecture, atomic update can be a feasible solution for easing the constraint caused by the global memory size. Instead of assigning a dedicated partial mutual histogram for each thread block, all thread blocks would update a single histogram in the global memory. Atomic update operations generally exhibit longer latency. The latest GPU architecture has attempted to resolve this performance bottleneck by introducing true cache hierarchy. Significant speedup in atomic update operations is reported compared to the previous generation of the architecture. With proper interleaving of compute and memory access operations, the memory access latency can be hidden from the overall execution time.

Some applications have sacrificed registration accuracy for performance by either decreasing the number of bins of the mutual histograms to fit a smaller version of the partial mutual histograms in the shared memory, or decreasing the number of samples (image voxels) in order to decrease the number of mutual histogram

accesses to the global memory. These alternatives might seem applicable for some applications, but are not practical for MI-based nonrigid image registration and are not considered in this work as a high degree of accuracy is crucial for eventual clinical use.

Philip [50] proposed a bit-slicing solution, which subdivides the images by voxel intensity, and consequently subdivides the mutual histogram. With this approach, each thread block is assigned to reference image voxels of the same intensities; therefore, only one "slice" of the mutual histogram (256 bins in the case of 8-bit intensity images) is needed to be maintained in the shared memory. The bit-slicing solution addresses the limitation in shared memory and the time overhead in the mutual histogram combination step; however, it requires fast preprocessing and load balancing support. When the intensity distribution is unbalanced, multiple blocks would be required to process voxels of a single intensity and mutual histogram combination step still cannot be avoided.

These proposed solutions addressed some of the issues in the lengthy MI computation process but have not yet fully exploited the compute capability of the GPU, which is architecturally different from the traditional CPU or CPU-cluster architecture, and integrated that into their algorithms. The constraints in the number and the size of the thread blocks drafted in these solutions limited the scalability of these algorithms. As a result, only moderate acceleration was achieved with the growing trend of multicore architectures.

GPU Implementation of Rigid Registration

We have first explored the fundamental design considerations based on the basic rigid image registration problem. In rigid registration, the entire 3D image is transformed with the same transformation matrix and the computation of each element is totally independent from other elements. If unlimited number of threads and compute capacity are available, all voxels could potentially be processed concurrently. This observation leads to the following proposed solution.

Voxel-to-Thread Approach

To better utilize the GPU capacity for computation acceleration problem, maximization of parallelism and minimization of memory access are the two fundamental goals we want to achieve. One of the crucial decisions in achieving these goals is to determine the proper image subdivision and thread block allocation schemes to better balance the throughput and memory usage.

As discussed previously, designers typically approach this problem taking the global memory limitation of fitting the mutual histograms as priority. As a result, only small number of blocks is initialized with the maximum number of threads allowable to iterate through the assigned portion of the subdivided image. We approach this problem from the opposite direction by taking the benefit of the enhanced atomic update operations in the latest architecture. Instead of limiting

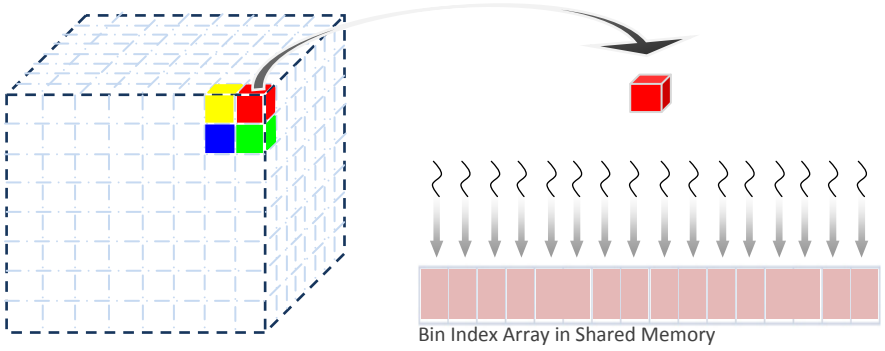

Figure 3.4: Voxel-to-Thread based thread block assignment scheme

ourselves by the memory usage of storing the mutual histograms, we maximize the parallelism of the algorithm by launching as many threads as the number of voxels in the image. Each thread is dedicated to process only one voxel in the reference image. Instead of maintaining a partial mutual histogram of any form in each block and having each thread to immediately update the result onto its local histogram, each thread will simply write the resulting bin index to an array (Figure 3.4). Therefore, the size of the thread blocks is solely determined by the number of registers and shared memory used by each thread for reading the image voxel intensities and storing the resulting bin indices. The size of the thread

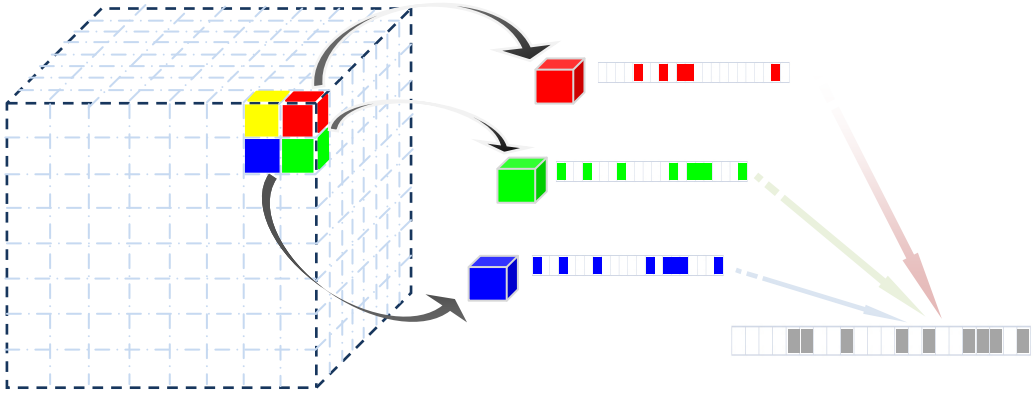

Figure 3.5: Voxel-to-Thread based blocks assignment scheme

blocks is equivalent to the size of the subimage blocks in this approach (Figure 3.5). Therefore, the larger the thread block size, the fewer is the total number of thread blocks.

Although it seems reasonable to define the thread block size simply based on the maximum technology limits *(1024 threads for Fermi)* and memory usage, the balance between the number of threads per block and the overall number of blocks should be considered. Larger block size might not result in optimal performance. We have examined the effect in performance with various block sizes ranging from 32 threads (one warp) to 1024 threads. For rigid registration with nearest-neighbor interpolation, specifying 128-threads per block results in optimal performance (Table 3.1). Thread block dimensions also slightly alter the overall performance. As GPU memory access is warp-based, better block dimension definition could lower the number of memory accesses. Table 3.2 shows four different block dimension definitions and the associated registration runtime. Blocks of 8 x 4 x 4 show slight performance gain in runtime.

Number of Threads	(sec) Case 1	Case 2	Case 3	Case 4	Case 5
32	6.66	5.36	6.56	6.94	8.37
64	4.97	3.99	4.90	5.17	6.22
128	4.83	3.86	4.75	5.02	6.01
256	4.93	3.95	4.86	5.13	6.15
512	5.06	4.05	4.98	5.26	6.32
1024	5.39	4.33	5.33	5.62	6.75

Table 3.1: Thread Block Size vs Registration Runtime (sec)

x	Y	z	(sec) Case 1	Case 2	Case 3	Case 4	Case 5
8	4	4	4.76	3.81	4.67	4.95	5.94
16	4	2	4.86	3.87	4.79	5.05	6.05
16	8	1	4.82	3.84	4.75	5.01	6
32	4	1	4.92	3.93	4.84	5.11	6.09

Table 3.2: Thread Block Dimension vs Registration Runtime (sec)

Warp Level Global Memory Access Reduction

The resulting bin indices from each thread block will then be updated to the mutual histogram residing in the global memory. If we were to follow the naïve approach and let every thread accumulate its bin index onto the global histogram, the number of memory accesses will be on the order of *O(n)*. Since the resulting bin index of each thread is totally random, and the threads access the global memory in warp base, the naïve approach will also cause memory update conflict (Figure 3.6). As a result, memory latency dominates the overall runtime and significantly impairs the performance.

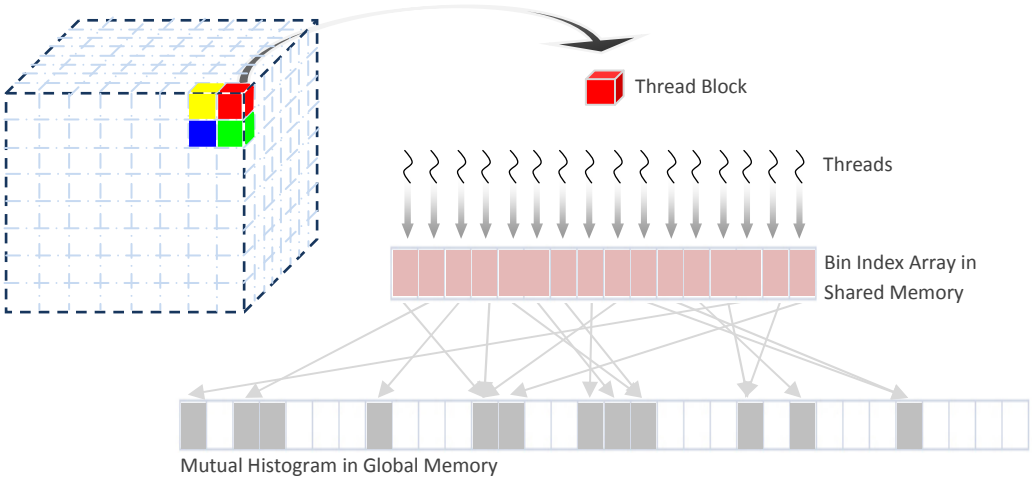

Figure 3.6: Thread blocks Update Mutual Histogram in Global Memory

Shams et. al [46] have proposed the *'sort and count'* algorithm to address this issue (Figure 3.7). This algorithm suggested sorting the elements of the resulting array of bin indices and further consolidating by counting the number of same appearances in the shared memory before updating the values to the global memory. Table 3.3 shows the execution time with and without applying the *sort and count* technique. The result shows a factor of 3.3x improvement with the *sort and count* technique.

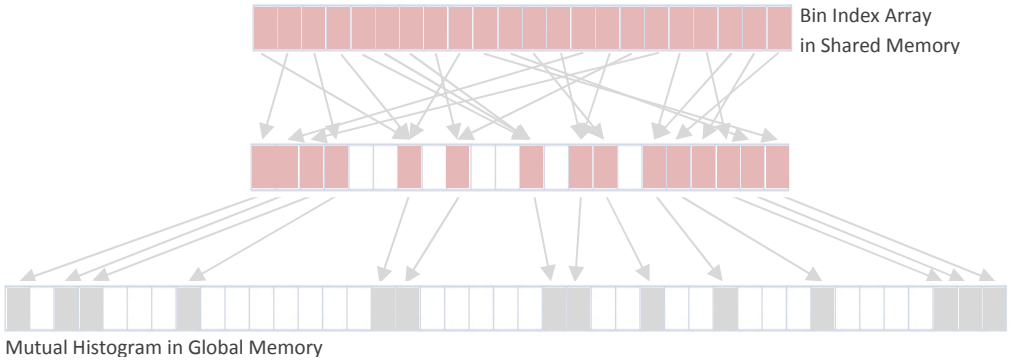

Figure 3.7: Thread blocks Update Mutual Histogram in Global Memory with the *Sort and Court* Technique

	(sec)					*(ms/MVoxel/iter)*
	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	***Normalized***
with S&C	21.71	16.84	21.37	22.29	26.93	*9.24736*
without S&C	6.49	5.24	6.4	6.78	8.18	*2.80607*

Table 3.3: Execution with and without 'sort and count' (sec), and average normalized result (ms/MegaVoxel/iter)

This approach significantly reduces the number of global memory accesses if the distribution of the mutual histogram is dense, which would have caused server update conflicts without reducing the number of accesses. However, in the case of sparse mutual histogram distribution, the *sort and count* algorithm will result in

very minimal improvement in performance. In addition, the runtime of the sort and count operation is not unnoticeable given the most optimal parallel sort algorithm (e.g., bitonic sort which runs in $O(log^2(n))$ time). The timing trade-off between the *sort and count* operation and global memory access latency has to be properly addressed.

Sort Group

In our implementation, we have considered the warp nature of the GPU architecture and suggested the concept of *sort group* (Figure 3.8). While GPU memory access is executed in warp based, we suggest applying warp based data reduction (as opposed to thread block based) and only consolidating the elements in the resulting array in a group size of 16 (half warp) or 32 (full warp). This approach guarantees that the runtime of the sorting operation is quick (16 as opposed to 1024, the maximum number of allowable threads per block) and at the same time the memory access latency is reduced by a significant magnitude.

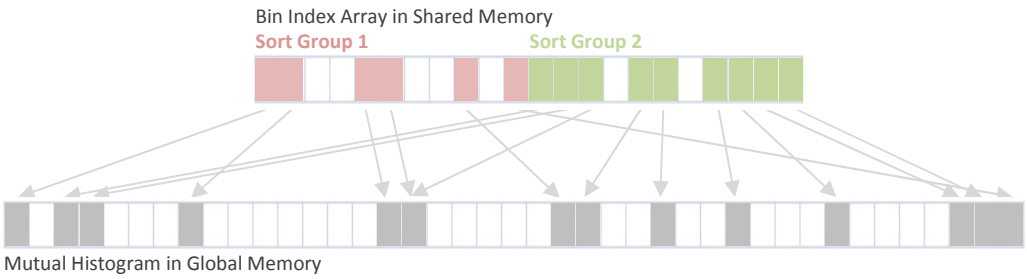

Figure 3.8: Thread blocks Update Mutual Histogram in Global Memory with Sort Groups

This hypothesis has been proven by the experimental result showed in Table 3.4 and Figure 3.9. In this experiment, a block size of 128 threads was defined. Without any consolidation preformed on the bin index array, rigid registration takes 16.84 to 26.93 seconds depending on the number of optimization iterations., normalized to *9.25ms/MVoxel/itr*. With small degree of consolidation, sort group size of 2, the runtime reduced by 40%. Sorting the entire thread block as the *'sort and count'* technique suggested, 128 threads in this case, is shown not to be optimal as the time taken by the sorting operation dominates and exceeds the memory access latency. With the group size of 16, the size of a half warp, the registration process exhibits the shortest runtime of *2.04542ms/MVoxel/itr*.

Sort group size	(sec) Case 1	Case 2	Case 3	Case 4	Case 5	*(ms/MVoxel/iter) Normalized*
1	21.71	16.84	21.37	22.29	26.93	*9.24736*
2	12.38	9.65	12.17	12.73	15.38	*5.28009*
4	7.86	6.18	7.74	8.1	9.77	*3.36103*
8	5.72	4.53	5.62	5.92	7.13	*2.4518*
16	**4.76**	**3.8**	**4.68**	**4.94**	**5.94**	***2.04542***
32	4.8	3.86	4.73	5.01	6.03	*2.07166*
64	5.13	4.15	5.06	5.35	6.46	*2.21785*
128	6.49	5.24	6.4	6.78	8.18	*2.80607*

Table 3.4: Sort and count experiment with different group size (sec), and average normalized result (ms/MegaVoxel/iter)

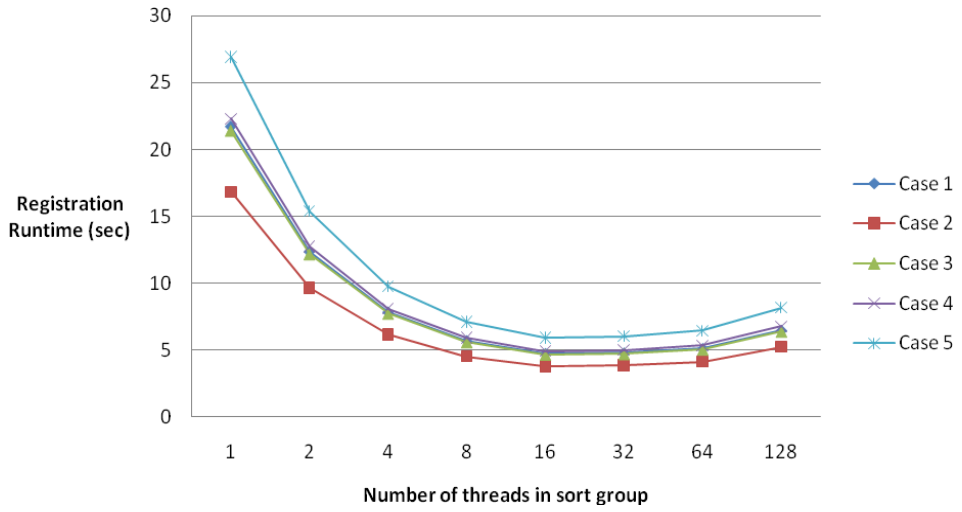

Figure 3.9: Sort and count experiment with different group size

Quick Sort and Merge

The sort and count technique combined with the concept of sort group significantly reduces global memory access conflict. With the proper selection of the sort group size, the runtime performance achieves a fact of 4.5x speedup. However, the performance improvement is scaled down almost linearly when it is applied to registration with partial volume interpolation.

In nearest-neighbor interpolation, every image voxel in the reference image voxel is mapped to one nearest-neighbor in the floating image which results at most one intensity value. However, in partial volume interpolation, each image voxel, which is mapped to eight nearest neighbors (for 3D image), contribute 8x times the number of entries to the histogram (Figure 3.10). As the size of the bin index array is 8 times larger than the NN interpolation implementation, the overall runtime scaled almost linearly. Table 3.5 shows the execution time for the same

set of case images registered with partial volume interpolation. Note that registration with different interpolation scheme will alter the convergence path during optimization which will result in different number of iterations and total runtime.

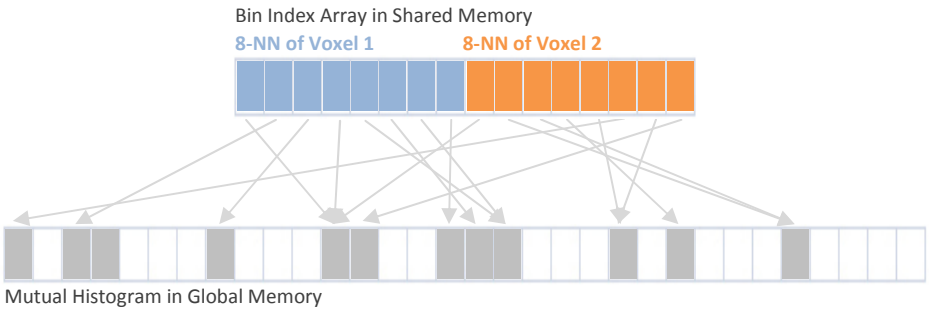

Figure 3.10: Update PV-Rigid Registration Mutual Histogram entries in Global Memory with Data Consolidation

			(sec)			*(ms/MVoxel/iter)*
Interpolation	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**	***Normalized***
NN	4.76	3.8	4.68	4.94	5.94	2.04542
PV	29.28	36.39	21.37	26.99	25.59	10.58965

Table 3.5: Registration Runtime with NN and PV interpolation (sec), and average normalized result (ms/MegaVoxel/iter)

Here we propose a more efficient solution to consolidate the data in the shared memory based on the distribution of the data generated in partial volume interpolation. As mentioned before, each image voxel in the reference will map to eight voxels in the floating image translated to eight entries to the mutual histogram. Since these eight entries are generated from the same reference image voxel, by construction, they belong to the same 256-bin sub-histogram and can be easily consolidated before being sorted with other entries in the bin index array. A quick serial sorting algorithm can be used to efficiently sort eight entries in the

registers before being written to the shared memory. Figure 3.11 illustrates the idea pictorially. Table 3.6 shows the execution time improvement compared with the shared memory level sort and count technique.

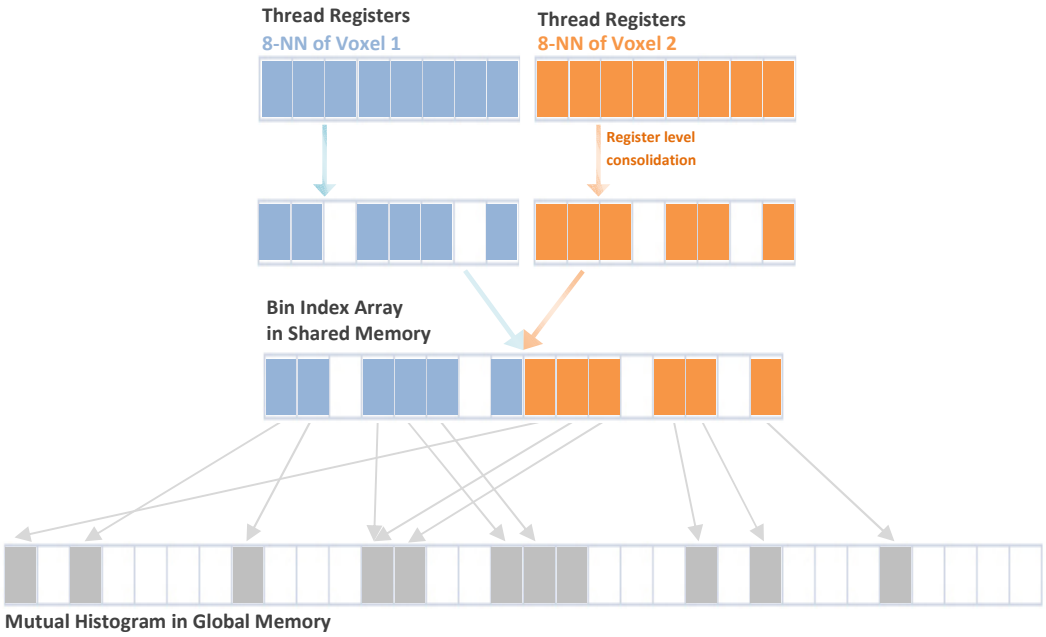

Figure 3.11: Update PV-Rigid Registration Mutual Histogram entries in Global Memory with Register-level Data Consolidation

	(sec)					*(ms/MVoxel/iter)*
	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	***Normalized***
S&C	29.28	36.39	21.37	26.99	25.59	10.58965
Reg-level Sort	14.51	18.07	10.58	13.4	12.68	5.250762

Table 3.6: Comparison of Registration Runtime with *'Sort and Count'* and Register-level Sort Technique (sec), and average normalized result (ms/MegaVoxel/iter)

Multiple partial histograms

When the distribution of the mutual histogram is dense, threads from multiple blocks try to update the same group of elements in the histogram at the same time.

The overall performance would be impaired when the histogram is updated through atomic add. To further reduce the memory delay caused by atomic update in specific, multiple partial mutual histograms are allocated in the global memory space to distribute the burden in memory bandwidth. The blocks are randomly assigned to one of the partial mutual histogram. The final mutual histogram is produced at the end by combining these partial mutual histograms using parallel reduction algorithm. Table 3.7 shows the impact on resulting NN-based registration runtime with allocating different number of partial histograms. As the size of the dataset increases in the PV-based registration, allocating multiple partial mutual histograms shows significant improvement of about 35% in the best case. Table 3.8 shows that using 8 partial mutual histograms presents a desirable balance between the performance gain and memory usage.

Number of Partial Histogram	(sec) Case 1	Case 2	Case 3	Case 4	Case 5	*(ms/MVoxel/iter) Normalized*
1	4.76	3.82	4.68	4.94	5.94	*2.04757*
2	4.4	3.57	4.37	4.62	5.58	*1.91129*
4	4.35	3.51	4.29	4.54	5.48	*1.88004*
8	4.32	3.49	4.26	4.52	5.44	*1.86828*
16	4.32	3.5	4.27	4.52	5.44	*1.87023*
32	4.37	3.52	4.31	4.57	5.49	*1.88777*
64	4.44	3.6	4.4	4.65	5.61	*1.92514*

Table 3.7: Number of Partial histograms vs. Runtime of Rigid Registration with NN-interpolation (sec), and average normalized result (ms/MegaVoxel/iter)

Number of Partial Histogram	(sec) Case 1	Case 2	Case 3	Case 4	Case 5	*(ms/MVoxel/iter) Normalized*
1	24.77	31.18	18.08	22.89	21.6	*8.982152*
2	16.98	21.26	12.39	15.69	14.85	*6.153763*
4	14.78	18.43	10.78	13.66	12.93	*5.352159*
8	14.51	18.07	10.58	13.4	12.68	*5.250762*
16	14.48	18.01	10.54	13.36	12.64	*5.234684*
32	14.57	18.1	10.6	13.43	12.71	*5.263665*
64	14.37	17.87	10.46	13.25	12.56	*5.195398*

Table 3.8: Number of Partial histograms vs. Runtime of Rigid Registration with PV-interpolation (sec), and average normalized result (ms/MegaVoxel/iter)

Discussion

In this chapter, we have discussed several design considerations to efficiently translate the compute intensive image registration problem onto the GPU architecture. Our implementation maps the image voxel space to the GPU thread block space to maximize the number of threads and thread blocks. With a large number of blocks in GPU implementation, part of the memory access latency would be hidden from the compute operation resulting better overall performance.

We have compared the GPU-based rigid registration implementation with the CPU and the FPGA implementations. Table 3.9 shows the registration timing performance of the five CT-CT test cases on three different platforms. The GPU implementation outperforms the CPU implementation and the FPGA implementation by about 30 times and 3 times respectively. The GPU solution converged to the exact result as the CPU implementation as the GPU used double-precision arithmetic. The FPGA implementation, however, uses a pseudo-double precision solution and 7-bit image intensity due to hardware limitation, the

resulting registered images are slightly different from those of the other two implementations.

Platform	Case 1	Case 2	(sec) Case 3	Case 4	Case 5	*(ms/MVoxel/iter) Normalized*
CPU	416.73	519.17	303.21	381.72	362.43	*150.36*
FPGA	45.93	54.54	55.03	38.74	45.37	*16.6338*
GPU	14.51	18.07	10.58	13.4	12.68	*5.25076*

Table 3.9: Rigid Registration Timing Result with CPU, FPGA, and GPU (sec), and average normalized result (ms/MegaVoxel/iter)

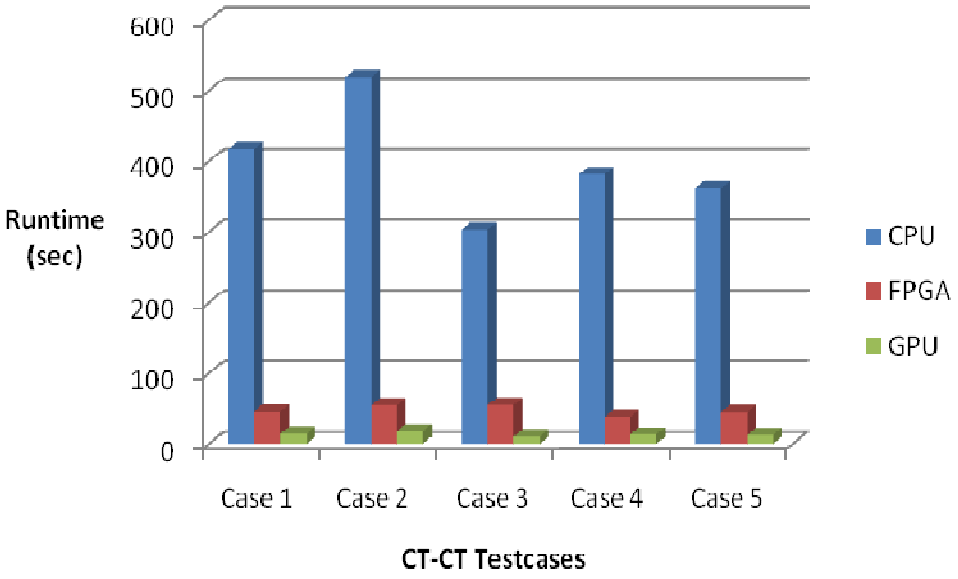

Figure 3.12: Rigid Registration Timing Result with CPU, FPGA, and GPU (sec)

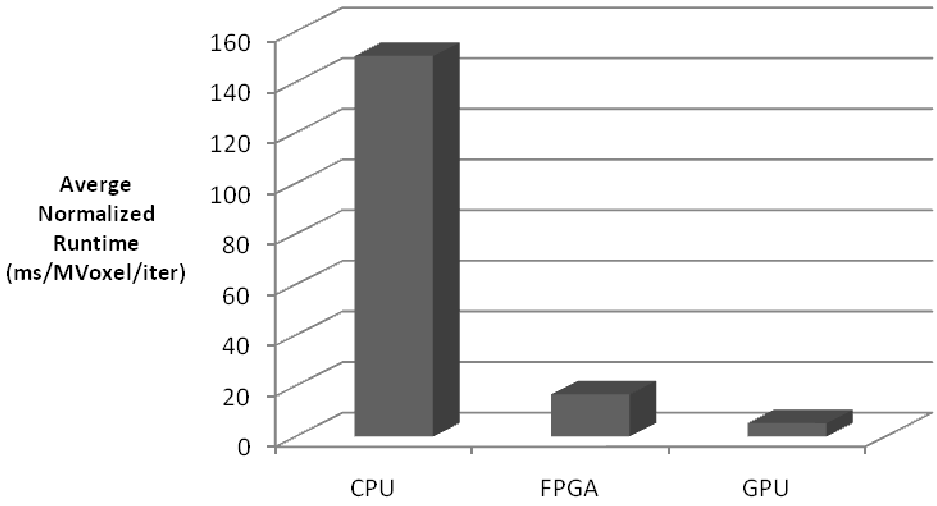

Figure 3.13: Average Normalized Rigid Registration Timing Result (ms/MVoxel/iter)

The registration accuracy is measured based on the RMS of the resulting deformation fields from different implementations and the reference. Table 3.10 shows the rigid registration result of the five test sets with three different implementations. The GPU implementation matches the accuracy as the CPU implementation. As we can see from the accuracy result, rigid registration alone is not sufficient to recover local deformation which leans our discussion to GPU-based nonrigid registration implementation.

Platform	Case 1	Case 2	(Voxel) Case 3	Case 4	Case 5
CPU	4.911486	4.180068	3.478706	5.148131	3.761118
FPGA	4.450058	4.313598	3.231499	4.220689	3.713284
GPU	4.911486	4.180068	3.478706	5.148131	3.761118

Table 3.10: Rigid Registration Accuracy Result with CPU, FPGA, and GPU (Voxel)

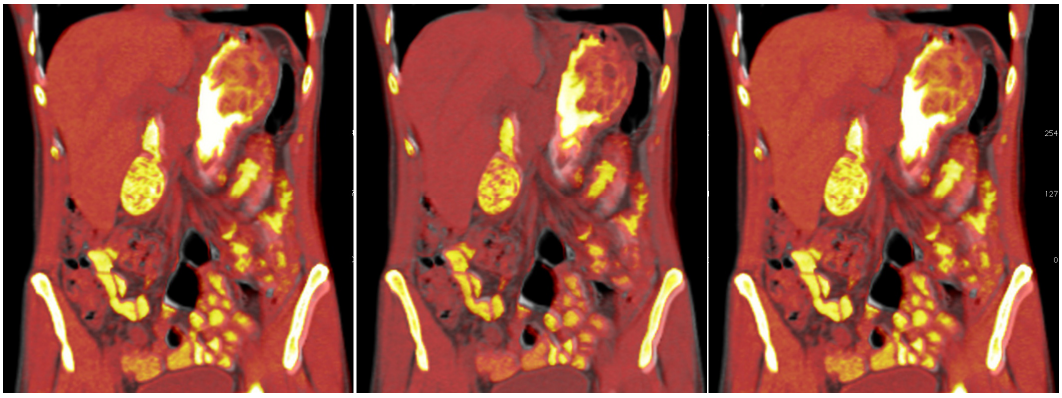

Figure 3.14: Case 1 Rigid Registration Result with Three Platforms
CPU (left), FPGA (center), GPU (right)

Chapter 4: Parallelization of Hierarchical Volume Subdivision-based Registration Algorithm on GPU

This chapter further extends our discussion on GPU-based rigid image registration implementation to nonrigid registration application and explores the parallelizable nature of a selected nonrigid registration algorithm. Our discussion will focus on the hierarchical volume subdivision-based registration algorithm [4] which has been proven to be accurate and structurally favorable for parallel implementation. First, we show how our rigid image registration implementation maps to the hierarchical volume subdivision algorithm. Then, we discuss the limitations encountered specifically in nonrigid registration and their associated solutions. Finally, we compare the performance of our GPU implementation with our single-processor implementation and other reported results.

Hierarchical Volume Subdivision-based Nonrigid Image Registration

While most nonrigid image registration algorithms exhibit fundamental limitations in parallelizability for efficient parallel implementation, the hierarchical volume subdivision-based registration algorithm presents a favorable framework for adopting the GPU implementation presented in the last session to the nonrigid registration applications.

Subvolume-based MI Optimization

Instead of computing the MI of the entire image, the volume subdivision-based algorithm subdivides the image in half in all three dimensions stepping down each hierarchical level, resulting eight times the number of subvolumes compared to

those at the upper level. All subvolumes of the same parent subvolume inherit the same global transformation from the upper level. Each subvolume then exercises its own independent iterative optimization path similar to the rigid registration problem. This means each subvolume performs transformation, computes MI, and updates the registration parameters independently from other subvolumes at the same hierarchical level.

GPU-based Implementation

We first approach this problem based upon the framework developed from the rigid registration implementation. We first perform rigid registration (Level 0) on the two input images. The reference image is then subdivided into a number of subvolumes and the registration result from Level 0 is pushed down to the next intermediate level immediately below it. Each subvolume will be processed one by one in the same fashion as in rigid registration with only one eighth of the total voxels being processed at a time (Figure 4.1).

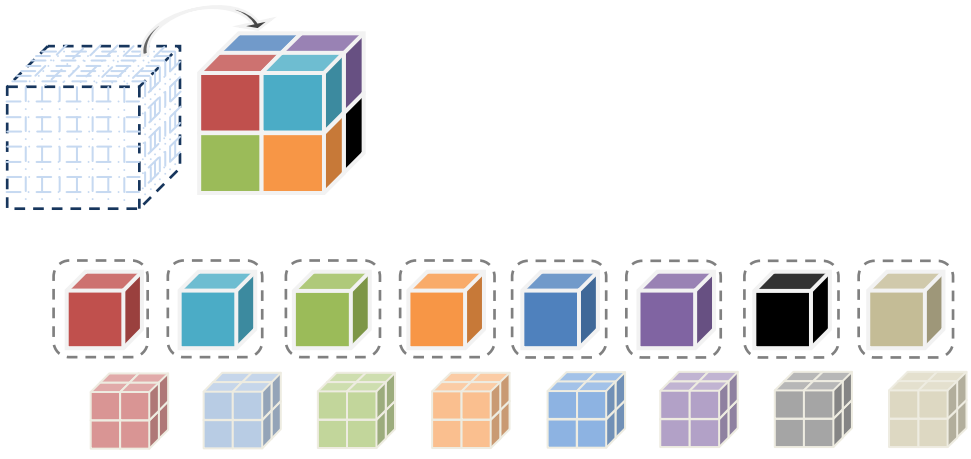

Figure 4.1: Hierarchical Volume Subdivision-based Nonrigid Registration

After all subvolumes in the current hierarchical level have completed individual local rigid registration, the deformation field of the entire image is obtained through interpolation. If applicable, the image can be further subdivided into smaller subvolumes. The same scheme is applied to the lower level.

Image Blocks vs GPU Blocks

Similar to the solution for rigid registration, the image voxels are spatially mapped to the GPU thread blocks. The same Voxel-to-Thread approach for thread block allocation is taken in nonrigid registration. This presents a straightforward translation of the rigid implementation to the nonrigid problem. Figure 4.2 shows the pseudo code of this nonrigid image registration implementation.

1. Global rigid image registration (as discussed in previous chapter)
2. Hierarchical volume subdivision-based algorithm begins
    2.1. While (subvolume dimension / 2) > (smallest allowable dimension)
        2.1.1. Perform octree-based subdivision on the reference image
        2.1.2. Compile initial mutual histogram *(MH_{Total})* of the entire image with the transformation model obtained from the previous hierarchical level
        2.1.3. For each subvolume
            2.1.3.1. Compile the mutual histogram of the subvolume *($MH_{subvolume}$)* with the inherited transformation model
            2.1.3.2. Compute MH_{Rest} by subtracting MH_{Total} by $MH_{subvolume}$
            2.1.3.3. Define an initial transformation model and set optimization termination condition to false
            2.1.3.4. While MI maximization termination condition is false
                2.1.3.4.1. Compile the mutual histogram of the subvolume $MH_{subvolume}$ with the revised transformation model
                2.1.3.4.2. Compute the *MH* by adding $MH_{Rest\ and}$ $MH_{subvolume}$
                2.1.3.4.3. Compute MI from *MH*
                2.1.3.4.4. Determine optimization algorithm termination condition based on the resulting MI
                2.1.3.4.5. Update transformation model

Figure 4.2: Pseudo code of Nonrigid Image Registration

The five CT-CT testcases are again used to benchmark the nonrigid implementation. Given the smallest allowable subvolume dimension to be 16 x 16 x 16, the hierarchical volume subdivision algorithm registered the images through four nonrigid registration levels (Level 1-4) plus the initial rigid registration (Level 0). Table 4.1 and 4.2 show the execution time and relative speedup of each level compared with the result generated by the CPU implementation and Table 4.3 shows the final nonrigid registration accuracy result. This implementation shows a factor of 6x speedup.

			(sec)		
CPU	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**
L0	414.66	517.71	303	381.54	362.41
L1	538.96	539.6	535.19	536.95	539.1
L2	612.64	611.01	600.23	610.96	604.57
L3	1295.01	1301.99	1288.84	1295.86	1292.18
L4	6560.45	6593.54	6572.82	6577.72	6576.02
Total	9421.72	9563.85	9300.08	9403.03	9374.28

			(sec)		
GPU	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**
L0	14.51	18.07	10.58	13.4	12.68
L1	33.14	33.21	32.81	33.05	33.13
L2	53.81	53.51	52.16	53.64	52.59
L3	190.51	190.11	192.15	189.73	188.69
L4	1270.18	1269.56	1269.9	1271.11	1268.1
Total	1562.15	1564.46	1557.6	1560.93	1555.19

Table 4.1: Nonrigid Registration Runtime (sec) CPU (top), GPU (bottom)

	Case 1	Case 2	Case 3	Case 4	Case 5
L0	28.57753	28.65025	28.63894	28.47313	28.58123
L1	16.26313	16.24812	16.3118	16.2466	16.27226
L2	11.38524	11.41861	11.50748	11.39001	11.49591
L3	6.797596	6.848614	6.707468	6.830022	6.848164
L4	5.164977	5.193563	5.175856	5.174784	5.185727
Total	6.031252	6.113196	5.970776	6.023992	6.027739

Table 4.2: GPU-based Nonrigid Registration Execution Time Speedup Compared to CPU Implementation

Platform	Case 1	Case 2	(Voxel) Case 3	Case 4	Case 5
CPU	1.256712	0.86262	0.682862	1.506372	0.781162
GPU	1.264021	0.866931	0.696836	1.506045	0.793361

Table 4.3: Nonrigid Registration Accuracy Result with CPU and GPU (Voxel)

Discussion

As the number of subvolumes goes up at the lower hierarchical levels, the number of GPU kernel calls goes up proportionately. Although the total number of processed voxels remains the same, the number of voxels being processed in each GPU kernel call (in each subvolume) decreases. As a result, the number of threads being exercised in each call decreases and eventually the GPU compute capacity is no longer fully utilized. As the memory access latency remains constant, even with the decreased number of possible update conflicts, the memory latency dominates the overall execution time. In addition, as the number of subvolumes goes up, the number of independent mutual histograms goes up. The time spent in calculating MI for each subvolume per iteration increases linearly as well.

As subvolumes in the same hierarchical level are independent from all other subvolumes, all subvolumes could be processed in parallel if compute and memory resources are sufficient. Efficiently scheduling compute operations and allocating memory resources will further improve the performance by a great magnitude.

Chapter 5: Optimal Solution

In this chapter, a fully optimized GPU-based hierarchical volume subdivision based nonrigid image registration solution is presented. Our discussion will focus on the bottlenecks previously identified and discuss the proposed solutions. Finally, we will present the optimal performance and accuracy achieved with this implementation.

Lessons Learnt

Previous result (Table 4.1) has shown that our GPU implementation provides a significant performance speedup in rigid and upper levels of nonrigid registration. As the number of subvolumes goes up, the speedup plateaus and eventually starts to exhibit linear growth (Table 4.2). Primarily, this problem is caused by the decrease in parallelism while the number of voxels in each subvolumes decreases. The MI computation time and the memory access latency remain constant for each subvolume while the mutual histogram accumulation time becomes increasingly unnoticeable. If this trend is continued, the GPU implementation starts behaving as a single processor implementation.

Subvolume Group (svGrp)

To achieve the highest efficiency in GPU implementations, it is better to have more threads and thread blocks running on the GPU to better utilize the compute resource so as to hide the memory access latency. Based on the hierarchical volume subdivision algorithm framework, each voxel determines the

corresponding floating image voxel and the resulting bin index independently from all other elements in the image in both rigid and nonrigid levels. There is no data dependency among subvolumes thus all subvolumes could be processed simultaneously if compute and memory resources are sufficient. With one dedicated thread per voxel, the full image can be processed by the GPU in parallel without separate kernel call for each subvolume. The critical implementation element is to ensure that each voxel gets the correct transformation matrix associated with its subvolume and updates the corresponding mutual histogram accordingly.

This modification in the implementation once again raises concern in the global memory limitation. All voxels in all subvolumes are essentially being processed at the same time (in the same kernel call). Therefore, multiple groups of mutual histograms are allocated in the global memory: at least one mutual histogram per subvolume but, ideally, multiple mutual histograms for each subvolume to avoid update conflict. The global memory size now limits the number of mutual histograms allocated, thus constrains the number of possible subvolumes being processed in parallel.

We have introduced subvolume group *(svGrp)* in our implementation which defines the number of subvolumes being processed in parallel for each hierarchical level (Figure 5.1). Figure 5.2 shows the pseudo code of this implementation.

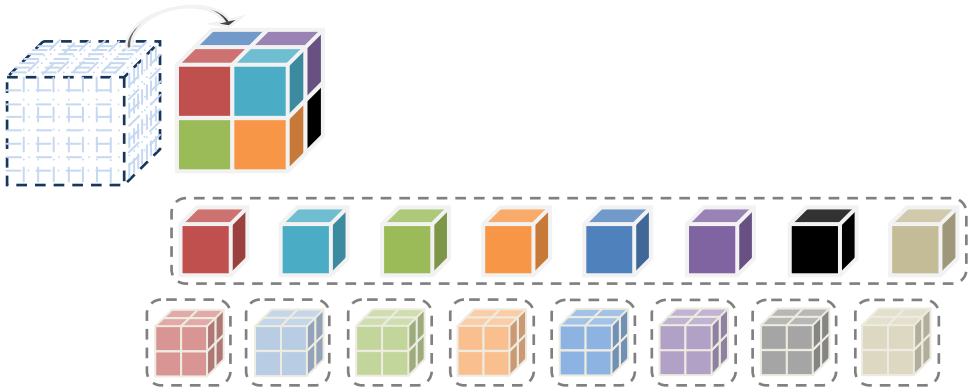

Figure 5.1: Hierarchical Volume Subdivision-based Nonrigid Registration with the Subvolume Groups of 8

1. Global rigid image registration (as discussed in previous chapter)
2. Hierarchical volume subdivision based algorithm begins
   - 2.1. While (subvolume dimension / 2) > (smallest allowable dimension)
     - 2.1.1. Perform octree-based subdivision on the reference image
     - 2.1.2. Compile initial mutual histogram *(MH_{Total})* of the entire image with the transformation model obtained from the previous hierarchical level
     - 2.1.3. For each subvolume group *(svGrp)*
       - 2.1.3.1. Compile the mutual histograms of the subvolumes within the *svGrp* *($MH_{subvolume}$)* with the inherited transformation model
       - 2.1.3.2. Compute MH_{Rest}(s)by subtracting MH_{Total} by $MH_{subvolume}$(s)
       - 2.1.3.3. Define an initial transformation model and set optimization termination condition to false for each subvolumn in the *svGrp*
       - 2.1.3.4. While MI maximization termination condition is false
         - 2.1.3.4.1. Compile the mutual histogram of the each subvolume in the *svGrp* $MH_{subvolume}$ with the revised transformation model
         - 2.1.3.4.2. Compute the *MH*(s) by adding MH_{Rest}(s) *and* $MH_{subvolume}$(s)
         - 2.1.3.4.3. Compute MI(s) from *MH*(s)
         - 2.1.3.4.4. Determine optimization algorithm termination condition for each subvolume based on the corresponding MI(s)
         - 2.1.3.4.5. Independently Update transformation models for the subvolumes

Figure 5.2: Pseudo code of GPU-based Nonrigid Registration with the Concept of Subvolume Groups.

Table 5.1 shows the registration timing results of Case 1 with subvolume group size of 8, 64, and 512 subvolumes per subvolume group. The goal is to process as many subvolumes as possible on each hierarchical level. The subvolume group size under tested ranges from 8 to 512, the number of subvolumes in Level 1 to Level 3. The maximum group size of 512 is limited by the size of global memory. The result below shows the degree of performance improvement with higher parallelization across subvolumes.

	(sec)				
Subvolume Group	**L0**	**L1**	**L2**	**L3**	**L4**
8	14.52	25.48	26.23	42.94	194.15
64	14.51	25.48	24.92	31.8	104.8
512	14.51	25.47	24.93	29.93	90.62

Table 5.1: Subvolume Group Size vs Registration Runtime (sec)

Partial Histogram per subVolume

As the number of the voxels per subvolume decreases going down the subdivision hierarchy, the probability of memory update conflict on the mutual histogram goes down. The concept of allocating multiple mutual histograms in the global memory space is not as essential for the lower hierarchical level. The time spent in consolidating partial histograms can be reduced. As more global memory resource is freed up, more subvolumes can be processed in parallel. Table 5.2 shows the experimental result on how the number of partial histograms affects the total runtime for each hierarchical level. The subvolume group size is set to 8 across all nonrigid hierarchical levels (L1-L4). For the upper hierarchical level

(L1), utilizing eight partial mutual histograms shows the best performance while using use one partial mutual histogram in the lowest level (L4) performs better. Based on the result in Table 5.2, using 8-8-4-2-1 partial mutual histograms for L0-L4 shows the best execution time performance.

# of Partial Histograms	L0	L1	L2	L3	L4	Total
1	14.52	25.48	26.23	42.94	194.07	304.66
2	14.52	21.02	23.16	42.51	199.02	301.64
4	14.52	19.37	21.78	42.82	210.9	310.79
8	14.51	18.93	21.86	46.29	237.87	340.86

Table 5.2: Number of Partial Histograms vs Registration Runtime (sec)

Optimal Solution

To summarize the results from the previous discussion, we present here an optimal GPU/MI-based nonrigid image registration implementation. The hierarchical volume subdivision algorithm is efficiently ported to the GPU architecture to maximize the parallelized capacity. Various implementation aspects have been explored and the optimized solutions were proposed and tailored for each hierarchical level. A thread block size of 8 x 4 x 4 is defined for all hierarchical levels; register-based data consolidated is applied, subvolume group size of 512 is maximized to global memory capacity, different numbers of partial mutual histograms are allocated for different hierarchical levels as suggested in Table 5.2.

Table 5.3 shows the total registration runtime of the five test cases along with the timing breakdown for each hierarchical level. Table 5.4 shows the improvement

in registration accuracy of nonrigid registration compared with rigid registration. The registered images of Case 1 are presented in Figure 5.3. The left shows the axial/coronal fusion images and the right shows the axial/coronal different (subtracted) images between the reference image and the registered image. The registration error is nearly visually unnoticeable. Figure 5.4 shows effect in registration by the fusion and subtracted images before registration, after rigid registration and after nonrigid registration.

	Case 1	Case 2	Case 3	Case 4	Case 5	*Normalized*
L0	14.51	18.07	10.58	13.4	12.68	*5.25076*
L1	18.94	18.93	18.89	18.92	18.89	*5.63681*
L2	20.54	20.54	20.48	20.51	20.49	*6.11305*
L3	29.94	29.95	29.99	29.79	29.73	*8.90493*
L4	90.62	90.78	91.05	90.73	90.79	*27.0587*
Total	175.98	179.68	172.4	174.76	173.99	*52.96428*

Table 5.3: Optimal Nonrigid Image Registration Timing Result (sec), and average normalized result (ms/MegaVoxel/iter)

	Case 1	Case 2	Case 3	Case 4	Case 5
Rigid	4.911486	4.180068	3.478706	5.148131	3.761118
Nonrigid	1.264021	0.866931	0.696836	1.506045	0.793361

Table 5.4: Optimal Rigid and Nonrigid Image Registration Accuracy Result (voxel)

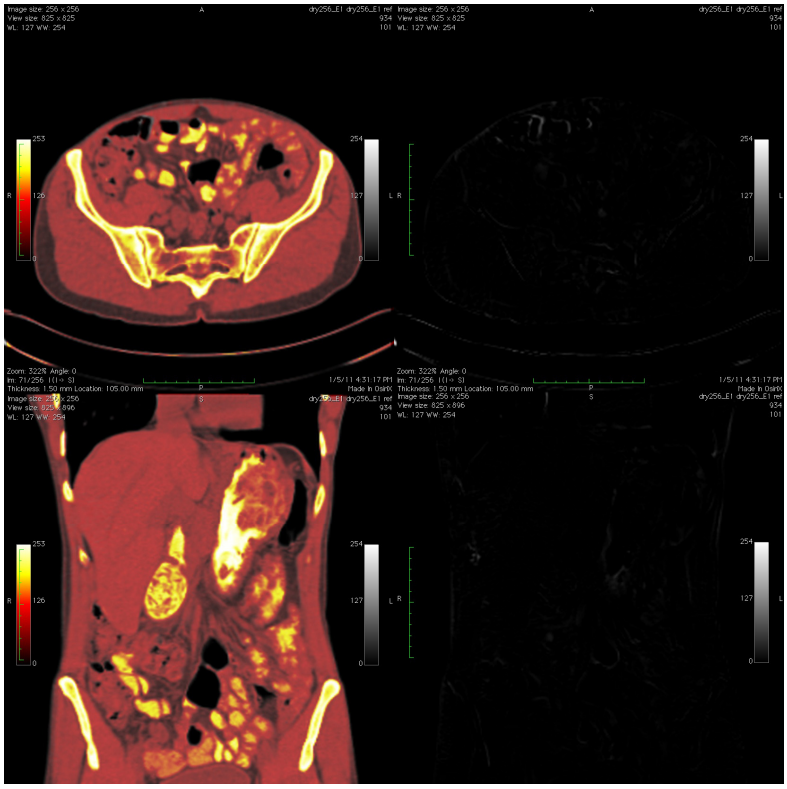

Figure 5.3: Optimal Nonrigid Image Registration Result of Case 1
Fusion Image (left), Different Image (right)

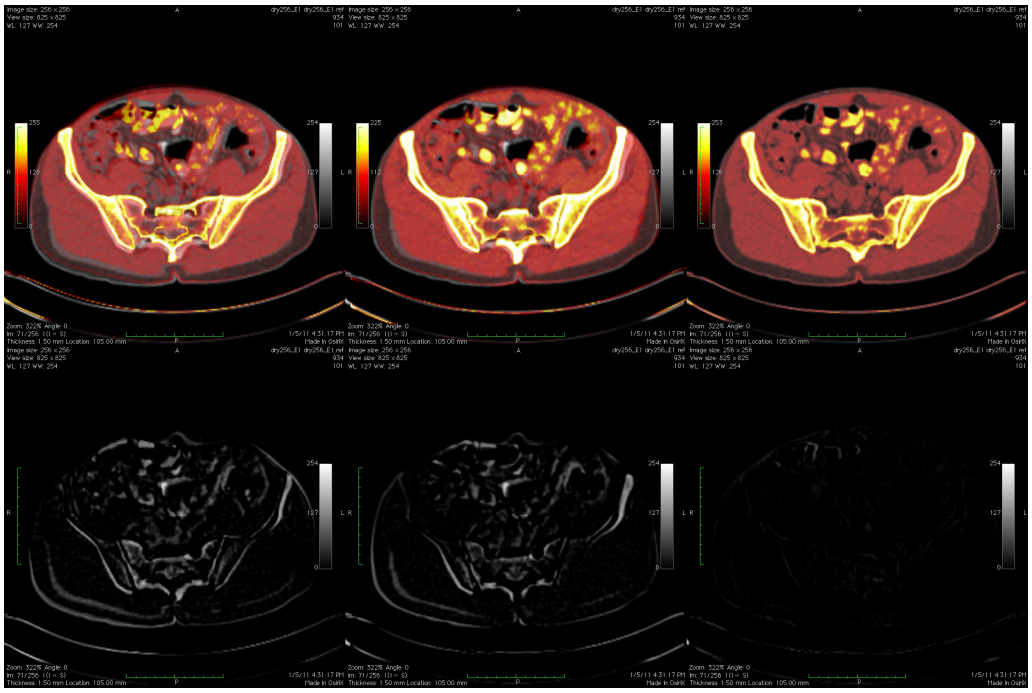

Figure 5.4: GPU-based Nonrigid Image Registration Result of Case 1
with No-Registration, Rigid Registration, Nonrigid Registration (Left-to-Right)

Chapter 6: Conclusions and Future Work

Automatic intensity based nonrigid rigid registration has found wide usage in medical image processing applications. One of the most accurate, reliable and fully automatic approaches to 3D image registration is maximization of mutual information (MI) between two images. However, long execution time continues to limit MI-based registration to be practical.

Driven by the 3D graphics market demand, modern GPUs have evolved into a highly parallel, multithreaded, multicore processor with tremendous computational horsepower and high memory bandwidth. The increased flexibility of the most recent generation of GPU hardware and programming model has unlocked the computational power of the GPU and made accessible to numerically intensive general purpose applications. With effective utilization of the GPUs compute capacity, numerically intensive applications will achieve significant performance gains. Therefore, development of efficient data-parallel algorithms and implementation is crucial for performance improvement.

Adaptation of MI-based image registration onto GPU architecture is an interesting multivariable problem. We have presented an efficient parallel implementation for nonrigid registration based on the parallelizable hierarchical volume subdivision based algorithm. Several optimization techniques are discussed; including block size optimization for hiding memory access latency, warp based data reduction to reduce memory access conflict, and the use of multiple partial histograms to

reduce memory conflict. To achieve the optimal performance, the optimization techniques are applied with different parameters to different hierarchical level based on the number and size of the subvolumes.

The optimal GPU/MI-based image registration solution was tested with five CT-CT datasets. Accuracy and performance of this implementation were compared with a single-CPU and 3-FPGA implementations. A summary of the result of Case 1 is presented below (Table 6.1). Table 6.2 shows the nonrigid registration accuracy of the three implementations. Results of the CPU and the GPU implementations match closely as the result of the FPGA implementation differs slightly due to the difference in input image intensity accuracy (Figure 6.1).

	(sec)						*(ms/MVoxel/iter)*
Case 1	**L0**	**L1**	**L2**	**L3**	**L4**	**Total**	***Normalized***
CPU	414.66	538.96	612.64	1295.01	6560.45	9421.72	*2834.853*
3FPGA	45.936	21.972	20.964	29.248	102.146	220.266	*68.54616*
GPU	14.51	18.94	20.54	29.94	90.62	174.55	*52.93723*

Table 6.1: Nonrigid Registration Timing Result of Case 1 with CPU, 3-FPGA, and GPU Implementations (sec), and average normalized result (ms/MegaVoxel/iter)

	(Voxel)				
Platform	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**
CPU	1.256712	0.86262	0.682862	1.506372	0.781162
FPGA	1.208796	1.189339	0.91128	1.482676	0.998999
GPU	1.264021	0.866931	0.696836	1.506045	0.793361

Table 6.2: Nonrigid Registration Accuracy Result with CPU, 3-FPGA, and GPU Implementations (Voxel)

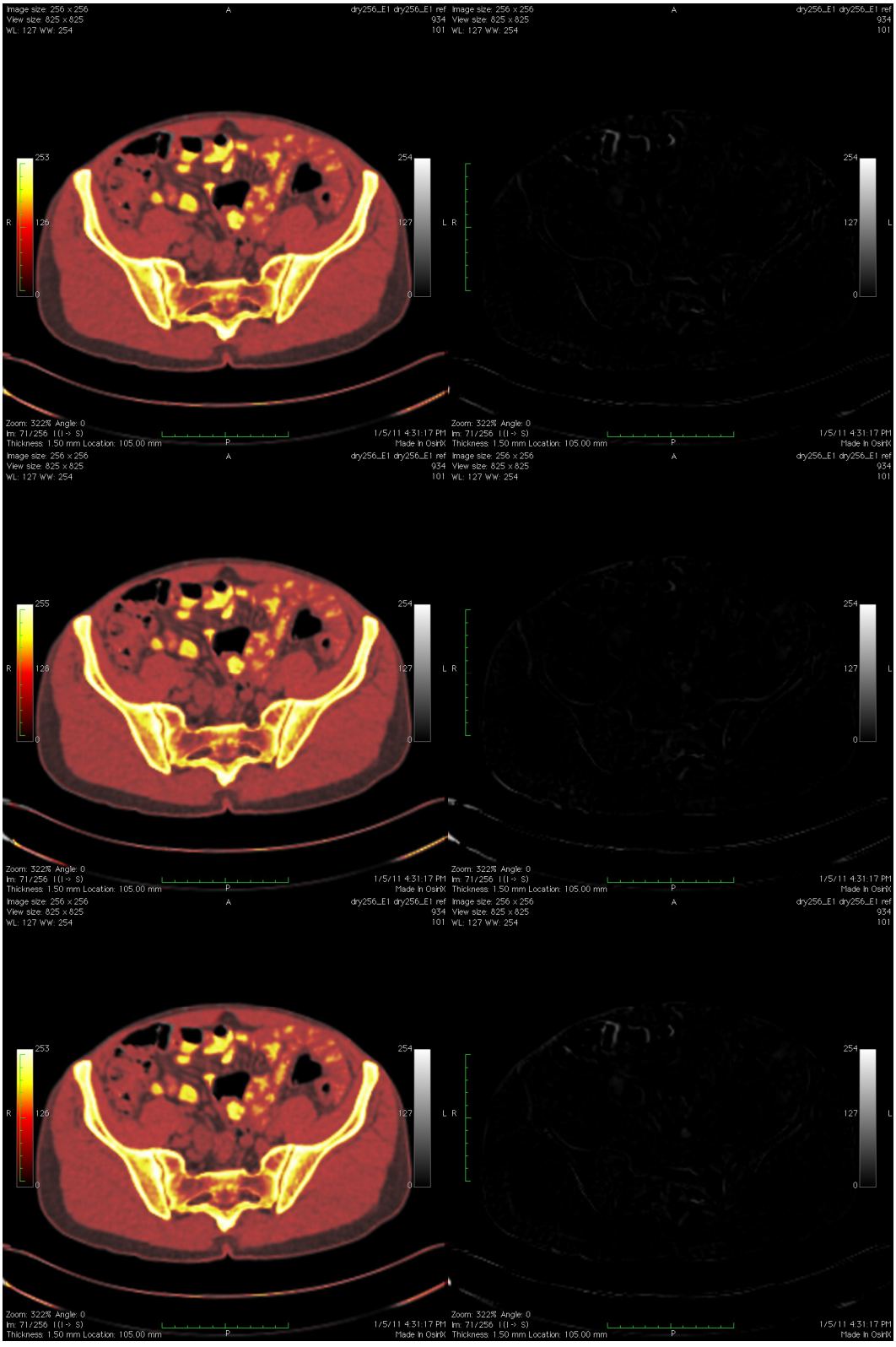

Figure 6.1: Nonrigid Registration Result with CPU, 3-FPGA, and GPU Implementations (Top-to-bottom)

Bibliography

[1] C. Meyer, J. L. Boes, B. Kim, P. H. Bland, R. L.Wahl, K. R. Zasadny,P. V. Kison, K. Koral, and K. A. Frey, "Demonstration of accuracy and clinical versatility of mutual information for automatic multimodality image fusion using affine and thin plate spline warped geometric deformations," Med. Image Anal., vol. 1, no. 3, pp. 195–206, 1997.

[2] D. Rueckert, L.I. Sonoda, C. Hayes, D.L.G. Hill, M.O. Leach, D.J. Hawkes, "Nonrigid registration using free-form deformations: application to breast MR images," IEEE Trans. Med. Imaging 18(8) (1999) 712–721.

[3] D. L. G. Hill, C. R. Maurer Jr., A. J. Martin, S. Sabanathan, W. A.Hall, D. J. Hawkes, D. Rueckert, and C. L. Truwit, "Assessment of intraoperative brain deformation using interventional MR imaging," in Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI'99), vol. 1679, Lecture Notes in Computer Science, C. Taylor and A. Colchester, Eds., Cambridge, U.K., Sept. 1999, pp. 910–919.

[4] V. Walimbe and R. Shekhar, "Automatic elastic image registration by interpolation of 3D rotations and translations from discrete rigid-body transformations," Medical Image Analysis, vol. 10(6), p. 899, 2006.

[5] JB Maintz, MA Viergever. "A survey of medical image registration. Medical Image Analysis." 1998;2(1):1-36.

[6] J. West, JM Fitzpatrick, MY Wang, et al. "Retrospective intermodality registration techniques for images of the head: Surface-based versus volume-based." IEEE Transactions on Medical Imaging. 1999;18(2):144-150.

[7] WMI Wells, P. Viola, H. Atsumi, S. Nakajima, R. Kikinis. "Multi-modal volume registration by maximization of mutual information. Medical Image Analysis." 1996;1(1):35-51.

[8] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, P. Suetens. "Multimodality image registration by maximization of mutual information." IEEE Transactions on Medical Imaging. 1997;16(2):187-198.

[9] C. Studholme, D. Hill, D. Hawkes. "Overlap invariant entropy measure of 3D medical image alignment." Pattern Recognition. 1999;32(1):71-86.

[10] A. Carrillo, J. Duerk, J. Lewin, D. Wilson. "Semiautomatic 3-D image registration as applied to interventional MRI liver cancer treatment." IEEE Transactions on Medical Imaging. 2000;19(3):175-185.

[11] C. Studholme, D. Hill, D. Hawkes. “Automated three-dimensional registration of magnetic resonance and positron emission tomography brain images by multiresolution optimization of voxel similarity measures.” Medical Physics. 1997;24(1):25-35.

[12] J. Pluim, J. Maintz, M. Viergever. “Mutual-information-based registration of medical images: a survey.” IEEE Transactions on Medical Imaging. 2003;22(8):986-1004.

[13] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," Medical Image Analysis, vol. 2(1), pp. 1-36, 1998.

[14] D. L. G. Hill, "Medical image registration," Physics in Medicine & Biology, vol. 46(1), p. 1, 2001.

[15] J. V. Hajnal, D. J. Hawkes, and D. L. G. Hill, Medical image registration. Boca Raton: CRC Press, 2001.192

[16] J. F. Krucker, G. L. LeCarpentier, J. B. Fowlkes, and P. L. Carson, "Rapid elastic image registration for 3-D ultrasound," IEEE Transactions on Medical Imaging, vol.21(11), pp. 1384-1394, 2002.

[17] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P.Suetens, G. Marchal, Automated multimodality medical image registration using information theory, in: Proceedings of the International Conference Information Processing in Medical Imaging: Computational Imaging and Vision, vol. 3, April, 1995, pp. 263–274.

[18] P. Viola, W.M. Wells III, Alignment by maximization of mutual information, in: Proceedings of the International Conference on Computer Vision (ICCV), June, 1995, pp. 16–23.

[19] C. E. Shannon, “A mathematical theory of communication,” Bell Syst. Tech. J., vol. 27, p. 379–423/623–656, 1948.

[20] R. P. Woods, S. R. Cherry, and J. C. Mazziotta, “Rapid automated algorithm for aligning and reslicing PET images,” J. Comput. Assist. Tomogr., vol. 16, no. 4, pp. 620–633, 1992.

[21] R. P. Woods, J. C. Mazziotta, and S. R. Cherry, “MRI-PET registration with automated algorithm,” J. Comput. Assist. Tomogr., vol. 17, no. 4, pp. 536–546, 1993.

[22] D. L. G. Hill, D. J. Hawkes, N. A. Harrison, and C. F. Ruff, “A strategy for automated multimodality image registration incorporating anatomical knowledge

and imager characteristics," in Information Processing in Medical Imaging, H. H. Barrett and A. F. Gmitro, Eds. Berlin, Germany: Springer-Verlag, 1993, vol. 687, pp. 182–196. Lecture Notes in Computer Science.

[23] A. Collignon, D. Vandermeulen, P. Suetens, and G. Marchal, "3D multimodality medical image registration using feature space clustering," in Computer Vision, Virtual Reality, and Robotics in Medicine, N. Ayache, Ed. Berlin, Germany: Springer-Verlag, 1995, vol. 905, pp. 195–204. Lecture Notes in Computer Science.

[24] C. Studholme, D. L. G. Hill, and D. J. Hawkes, "Multiresolution voxel similarity measures for MR-PET registration," in Information Processing in Medical Imaging, Y. Bizais, C. Barillot, and R. Di Paola, Eds. Dordrecht, The Netherlands: Kluwer, 1995, pp. 287–298.

[25] A. Collignon, "Multi-modality medical image registration by maximization of mutual information," Ph.D. dissertation, Catholic Univ. Leuven, Leuven, Belgium, 1998.

[26] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, "Automated multi-modality image registration based on information theory," in Information Processing in Medical Imaging, Y. Bizais, C. Barillot, and R. Di Paola, Eds. Dordrecht, The Netherlands: Kluwer, 1995, pp. 263–274.

[27] P. Viola and W.M. Wells III, "Alignment by maximization of mutual information," in Proc. Int. Conf. Computer Vision, E. Grimson, S. Shafer, A. Blake, and K. Sugihara, Eds. Los Alamitos, CA, 1995, pp. 16–23.

[28] P. A. Viola, "Alignment by maximization of mutual information," Ph.D. dissertation, Massachusetts Inst. Technol., Boston, MA, 1995.

[29] W.M. Wells III, P. Viola, and R. Kikinis, "Multi-modal volume registration by maximization of mutual information," in Medical Robotics and Computer Assisted Surgery. New York: Wiley, 1995, pp. 55–62.

[30] C. Studholme, "Measures of 3D Medical Image Alignment," Ph.D. dissertation, Univ. London, London, U.K., 1997.

[31] M. Holden, D. L. G. Hill, E. R. E. Denton, J. M. Jarosz, T. C. Cox, T. Rohlfing, J. Goodey, and D. J. Hawkes, "Voxel similarity measures for 3-D serial MR brain image registration," IEEE Transactions on Medical Imaging, vol. 19(2), pp. 94-102, 2000.

[32] J. Tsao, “Interpolation artifacts in multimodality image registration based on maximization of mutual information.” IEEE Trans. Med. Imaging 22, 854–864. 2003

[33] W. H. Press, Numerical recipes in C++ : the art of scientific computing, 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2002.

[34] R. Shekhar and V. Zagrodsky, "Mutual information-based rigid and nonrigid registration of ultrasound volumes," IEEE Transactions on Medical Imaging, vol. 21(1), pp. 9-22, 2002.

[35] R. Shekhar, V. Walimbe, S. Raja, V. Zagrodsky, M. Kanvinde, G. Y. Wu, and B. Bybel, "Automated 3-dimensional elastic registration of whole-body PET and CT from separate or combined scanners," Journal of Nuclear Medicine, vol. 46(9), pp. 1488-1496, Sep 2005.

[36] V. Walimbe, V. Zagrodsky, S. Raja, B. Bybel, M. Kanvinde, and R. Shekhar, "Elastic registration of three-dimensional whole body CT and PET images by quaternion-based interpolation of multiple piecewise linear rigid-body registrations," in Proceedings of SPIE Medical Imaging, 2004, pp. 119-128.

[37] V. Elsen, P.A., Pol, E.J.D., Viergever, M.A., 1993. Medical image matching – a review with classification. IEEE Eng. Med. Biol. 12, 26–39.

[38] NVIDIA CUDA Programming Guide, ver 3.1, http://developer.download.nvidia.com/compute/cuda/3_1/NVIDIA_CUDA_Programming_Guide_3.1.pdf

[39] R. V. Nieuwpoort and J. W. Romein. “Using Many-Core Hardware to Correlate Radio Astronomy Signals, Proceedings of the ACM International Conference on Supercomputing (ICS’09), pp. 440-449, June 8-12, 2009, Yorktown Heights, New York, USA.

[40] B. G. Levine, J. E. Stone, and A. Kohlmeyer. “Fast Analysis of Molecular Dynamics Trajectories with Graphics Processing Units — Radial Distribution Function Histogramming” Journal Computational Physics, 2011. (Accepted)

[41] NVIDIA’s Next Generation CUDA Compute Architecture: Fermi Whitepaper, http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[42] W. Plishker, O. Dandekar, S. S. Bhattacharyya, and R. Shekhar, “Utilizing hierarchical multiprocessing for medical image registration,” IEEE Signal Processing Mag., vol. 27, no. 2, pp. 62–68, Mar. 2010.

[43] P. Muyan-Özçelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," in Proc. Int. Conf. Computational Science and Its Applications (ICCSA), 2008, pp. 5–8.

[44] Y. Lin and G. Medioni, "Mutual information computation and maximization using GPU," in Proc. IEEE Computer Vision and Pattern Recognition (CVPR) Workshops, June 2008, pp. 1–6.

[45] R. Shams and N. Barnes, "Speeding up mutual information computation using NVIDIA CUDA hardware," in Proc. Digital Image Computing: Techniques and Applications (DICTA), Adelaide, Australia, Dec. 2007, pp. 555–560.

[46] R. Shams, P. Sadeghi, R. A. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," Comput. Meth. Programs Biomed., to be published.

[47] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, "A Survey of Medical Image Registration on Multicore and the GPU." IEEE Signal Processing Magzine, March, 2000.

[48] C. Vetter, C. Guetter, C. Xu, and R. Westermann, "Non-rigid multi-modal registration on the GPU," in Proc. SPIE Medical Imaging: Image Processing, Feb. 2007, pp. 651228-1–651228-8.

[49] ATI stream computing user guide, version 1.4.0.a, ATI http://developer.amd.com/

[50] M. Philip. Parallelization of Non-rigid Image Registration. Master Thesis, Unvi. Of Maryland College Park, 2008