5-2017

Towards Real-time Remote Processing of Laparoscopic Video

Zahra Ronaghi
Clemson University, zronagh@clemson.edu

TOWARDS REAL-TIME REMOTE PROCESSING OF LAPAROSCOPIC VIDEO

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Bioengineering

---

by
Zahra Ronaghi
December 2016

Accepted by
Dr. David Kwartowitz, Committee Chair
Dr. Delphine Dean
Dr. J. Barr von Oehsen
Dr. S. Duke Herrell

ABSTRACT

Laparoscopic surgery is a minimally invasive technique where surgeons insert a small video camera into the patient's body to visualize internal organs and use small tools to perform these procedures. However, the benefit of small incisions has a disadvantage of limited visualization of subsurface tissues. Image-guided surgery (IGS) uses pre-operative and intra-operative images to map subsurface structures and can reduce the limitations of laparoscopic surgery. One particular laparoscopic system is the daVinci-si robotic surgical vision system. The video streams generate approximately 360 megabytes of data per second, demonstrating a trend toward increased data sizes in medicine, primarily due to higher-resolution video cameras and imaging equipment. Real-time processing this large stream of data on a bedside PC, single or dual node setup, may be challenging and a high-performance computing (HPC) environment is not typically available at the point of care. To process this data on remote HPC clusters at the typical 30 frames per second rate (fps), it is required that each 11.9 MB (1080p) video frame be processed by a server and returned within the time this frame is displayed or 1/30th of a second. The ability to acquire, process, and visualize data in real time is essential for the performance of complex tasks as well as minimizing risk to the patient.

We have implemented and compared performance of compression, segmentation and registration algorithms on Clemson's Palmetto supercomputer using dual Nvidia graphics processing units (GPUs) per node and compute unified device architecture (CUDA) programming model. We developed three separate applications that run

simultaneously: video acquisition, image processing, and video display. The image processing application allows several algorithms to run simultaneously on different cluster nodes and transfer images through message passing interface (MPI). Our segmentation and registration algorithms resulted in an acceleration factor of around 2 and 8 times respectively. To achieve a higher frame rate, we also resized images and reduced the overall processing time. As a result, using high-speed network to access computing clusters with GPUs to implement these algorithms in parallel will improve surgical procedures by providing real-time medical image processing and laparoscopic data.

DEDICATION

I dedicate this work to my family. My parents Majid and Manijeh, my sister Maryam and my brother Mohammad who have always provided me with their unconditional love, support and guidance. Words cannot express how much you all mean to me, thank you!

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor Dr. David Kwartowitz for his support, guidance and encouragement throughout my PhD. I will be forever grateful for the opportunity to pursue this degree, attend conferences and mentor undergraduate students.

Thank you Dr. Edward Duffy for your guidance, helpful discussions and invaluable assistance in different parts of my research. I would also like to thank my dissertation committee members, Dr. Delphine Dean, Dr. Barr von Oehsen, Dr. Duke Herrell and also Dr. Bruce Gao for their time, feedback and guidance.

Thank you Dr. Melissa Smith and Karan Sapra for providing your knowledge on high performance computing and Ryan Izard on computer networks. I am also thankful to Dr. Kuang-Ching Wang, Clemson Computing and Information Technology group: Mr. Jim Bottum, Ms. Kathryn Mace, Mr. Christopher A. Konger, Mr. Brian Parker, Mr. Joseph Bernard, and Mr. Randall Martin for their help with the network infrastructure and the financial support received from the National Science Foundation (NSF) grant ACI-1245936.

I appreciate the support of Dr. Martine LaBerge and all the members of the Department of Bioengineering, especially Ms. Maria Torres and Ms. Melissa McCullough, for always being available to answer questions. Finally, I would like to thank my family and all of my friends at Clemson University who have supported me during my time here. Thank you dad, for always guiding and encouraging me to pursue further knowledge in engineering and science and mom, for being a great inspiration and role model in pursing engineering.

TABLE OF CONTENTS

Table of Contents (Continued) Page

LIST OF FIGURES

Figure Page

List of Figures (Continued) Page

List of Figures (Continued) Page

Chapter 1

INTRODUCTION AND MOTIVATION

1.1 Significance and Innovation

Minimally invasive surgery (MIS) reduces patient trauma, hospital stay, and recovery time through reducing incision sizes to small “keyholes”. In laparoscopic procedures a camera is used for real-time visualization of the surgical field and guidance of the procedures. However, small field of view of the laparoscope as well as small incision size will result in a small overall visual field of the underlying tissues of interest. The reduced visual field along with organ motion due to natural and surgical affect can limit the surgeon’s ability to target specific subsurface locations with high accuracy and precision; therefore, requiring external information such as medical imaging to complement laparoscopic video [1,2].

Pre-operative and intra-operative images can be processed to offer more visual details of a surgical area of interest. Pre-operative images are necessary to understand the patient’s anatomy and plan a surgical process. Some of the most common pre-operative imaging methods include Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and Nuclear Medicine combined with CT or MRI. Intra-operative imaging can be used for real-time observation of the surgical field especially for applications that the position of internal organs is unpredictable. X-ray fluoroscopy, ultrasound and hand assist laparoscopy are techniques that have been used for intra-operative medical imaging [3, 4].

Radical nephrectomy, resection of entire kidney as well as surrounding fat, lymphatic and the adrenal gland, was the traditional treatment for patients. Recently, partial nephrectomy has become common. This involves partial resection of the kidney, removal of diseased segment of renal tissue and leaving normal functioning kidney tissue. Identifying clear margin can reduce procedure time and limit disturbance of healthy tissue, thus shorter recovery time for the patient [5]. Improving intraoperative visualization such as image-guided surgery (IGS) techniques during the procedure will improve surgeon's view of the margin and surgical outcome.

IGS uses pre-operative and intra-operative images to map a surgical region of interest (ROI), thus providing surgeons with visualization of subsurface structures and accurate positional information during a surgical procedure [6, 7]. The accurate co-registration of an IGS system with laparoscopic video can reduce the impact of limited surgical access and allow for a resection with higher specificity and tighter margins, thus sparing more healthy tissues. As a result, augmentation and mapping imaging data to the operative field of view can improve accuracy and efficiency of surgical procedures [8-11].

After the introduction of the Computer Motion AESOP and the Intuitive Surgical daVinci both in 1998, the use of robotics in medicine has continued to grow. Some of the clinical benefits of medical robotics are improvement in dexterity and accuracy of manipulation compared to traditional laparoscopy [11]. Currently the primary robotic surgery devices on the market are the daVinci-si and daVinci-xi systems. For this work we will look at the daVinci-si robot, with a HD vision cart, which uses two parallel 1080p (1920x1080x3) High Definition video cameras to generate stereoscopic vision. These two

views are then displayed using a pair of high definition monitors set up in a stereopticon-like device thus allowing the clinician the ability to perceive depth [12, 13]. If captured raw, these video streams generate approximately 360 megabytes of data per second.

Processing of the volume of data generated by the stereo video feeds would be demanding for a local system, and thus high-performance computing (HPC) hardware is needed [14-16], which can be both expensive (cost of power, cooling, system administration) and large in size. Even though the price of computing hardware continues to fall, the price of HPC hardware can limit systems to only hospitals with large financial resources, while physical size considerations can make hardware difficult to place in any operating room. As the size of data generated by surgery increases from enhanced video resolutions and new instrumentation in the operating room. The need to process and store this data is increasingly becoming a problem requiring efficient connectivity to HPC systems at remote locations.

I have developed image processing algorithms for augmentation of laparoscopic video with pre-operative and intra-operative medical images on HPC clusters. Computing clusters are often not available in hospital environments, and thus data will need to be transported from the point of care to be processed, and results will need to be returned in real-time. To realize the goal of rapid processing of laparoscopic data, we propose the development of a protocol where data is captured, transmitted, processed, returned, and displayed within $1/30^{th}$ of a second for the left and right eye synchronization of the daVinci. At the 30 fps rate of the video, each frame will take $1/30^{th}$ of a second (or 33ms). The two video streams will be synchronized if the overall time is less than one frame (33ms).

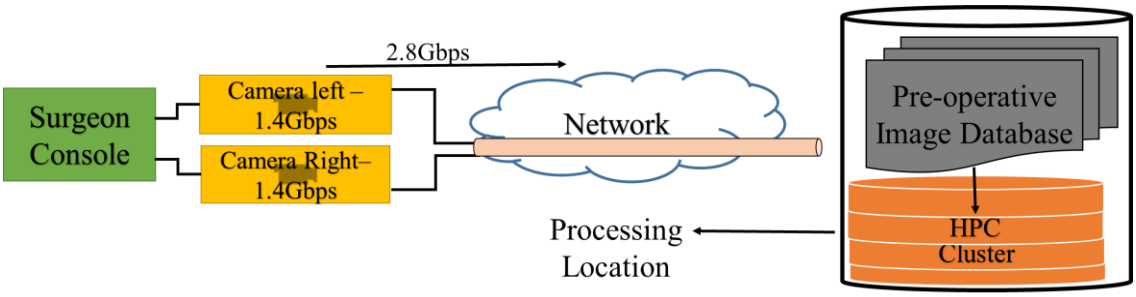


Figure 1.1 - System overview of laparoscopic cameras, network and cluster

Clemson University has recently deployed the OpenFlow software defined network (SDN) communication protocol on its network and is in the process of leveraging the new protocol and its connection to the Internet2 Innovation Platform [18]. SDN enables programmable network control for high-bandwidth and dynamic applications [17]. The new network will assist researchers in overcoming the network limitation to flexibly establish 10 gigabit or higher end-to-end connectivity, within a scientific demilitarized zone (DMZ), thus creating a frictionless connection between machines. DMZ ensures publicly accessible servers are placed on an isolated network segment [18]. As a result, using high-speed network to access computing clusters will lead to real-time or near real-time medical image processing with great flexibility.

The augmentation of live high definition stereo laparoscopic data with pre-operative or intra-operative medical images requires more computational capability than is possible on a single workstation. More advanced visualizations may require rendering techniques more suited to a high-performance computing environment. Therefore, flexibility in the transmission of data to multiple sites is important. Furthermore, the ability to optimize the data transmission to take advantage of the specialized processing and

rendering techniques at multiple endpoints is the key to allowing for rendering and processing as a service and also providing optimal data.

1.2 Motivation and Approach

A laparoscopic camera of interest is the vision system of daVinci-Si robotic surgical system. The vision system generates two parallel video streams one representing the left eye and one representing the right eye; totaling approximately 360 megabytes of data per second. The burden of processing this data on a bedside PC has become overwhelming and availability of a point of care high-performance computing (HPC) environment is rare. To process this data on remote HPC clusters at the typical 30 frames per second (fps) rate, each 11.9 MB video frame must be processed by a server and returned within 1/30th of a second. The ability to acquire, process and visualize data in real-time is essential for performance of complex tasks as well as minimizing risk to the patient. We hypothesize that by using high-speed networks to access computing clusters will lead to near real-time medical image processing and improve surgical outcomes by providing real-time enhanced laparoscopic data. I therefore propose the following goals:

Processing of the data generated by the stereo video feeds would be demanding for a local system, and thus high-performance computing (HPC) hardware is needed, which can be both expensive and large in size. Computing clusters are often not available in operating rooms (OR) or hospital environments, and thus data will need to be transported from the point of care to be processed, and results will need to be returned in real-time. For this work we will look at the daVinci-si robot, with a HD vision cart, which uses two

parallel 1080p (1920x1080x3) High Definition video cameras at 30fps rate to generate stereoscopic vision. It is required that each 11.9 MB video frame be processed by a server and returned within 1/30th of a second (~33ms). We propose the development of a protocol where data is captured, transmitted, processed, returned, and displayed within 33ms for the left and right eye synchronization of the daVinci.

Through the use of high-speed access to a high-performance computing resource, we believe that it will be possible to update models based on current surgical data. These methods will allow for real-time modification of the previously generated object map, based on current data. Object map is segmentation of volumetric images into multiple object regions, which stores the object information of every voxel in the image. For spatial volume registration, object maps allow structures segmented from different modalities to be combined with proper spatial relationships). We plan to take the augmented data previously generated and send it to the cluster for adaption and deformation. We will compare the developed algorithms on central processing unit (CPU) and graphics processing units (GPUs)

1.3 Dissertation Outline

Chapter 2 provides a background on laparoscopic surgery and image guided surgery as well as literature review of relevant work in the field of medical image processing and high performance computing. Chapter 3 provides background, method and results on our network infrastructure. Chapter 4 focuses on the medical image processing algorithms such as segmentation and registration. Chapter 5 details the experiment setup on Clemson's

supercomputer with dual Nvidia GPUs and evaluation of the medical image processing parallelization. Chapter 6 discusses an MPI-CUDA implementation of our framework. Finally the dissertation is concluded in chapter 7 with directions for future research.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

2.1 Laparoscopic surgery and image guided surgery

Minimally invasive surgery (MIS) reduces patient trauma and recovery time by using small "keyholes" instead of traditional open surgery. Surgeons insert a small video camera into the patient's body for visualization and guidance and use small tools to perform surgical procedures. MIS reduces size of incisions, patient recovery time, trauma and post-operative pain. However, small field of view will result in small visual field for surgeons, which could prolong procedures and does not provide information about subsurface structures [7].

Figure 2.1 - Laparoscopic procedure

Image guided surgery (IGS) uses pre-operative (for example magnetic resonance imaging (MRI) or Computed topography (CT)) or intra-operative (such as ultrasound (US) or fluoroscopy) medical images to map a surgical region of interest, providing surgeons with information regarding internal structures and their geometric relationships [7]. IGS systems relate measurements in one coordinate system to measurements in another. Determining mathematical relationship between objects in images with their physical location in the operating room can be done based on points, surfaces or volumes. This process is called registration and the images or instruments can be overlaid on other images, laparoscopic video or the patient. High-speed computing hardware will enable interactive registration and real-time surgical image guidance during procedures.

2.1.1 Robotic surgery

After the introduction of the Computer Motion AESOP and the Intuitive Surgical daVinci, both in 1998, the use of robotics in medicine has continued to grow. They play an important role in minimally invasive procedures by enhancing precision and dexterity. Currently, the primary robotic surgery devices on the market are the daVinci-si and daVinci-xi systems (Intuitive Surgical, Sunnyvale, CA, USA). The daVinci-si robot consists of the surgeon's console and patient side cart [16]. The high definition (HD) vision cart uses two parallel 1080p (1920x1080x3) high-definition video cameras to generate stereoscopic vision and allows clinicians to perceive depth. An example of a daVinci-si surgical system with added second surgeon console is shown in Figure 2.2.

Figure 2.2 – daVinci surgical robot [16]

Robotic assisted surgery also relies on laparoscopic cameras for vision and does not allow information of subsurface structures. Image guided surgery (IGS) allows visualization and evaluation of critical structures. Registration of tomographic medical images can be done using intrinsic (kinematic chain of the robot and robotic cart), extrinsic (optical or magnetic tracking systems) or hybrid localization in the operating room [12].

2.1.2 Image guided abdominal surgery

IGS has become the standard of care in neurosurgery that allows more specific resection with higher specificity and minimizes damage to healthy tissue. Image guided surgery for abdominal organs deals with challenging registration and segmentation methods that are caused by organ motion and soft tissue deformation. Real-time registration and remodeling of data are necessary during these surgical procedures. Pre-operative images can be used to understand the patient's anatomy and plan a surgical process. Intra-operative images can be used for real-time observation of the surgical field, as the position of internal organs is dynamic [19].

2.1.3 Augmented reality partial nephrectomy

Partial resection of the kidney or removal of diseased segment is called partial nephrectomy. Partial nephrectomy for renal cell carcinoma has become more common as it leaves a clear margin for the normal functioning part of the kidney and preserves renal function. Identifying clear margins can reduce procedure time and result in a resection with higher specify and tighter margins. Some of the advantages of laparoscopic partial nephrectomy compared to open partial nephrectomy are reduced blood loss, recovery time and hospital stay [20, 21].

daVinci robotic surgical systems provide advantages over traditional laparoscopic surgery such as improved dexterity, precision and high-quality 3D operative view. Augmented reality (AR) can be used with da Vinci robots to provide the surgeon with additional information and enhance the surgical procedure. AR systems enable overlay and alignment of pre-operative or intra-operative medical images onto the surgical field of view and improve surgeon's visualization, view of the margin and surgical outcome. However, intra-abdominal organs such as kidney change size and shape during procedures. Surgical tools and conditions, retractors and physiological changes may cause kidney deformation. Examples of image registration techniques for kidney include: manual registration using surgeon's knowledge of human anatomy, surface based registration that aligns surface anatomy with medical images, and stereoscopic registration using disparity between two cameras to triangulate placement of an object within a camera frame [20]. Some of these examples are presented in Figure 2.3.

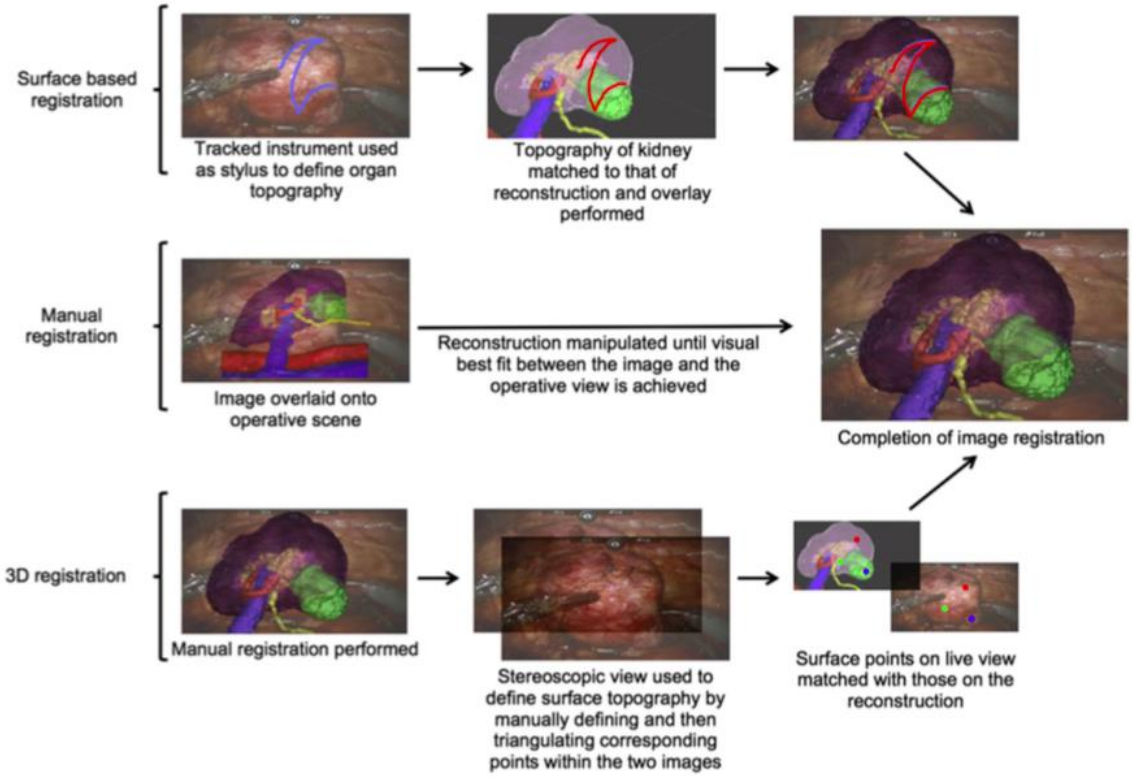


Figure 2.3 – Kidney image registration [20]

2.2 Medical image processing on GPUs

Medical image registration is the process of aligning medical images into a common coordinate space. Registration can be done using intrinsic or extrinsic registration. During intrinsic registration, features are based on the patient and extracted visually or computationally from input images. These features such as anatomical landmarks or image segmentation are used for matching and registration of images. Extrinsic registration uses artificial markers such as foreign objects and fiducial markers [22]. Figure 2.4 shows a

general registration solver. The main components include preprocessor, transformer, optimizer and similarity measure [14].

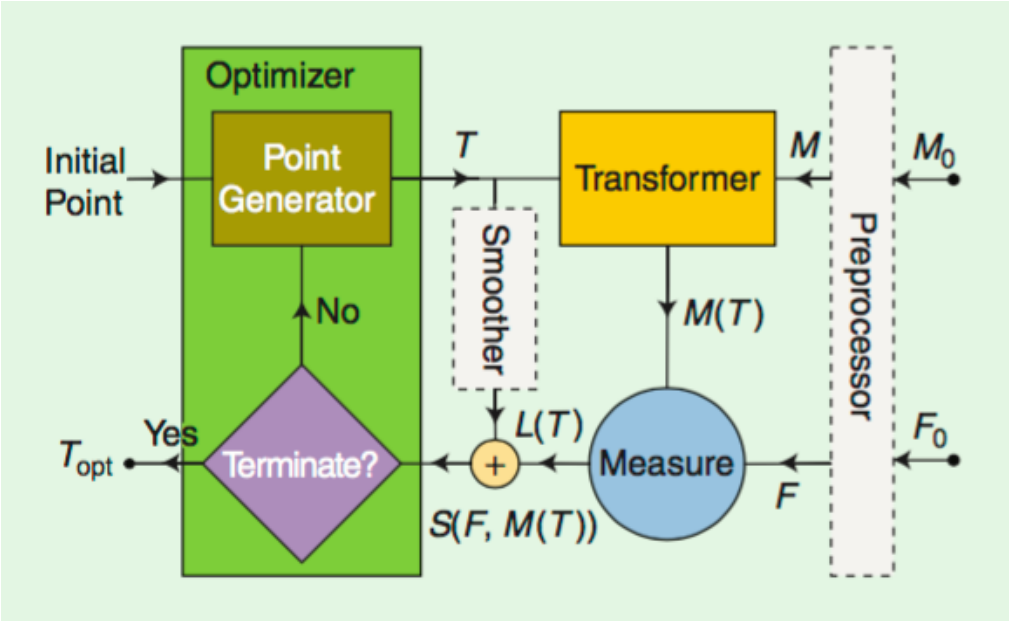


Figure 2.4 - Registration solver [14]

Image registration is computationally expensive and with increasing resolution and size of medical images or videos, real-time processing on a bedside or local PC has become more challenging. As a result, recent research on image processing using cell broadband engine (Cell/BE), field programmable gate arrays (FPGAs) and GPUs have increased.

Image segmentation divides elements of an image into a set of groups. Medical image segmentation can be used to segment brain structures, organs, tumors, etc. for diagnostics and intraoperative guidance. Segmentation algorithms are usually specific to an application and allow visualization of region of interest without unnecessary information [23].

2.2.1 Medical image registration on GPUs

One of the main goals of recent image registration research is performing real-time or near real-time calculation to provide more accurate information during surgical procedures and intraoperative applications. Depending on the application, these algorithms may be high throughput computing or require additional work and modification to transfer from CPU to GPU for faster processing. Several groups have reported faster linear interpolation on GPUs compared to CPU implementation. Shams et al. investigated real-time registration of 3D CT scans on a GPU using mutual information [24]; the CPU implementation took 45 seconds while the total time reduced to 3.7 seconds using a GPU. Brounstein et al. performed registration between CT and ultrasound on a GPU in 2 seconds [25]. The work reported by Ruijters discussed non-rigid registration between pre-operative and intra-operative 3D cone-beam CT volumes [26]; volumes of the size 256 x 256 x 256 voxels were registered in 329 seconds on CPU, 31.2 seconds using multi-thread and 7.4 seconds on a GPU. Using several GPUs instead of one could also reduce the total registration time further. The work reported by Plishker et al. compared CT image registration using one and four GPUs [27]; using a single GPU the registration was performed in 7.9 seconds and four GPUs reduced the time to 2.5 seconds. Optimization of naïve GPU implementations that are compute or memory-bound can also result in further speedup. Efficient GPU implementations can also reduce overall time by minimizing CPU-GPU communications. For example, Hwang et al. optimized a GPU implementation that resulted in a speedup of a factor 6 [28].

2.2.2 Medical image segmentation on GPUs

Image segmentation is also a computationally demanding procedure especially for large medical image datasets. GPUs can be used to compare multiple segmentation algorithms or perform interactive segmentation and visualization. Some of the common segmentation algorithms that have been executed in parallel include: thresholding, level-set, active contours, region growing and watershed transform [29]. Active contours segmentation uses moving contours and focuses on minimizing their energy. Perrot et al. focused on segmentation by active contours of large images (15-150 megapixels) and reduced the processing on GPU by a factor of 7 [30]. Level-set segmentation is similar to active contours, which propagates a contour in the image; however, it has the advantage of splitting and merging these contours without additional processing. In [33], the authors implemented a parallel interactive level-set segmentation and achieved 10x to 15x speedup. Watershed segmentation uses intensity value of pixels as their height to find watershed lines. Korbes et al. reported a parallel implementation of watershed segmentation that is six times faster than the serial version [32]. Real-time processing of dynamic images requires streaming this data directly to GPUs. Novontny et al. performed real-time instrument detection and tracking in ultrasound volume in 31 ms on a GPU [31].

Most of the work has focused on developing medical image processing algorithms for local CPUs. Several groups that have worked on using a network link for transfer of medical data and images: NifTK, PLUS, MITK-OpenIGTLink.

OpenIGTLink is a TCP based network protocol for medical data transfer. It has been mostly used for connecting devices and software within an operating room (OR) or computers on the same network [34]. NifTK uses NiftyIGI as the image guided intervention applications and NiftyLink for transfer of data. NiftyIGI provides visualization platform as well as receiving video and tracking data. NiftyLink sends OpenIGTLink messages using a client and server model [35]. Public software library for ultrasound (PLUS) has been developed for ultrasound guided intervention systems. It also uses OpenIGTLink protocol for receiving and streaming live data [36]. Medical imaging interaction toolkit (MITK) uses OpenIGTLink as the network interface to allow cross-application within computer-assisted intervention systems [37].

Chapter 3

DATA TRANSFER FOR REMOTE PROCESSING OF MEDICAL IMAGES

3.1 Introduction

In laparoscopic procedures, a camera is used for real-time visualization of the surgical field and guidance of the procedures; however, the small field-of-view of the laparoscope will result in a small overall visual field of the underlying tissues of interest. The reduced visual field along with organ motion can limit the surgeon's ability to target specific subsurface locations with high accuracy and precision, therefore, requiring external information such as medical imaging to complement laparoscopic video [1,2].

For this work, we will look at the daVinci-si robot, which uses two parallel 1080p (1920 × 1080 × 3) high-definition video cameras to generate stereoscopic vision. We will develop image processing algorithms for laparoscopic video with medical images on HPC clusters. Computing clusters are often not available in hospital environments, thus data will need to be transported from the point of care to be processed and the results will need to be returned in real time. To realize the goal of rapid processing of laparoscopic data, we propose the development of a protocol where data is captured, transmitted, processed, returned, and displayed within 1/30th of a second for the left and right eye synchronizations of the daVinci. This chapter includes comparison of different network protocols for transferring medical images and video to Clemson's Palmetto cluster.

3.2 Methods

3.2.1 Network model and protocols

Network designers use a layered architecture to design a network and protocols to specify services. Each layer can be implemented using software, hardware or a combination of both. Defining protocols by layers has many advantages. For example, each layer has a specific function and receives services from other layers. In TCP/IP model, Internet protocols are specified in five layers: Application, Transport, Network, Data link and Physical Layer as shown in Figure 3.1. Application and transport layers are implemented in software, Data link and physical layers are implemented in a Network Interface Card (NIC) such as Ethernet or WIFI, and Network layer is often a mix of software and hardware implementations [38].

The reference model, OSI model is defined in seven layers: Application, presentation, Session, Transport, Network, Data link and physical layer. Two layers of presentation and session that are not in Internet model are defined in application layer based on application specification. OSI and TCP/IP models are shown in figure 3.1.

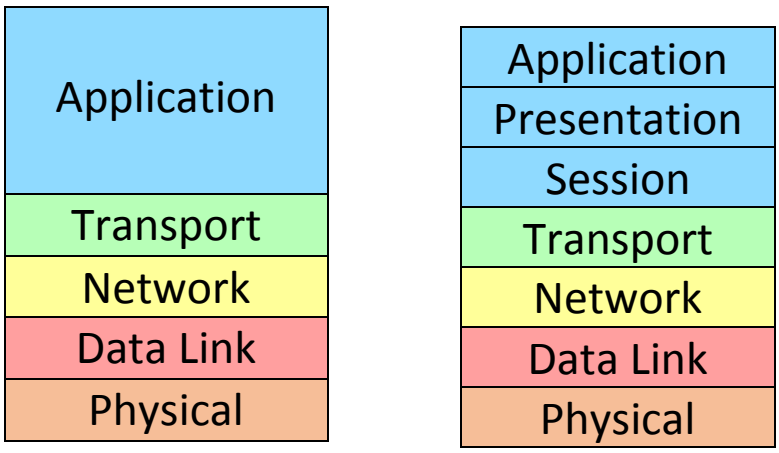


Figure 3.1 – TCP/IP (left) and OSI (right) model

Network applications are in application Layer. Protocols available for this layer are: HTTP (Web request and Transfer), SMTP (transferring email) and SFTP (for transferring files between computers). Transport layer transports application layer's messages between systems. Two transport protocols are: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The main function of the Link Layer is to check transmission error and manage data flow (Ethernet and WIFI). Physical layer is the transmission medium of the link (wired or wireless); fiber optic cable is an example of wired network and RF transceiver for wireless network.

3.2.2 Network delay and packet loss

In a packet switched network, data in the application layer is divided into groups of bits that are called packets. Ideally packets are transferred from source to destination without loss and delay. However, packets are passed through different routes and routers that will cause delays or errors. Four types of delays are: Processing, queuing, transmission, and propagation.

Processing delay is the time required to check a packet's header to specify route and direction. It also includes time for checking bit error in the packet. Processing delay in routers is usually in order of microseconds. After processing, packet is directed to the queue of the next link.

Queuing delay is the time that packet waits in queue to be transmitted onto the next hop link. If there is no packet in the queue, the queuing delay is zero and packet is transmitted to the next hop. If there are some packets in the queue, the queuing delay depends on the number of packets in the queue. It could be in the order of microsecond or milliseconds. The service policy is usually first come first served.

Transmission delay is the time to send a packet of L bits to the next node. If the link transmission rate is R bits per second, the transmission delay is L/R. The rate could be Mbps up to Gbps. For example, if the packet length is 1500 bits and the transmission rate is 100 Mbps, the transmission delay is

$$Transmission\ delay = \frac{L}{R} = \frac{1500}{100x10^6} = 15\mu sec$$

Propagation delay is the time required to send a packet from one router to the next. This time depends on the distance between two routers and speed of the link. Therefore, propagation delay is equal to:

$$Propagation\ Delay = \frac{d}{s} = \frac{Distance\ between\ Routers}{Propagation\ speed}$$

Propagation delay is negligible if two routers are close or on a campus of a university. However, if two routers are not close to each other, propagation delay should be considered.

Queuing delay is difficult to predict. Several research groups are working on calculating probability of having more than specified delay and also average queuing delay. It is usually assumed that the buffer is infinite and all of the packets can be stored in the buffer. However, an actual buffer is finite and if a packet arrives and the buffer is full, the router will drop the packet and results in packet loss. A packet loss will be detected and lost packet is retransmitted if TCP is used for the transport layer.

The sum of all of the types of delays is called total nodal delay. Total nodal delay is defined as:

$$\mathrm{d_{nodal}} = \mathrm{d_{proc}} + \mathrm{d_{queue}} + \mathrm{d_{trans}} + \mathrm{d_{prop}}$$

If there are N routers from source to destination, End-to-End delay (the total delay) from source to destination and is equal to:

$$d_{End-to-End} = N(d_{proc} + d_{queue} + d_{trans} + d_{prop})$$

Figure 3.2 shows the connection between source (client) and destination (server) with N routers in between.

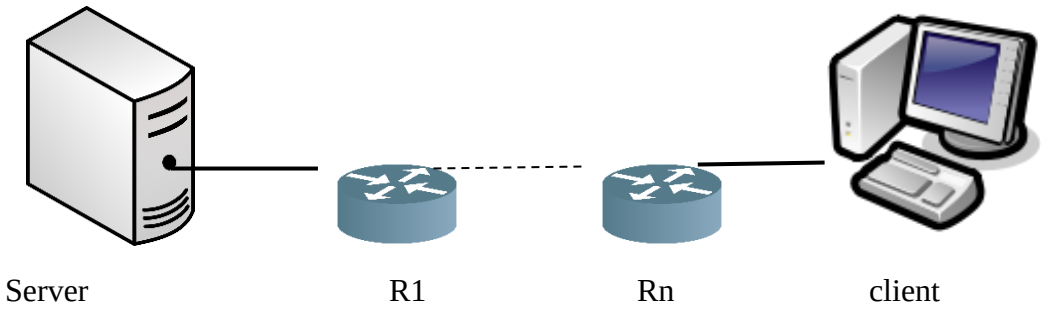


Figure 3.2- Client Server connection with N routers

3.2.3 TCP and UDP

User Datagram Protocol (UDP) is a connectionless transport layer protocol. There is no handshaking between source and destination before transmitting the packet so it is connectionless and no acknowledgement is sent from receiver to transmitter for received correct packets. Therefore, there is a unidirectional data flow in UDP connection. If a packet is lost, there is no retransmission of the packet. There is an error checking field bit that is added in the header field of the packet, which is used at the receiver to detect bit error. UDP is usually used for multimedia applications, since there is no delay for connection establishment. There is no connection state for sender and receiver, which makes handling many active clients at the same time easier. However, this protocol is not reliable and cannot be used for loss sensitive data i.e. real-time surgery.

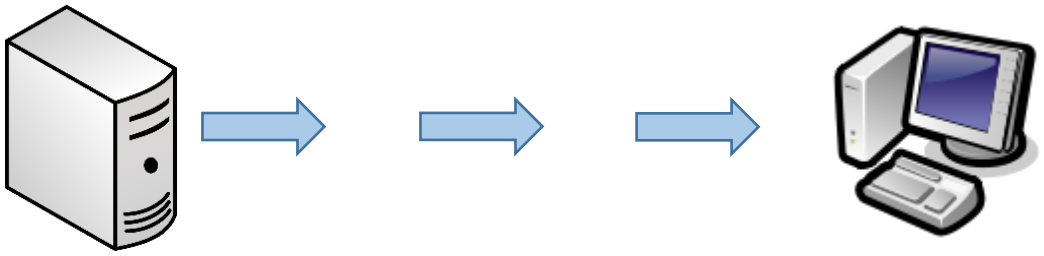


Figure 3.3- Unidirectional UDP Connection

Transmission Control Protocol (TCP) is a reliable and point-to-point connection between the source and destination. Before transferring data, there is a 3-way handshaking between source and destination, which adds overhead. TCP provides full duplex data transfer, meaning there is bi-directional data flow. An acknowledgement is sent from receiver to transmitter if the packet is received without error. If there is a bit error in the packet or packet is lost due to a full buffer in routers, the packet is retransmitted. A timer is set at the transmitter for each packet. If the acknowledgement of a packet is not received during the specified time, packet is retransmitted. In TCP, packets are numbered using segment number and are delivered in order. If a packet is received out of order, depending on the version of TCP, it can be buffered until previous packets are received.

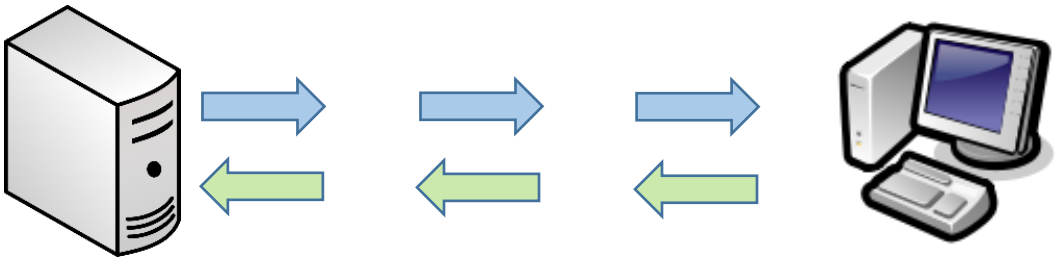


Figure 3.4- Bi-directional TCP Connection

3.2.4 Network infrastructure

Prior to the design of our overall system, network latency, bandwidth and speed were tested in a series of time-of-flight experiments. For our study we developed three separate applications that run simultaneously which serve the purposes of video acquisition (video client), video display (video server), and image processing (video server palmetto).

Currently a single PC is used for the acquisition and display of images (video client and video server), this machine is outfitted with a Piccolo Alert (Euresys Inc., San Juan Capistrano, CA, USA) SD frame grabber board, connectivity is provided over a 10 gigabit per second (Gbps) Ethernet connection. Our video processing module (video server palmetto) is running on the Palmetto compute cluster; rated number 4 amongst academic research clusters. Palmetto contains 1,978 compute nodes (20,728 cores) and 598 NVIDIA Tesla GPU accelerators [39]

The distance between the client (our local computer) and server (Palmetto Cluster) is 9 miles. Our local computer is placed in Clemson, SC and the cluster is located in Pendleton, SC. The connectivity from the outside to the Palmetto cluster is 10 Gbps. The communication within the Palmetto cluster (between nodes) is provided over InfiniBand or Myrinet. There is a 100 Gbps connection from the cluster to the Internet2 innovation platform, which provides rapid connectivity for learning and research between more than 100 U.S. universities. We have tested our transfers using our local area network and plan on connecting to multiple institutions through Internet2. Preliminary network tests were conducted using the workflow shown in Figure 3.5.

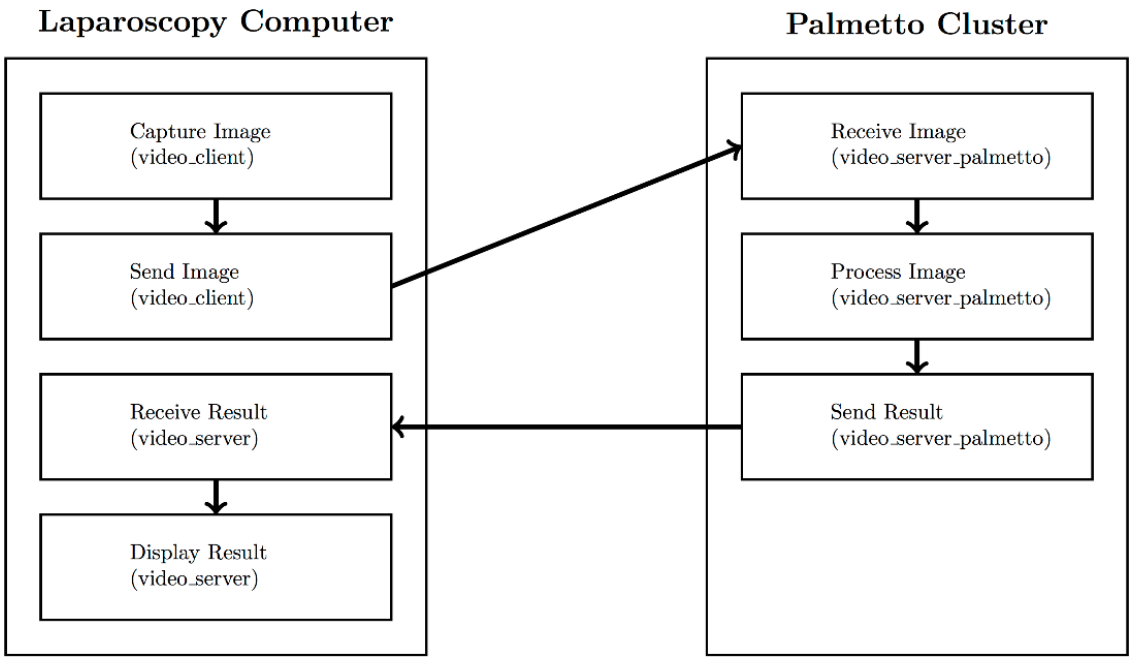


Figure 3.5 - Workflow for network tests, demonstrating the connectivity between the local laparoscopy computer (left) and the HPC Palmetto Cluster (right).

In our workflow we currently have tested different protocols and methods such as the secure file transfer protocol (SFTP), UDP and message passing interface (MPI) for the transfer of captured frames from the client machine (our local computer) to the HPC server (Palmetto cluster). We have measured the round-trip time for captured images of 57KB and have compared our results using the 1Gbps and updated 10Gbps network connections. We are trying to minimize network overhead and latency so we can use the HPC resources to improve the visualization and accuracy of laparoscopic video and provide surgeons with real-time augmented laparoscopic data during surgical procedures.

Clemson University has also recently deployed the OpenFlow software-defined network (SDN) communication protocol on its network and is in the process of leveraging the new protocol and its connection to the Internet2 Innovation Platform. SDN enables programmable network control for high-bandwidth and dynamic applications [17]. The

new network will assist researchers in overcoming the network limitation to flexibly establish 10 GBPS or higher end-to-end connectivity, within a scientific demilitarized zone (DMZ), thus creating a frictionless connection between machines [18].

SDN provides the ability to deploy and manage networks dynamically. Software defined networking started in 2010 and changed the network industry in 2011. SDN separates data plane from control plane and facilitates network monitoring and management. For Internet routers and Ethernet switches, control is implemented and coupled in the devices. In SDN, a logically centralized controller has a global view of the network and collects information from network resources. Routing algorithms are placed in a central section called controller. Controller receives policies and instructions and network information from network resources. Controller uses this information to find the best routes and makes the forwarding table. The forwarding rules are installed on the flow table of data plane switches that forward traffic based on the forwarding rules. Switches are responsible for forwarding the data according to the flow table rules which are made by the controller [40].

Routing is implemented for each flow. When a flow enters a switch, the switch compares flow fields with the flow table. If it matches the entry of the table, the corresponding action will be done; otherwise the switch uses OpenFlow protocol to send the flow to the controller. OpenFlow (OF) is one of the most common standards for SDN architecture. It contains many features for network monitoring and reconfiguration and allows network providers as well as researchers to add functionality to this protocol. The

controller calculates the route for this flow and adds an entry with flow fields and suitable action to the flow table.

The separation of control and data plane together with programmable nature of SDN, allows using it as an underlying technology for delivering online video and images in an efficient manner. SDN has some advantages such as intelligence and controllable architecture, less dependency on hardware technology, simple and flexible management, no dependency to a specific company's products, faster innovation implementation and testing. Policies can be applied easily using the flow tables[40, 41].

SDN architecture is shown in Figure 3.6:

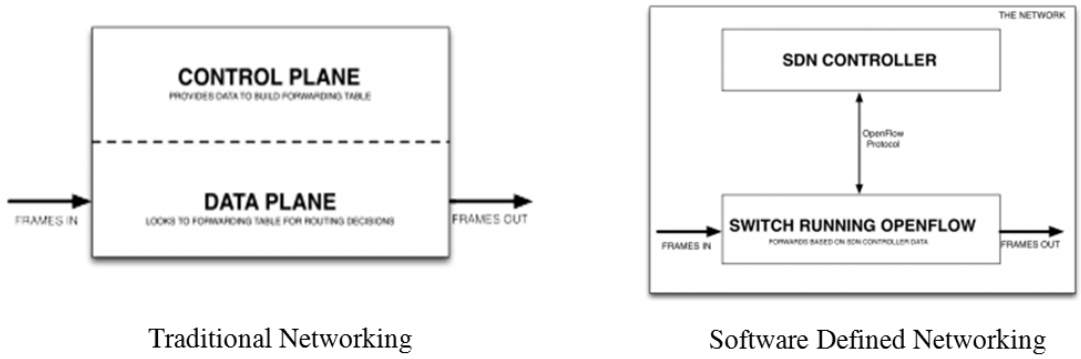


Figure 3.6 - Software Defined Network Architecture [69]

Steroid OpenFlow Service (SOS) is an OpenFlow-based network service that can seamlessly increase the performance of large data transfers over long-distance and high bandwidth networks (i.e. large delay-bandwidth- product networks) [42]. SOS takes advantage of the fact that TCP is typically unable to consume the available network bandwidth over large networks. TCP's algorithm only permits the connection source to send its window-size number of packets before receiving an acknowledgement. In a large

network, this can result in the sender waiting idly for the acknowledgement before being permitted to send additional packets.

SOS works by redirecting the TCP connection to a local SOS agent using an OpenFlow switch in the network path. The local SOS agent transparently acknowledges the packets from the source of the data transfer acting as if it is the intended destination. Because the local SOS agent is installed on the source's local network, the acknowledgement delay is small compared to the real destination, and the source can continuously transfer data to the local agent.

As the source SOS agent (surgeon console) accumulates a buffer of data from the data transfer source, a destination SOS (HPC cluster) agent is located in close proximity to the intended destination. The source and destination SOS agents communicate and agree to use a number of parallel TCP connections in order to rapidly transfer the data from the source SOS agent to the destination SOS agent. The destination SOS agent collects data from the parallel TCP connections and presents it to the intended destination as if it originated from the source across the large network. Figure 3.7 indicates the SOS network setup:

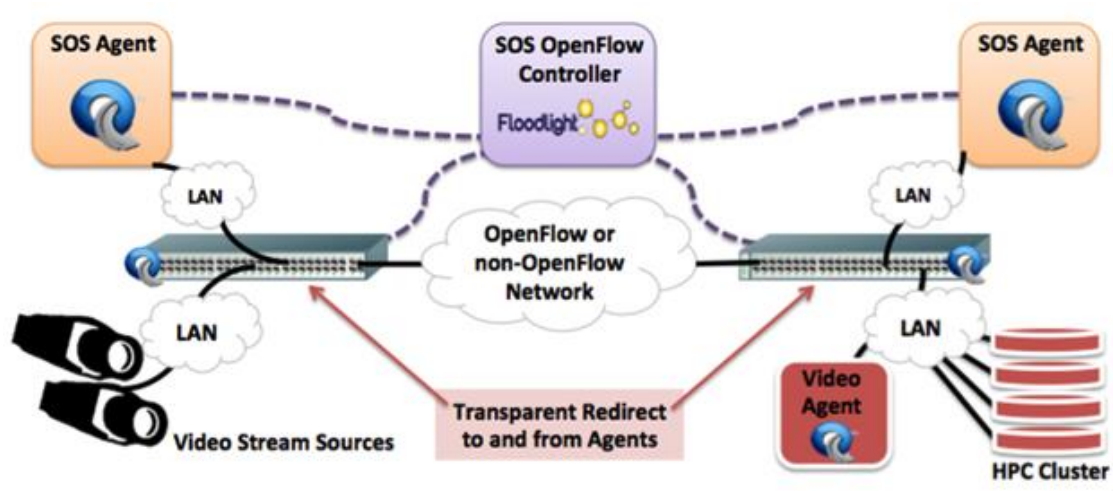


Figure 3.7 – SOS network setup

3.3 Results

The connection from our lab (cutters) to the cluster and back to cutters is shown in the figure below. We connect from cutters to Palmetto's user node though a 1Gb/s or 10Gb/s connection. The user node directs our job submission to a compute node for processing:

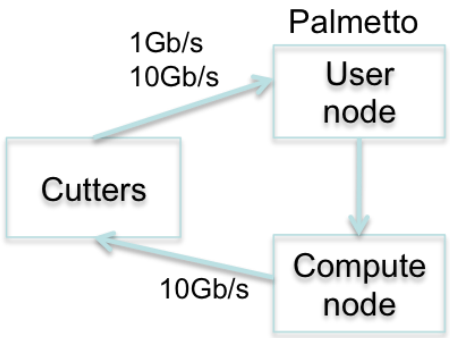


Figure 3.8 – Cutters to Palmetto's user node connection

Data from the time-of-flight tests for video frames using the SFTP file transmission protocol showed an increase in transmission time as the frame number increased, for both 1Gbps and 10Gbps networks, except for when the protocol had completed sending of

previous data which appears as valleys in the graphs. However, the round-trip time decreased by a factor of five to seven when using the 10Gbps network instead of 1Gbps. Figure 3.9 shows the file transfer using 1Gbps connection. The round-trip time is around 5seconds for the first file and 35seconds for the last. Figure 3.10 indicates the transmission using 10Gbps network. The round-trip time starts at 1s for the first frame and 5seconds for the 100th frame (note the difference in y-axis scales between the graphs).

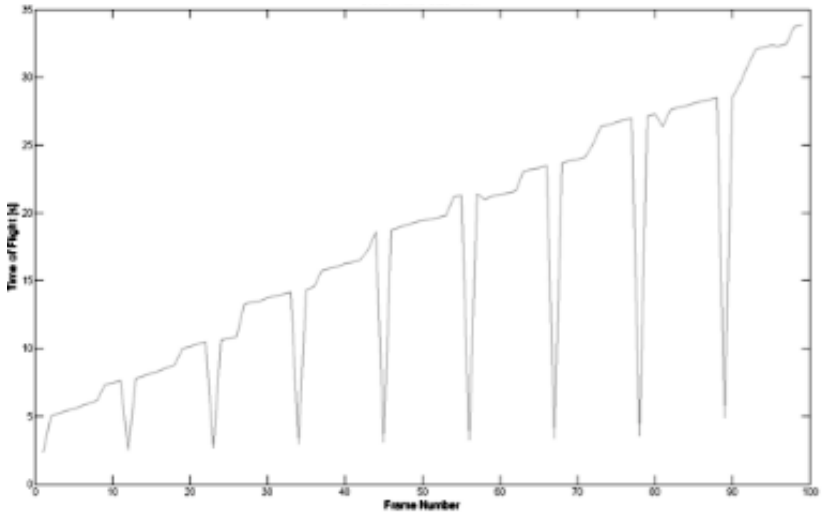


Figure 3.9– Round-trip time for SFTP using 1Gbps

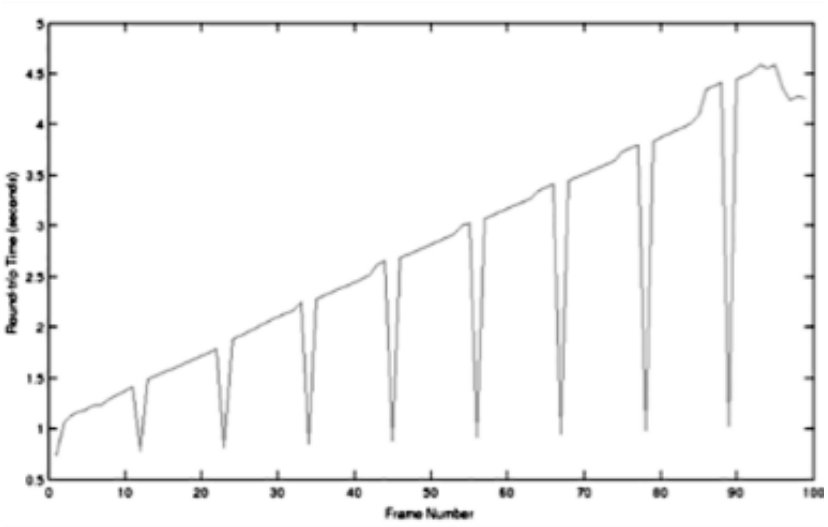


Figure 3.10– Round-trip time for SFTP using 10Gbps

The graphs show an increase in time as the number of frames increase and this could be caused by the user node of the cluster and que time. As a result, we also tested a direct connection from Cutters to a compute node on Palmetto.

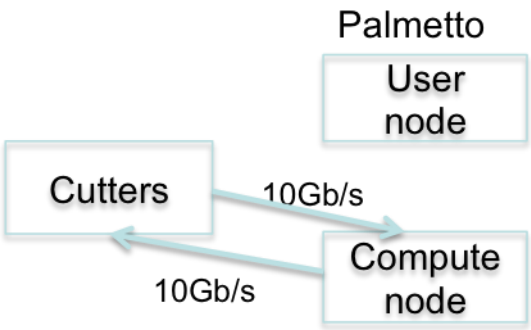


Figure 3.11– Cutters to Palmetto's compute node connection

After connecting directly to the compute node, the round-trip time decreased to 4ms and remained around the same value for all the frames. Figure 3.12 indicates the round-trip time in seconds between our local computer (client) and the Palmetto cluster (server).

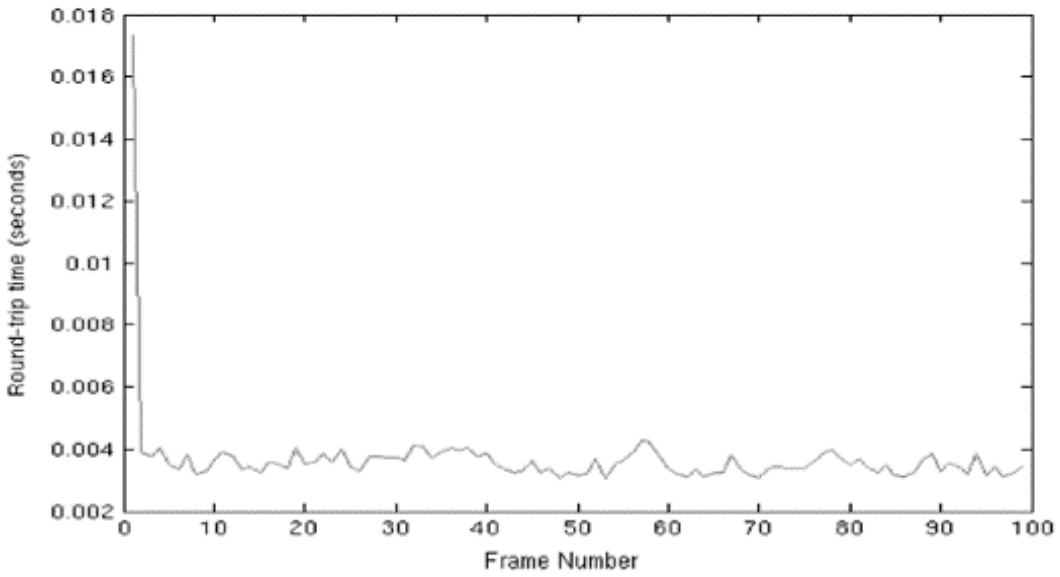


Figure 3.12–Round-trip time for SFTP through compute node using 10Gbps

SFTP runs over TCP, which offers end-to-end connection and ordered data transmission. However, the connection-oriented communication causes delays during data transmission. On the other hand, UDP creates a connectionless communication that will decrease the transfer time. This could result in packet loss depending on the network bandwidth and connectivity. We transferred the files using UDP and the overall transmission time decreased to 0.7ms with no packet loss using the 10Gbps network. Figure 3.13 indicates the round-trip time using UDP.

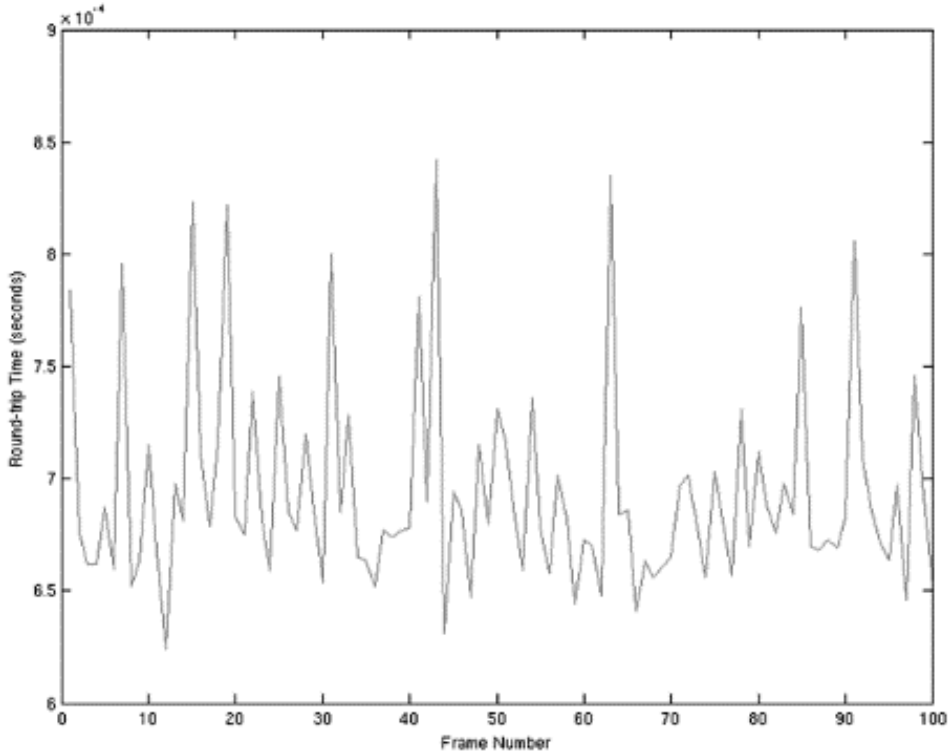


Figure 3.13– Round-trip time for UDP using 10Gbps

UDP and TCP are both in the transport layer of computer networks and have a package size limit of 64 KB. Consequently, larger files have to get reassembled before processing. As a result, we focused on the application layer and protocols that use UDP or a TCP connection, but handle the reassembly of large files. We tested our transfers using MPI. MPI is a message passing system and communications protocol that can be used for parallel programming [43]. We transferred the files using this method, and the results indicate that the transmission time is around 1.2 ms. Figure 3.14 shows the round-trip time using MPI.

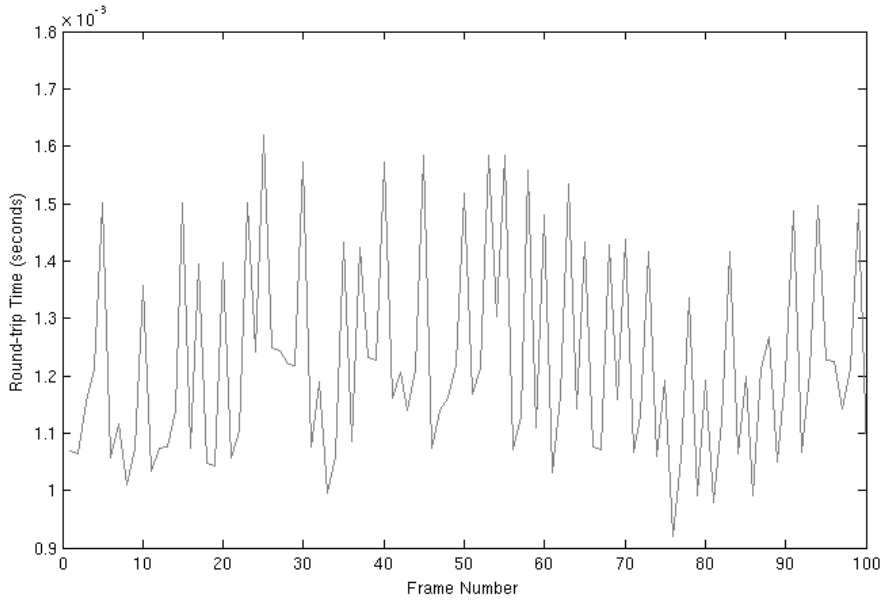


Figure 3.14– Round-trip time for MPI using 10Gbps

We also tested our transfers using MPI for larger files (6 MB) to measure and compare the transmission time for HD videos. The results indicate that the transmission is around 53 ms. Figure 3.15 shows the round-trip time using MPI for 6 MB files.

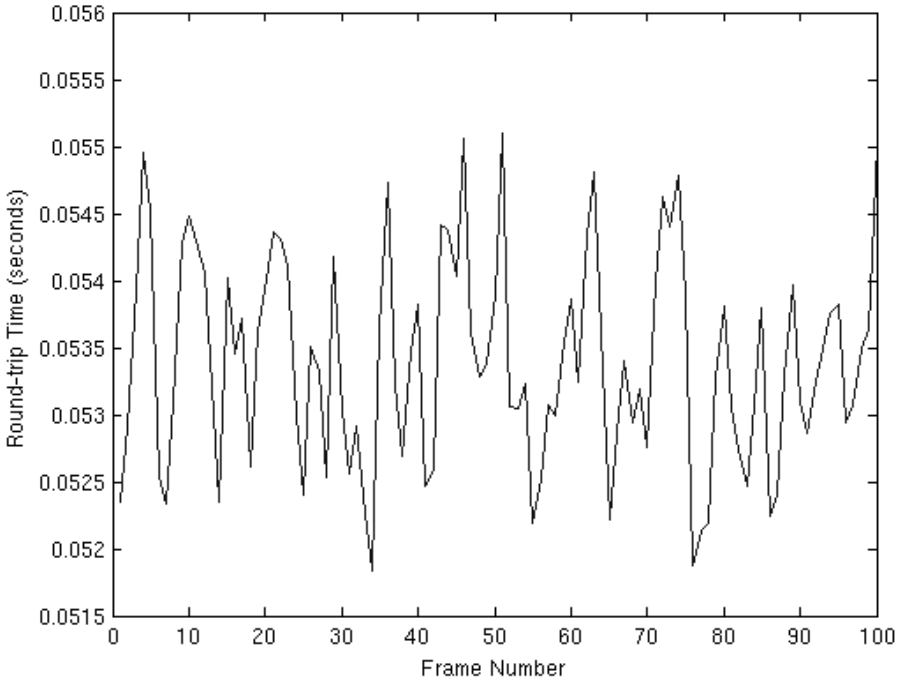


Figure 3.15– Round-trip time for MPI using 10Gbps for 6MB files

In our next step we used SOS to transfer files. SOS can be performance tuned to the environment in which it is deployed. Parameters such as the quantity of parallel connections to utilize, agent buffer sizes, and queuing for lost/out of order data can be customized. Figure 3.16 shows the performance implication of the number of parallel connections and the agent buffer size. As shown, for a 10Gbps link that can only achieve 130Mbps with a single TCP connection, SOS can increase the performance of this single connection up to 5.08Gbps. Note that the bottleneck in Figure 3.16 is the CPU of the agents. With better hardware, higher performance could be achieved. However, the daVinci only generates 2.88Gbps of raw video data. Thus, SOS can improve the network performance to support video stream transfer over TCP, which ensure reliability.

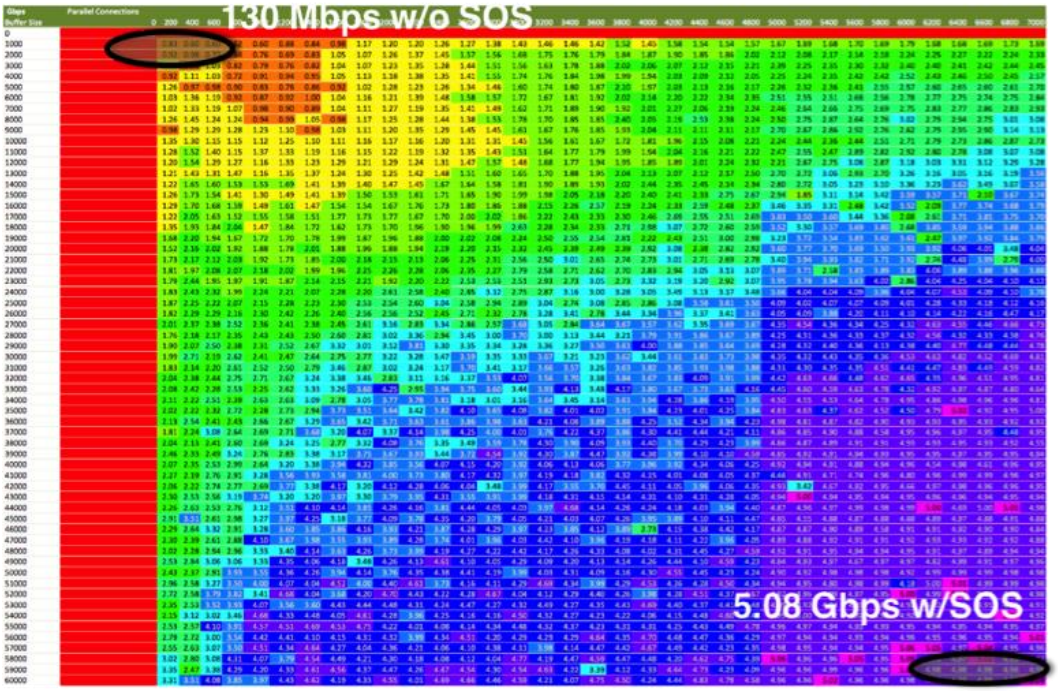


Figure 3.16– SOS parameter sweep

3.4 Discussion and Conclusions

In the field of computer assisted-surgery, particularly in image-guided surgery the ability to acquire, process, render, and visualize data in real-time is essential for performance of complex tasks, minimizing risk to the patient [29]. While complex surgeries have been performed for hundreds if not thousands of years, the advent of new computer based technologies has allowed for new levels of accuracy and precision, thus reducing postoperative complications, allowing faster recovery times, and causing less collateral damage to otherwise healthy tissues. The contextual validity of data degrades quickly as tools within the body move, patient organs move, and resections are performed. It is critical that data be acquired, processed, and returned in a timely manner [44].

Initial results from the use of SFTP ruled out its use due to high latency in time of flight measurements as well as noticeable increase in latency over the course of the test. During our tests we realized that access to the underlying file system and the fact that we must connect to each computing node over SSH through a head node, which is accessed by a large number of users, is causing significant latency issues. We were able to reduce the latency through direct connection to the compute note. However, SFTP runs over TCP, which uses acknowledgements and retransmission to prevent packet loss and causes delays. The encryption step of SFTP is also a limiting factor in reducing the round-trip time.

UDP is faster since there is no error recovery and acknowledgment mechanism; therefore, it is used for time-sensitive and real-time applications. The UDP tests resulted in a lower round-trip time and reliable connection with no packet loss due to availability of high bandwidth. However, UDP and TCP are both in the transport layer and have a packet size limit of 64 KB; hence, larger files have to get reassembled before processing. As a result, we focused on the application layer and protocols that use UDP or TCP connection, but handle the reassembly of large files. We tested our transfers using MPI, which is based on a TCP connection and is used for parallel programming, larger files, and HD video frames. Despite MPI using TCP connection, the round-trip time is 1.2 ms comparable with large files. In addition, the round-trip time for MPI is in the order of milliseconds compared with seconds for SFTP. The total transfer for HD video frames is around 53 ms using MPI. We will optimize and parallelize our algorithms to reduce the time. Through the development of a direct connection, we can achieve more direct control over of the

transmission of data as well as develop a communication system between clients and computing nodes to perform real-time medical image processing.

Security of medical data transfer (images and video) is also an important factor. Data security refers to technology or tools used to protect the data from unauthorized access and help professionals to keep patient's information in confidence. Data encryption algorithms are used to secure the data before transmission. Encryption can be implemented in software or hardware. There are several encryption algorithms that can be used for different applications. Depending on time constraint of the application and time of implementation of the algorithm, one of them is selected. Hardware implementation of the algorithm using Digital Signal Processing (DSP) or Field-Programmable Gate Array (FPGA) will reduce the time of encryption [45]. This will be investigated further before transferring real-time surgery and imaging data.

The augmentation of live high definition stereo laparoscopic data with pre-operative or intra-operative medical images requires more computational and storage capability than is possible on a single workstation. More advanced visualizations and processing may require new or improved rendering techniques more suited to a high-performance computing environment. Therefore, flexibility in the transmission of data to multiple sites is important. Furthermore, the ability to optimize the data transmission to take advantage of the specialized processing and rendering techniques at multiple endpoints is the key to allowing for rendering and processing as a service and also providing optimal data.

Through the use of high-speed access to an HPC resource, we believe that it will be possible to update models and allow for real-time modification of the previously generated object map, based on current surgical data. Success in this portion would lead to the ability of performing surgery at one site (origin), receiving an object map from another site (segmentation/rendering), forwarding this data along with relevant force data to a third site for deformation (model update), and returning this data to the origin for display.

Chapter 4

MEDICAL IMAGE REGISTRATION AND SEGMENTATION FOR LAPAROSCOPIC SURGERY

4.1 Introduction

Minimally invasive surgery reduces patient trauma, hospital stay, and recovery time through reducing incision size; however, the small field-of-view of the laparoscope as well as small incision size will result in a small overall visual field of the underlying tissues of interest. The reduced visual field along with organ motion due to natural and surgical effects can limit the surgeon's ability to target specific subsurface locations with high accuracy and precision; therefore, requiring external information such as medical imaging to complement laparoscopic video [1,2]. Image-guided surgery (IGS) uses preoperative and intraoperative images to map a surgical region of interest, thus providing surgeons with visualization of subsurface structures and accurate positional information during a surgical procedure [6, 7]. The accurate co-registration of an IGS system with laparoscopic video can reduce the impact of limited surgical access and allow for a resection with higher specificity and tighter margins, thus sparing more healthy tissues [8–11].

After the introduction of the Computer Motion AESOP and the Intuitive Surgical daVinci, both in 1998, the use of robotics in medicine has continued to grow. Some of the clinical benefits of medical robotics are improvement in dexterity and accuracy of

manipulation compared with traditional laparoscopy [11]. Currently, the primary robotic surgery devices on the market are the daVinci-si and daVinci-xi systems. The daVinci-si robot has a high-definition (HD) vision cart, which uses two parallel 1080p (1920x1080x3) high-definition video cameras to generate stereoscopic vision. These two views are then displayed using a pair of high-definition monitors set up in a stereopticon-like device [12, 13].

Processing of the volume of data generated by the stereo video feeds would be demanding for a local system, thus high-performance computing (HPC) hardware is needed [14–16], which can be both expensive and large in size. Even though the price of computing hardware continues to fall, the price of HPC hardware can limit it to only hospitals with adequate financial resources, while size considerations can make hardware difficult to place in any operating room. This section includes the serial segmentation and registration algorithms tested on CPUs. The next chapter contains the parallel version of these algorithms that were tested on a HPC cluster with GPUs.

4.2 Segmentation methods and results

4.2.1 Medical image segmentation

Image segmentation divides elements of an image into one or more regions of interest (ROIs) [46]. Some of the most common categories of image segmentation include: thresholding (histogram or slicing techniques), edge detection (detects edges that represent object boundaries) and region growing (starts from seed points and grows until meets desired segmentation result).

4.2.2 Thresholding

Thresholding is a gray value remapping operation:

$$g(v) = \begin{cases} 0 & \text{if} \quad v < t \\ 1 & \text{if} \quad v \geqslant t \end{cases}$$

where v is gray value and t is threshold value. Gray-valued image will be mapped to a binary image. Image will be segmented in two parts with pixel values of zero or one. Several threshold values can be used to segment an image into more than two segments. If n thresholds are used, the image can be segmented into n+1 segments.

$$g(v) = \begin{cases} 0 & \text{if} \quad v < t_1 \\ 1 & \text{if} \quad t_1 \leqslant v < t_2 \\ 2 & \text{if} \quad t_2 \leqslant v < t_3 \\ \vdots & \vdots \quad \vdots \\ n & \text{if} \quad t_n \leqslant v. \end{cases}$$

Thresholding is mostly used for images containing bright objects on dark backgrounds. The difference between gray value of an object with the background will result in

segmented image using thresholding. Different methods of determining pixel value or range of pixel values for a specific application include: 1. interactive selection of threshold by user,

2. using intensity histogram of an object and background to compare distribution values. The values from different histograms may overlap and maxima (peak) of histograms can be used to define a segmentation threshold. This value can be calculated using average of two peaks with gray values p_1 and p_2 : $\frac{p1+p2}{2}$ or gray value at minimum of two peaks: arg min H(v), where H(v) calculates the histogram value at gray value v.

3. using mean and standard deviation of user defined test points for semi-automatic segmentation, and

4. Otsu's method that iteratively tries different values for automatic selection of threshold value. The variance is calculated using weight factors (ω) for probability of correct selection of threshold (t) and mean value of μ [47].

$$\sigma_c^2(t) = \omega_1(t)\omega_2(t) \cdot (\mu_1(t) - \mu_2(t))^2$$

4.2.3 Canny edge detection

Segmentation of an image into separate objects can be accomplished through detecting edges of objects within the image. Edge detection can be used for ROIs that have strong edges compared to their surroundings. Most of these methods use first or second derivative to find areas that brightness or color changes relative to neighboring areas.

Typically edge detection will also detect partial edges that are not closed boundaries. Edge linking will be used to connect these partial edges that will result in an object boundary.

Canny edge detection is based on a step-edge model that detects high quality edges in images that represent object boundaries [48]. It consists of four steps:

1. Smoothing the input image (filter out noise) for example Gaussian filtering,

2. Finding intensity gradient of image (apply convolution masks in x and y directions): Convolving input image f (x , y) with Gaussian function $e^{-\frac{x^2+y^2}{2\sigma^2}}$ will result in a smoothed image. The magnitude (G) of an edge and gradient direction (angle,θ) of each pixel can be calculated using: (G and θ are arrays with the same size as the image)

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

3. Applying non-maximum suppression to the gradient magnitude (remove pixels that are not part of an edge), maximum values along each edge are calculated. First find the direction d_k that is closest to α(x,y). Second if the value of M(x,y) is less than one if its neighbors along d_k, suppress (g_N(x,y) = 0) otherwise g_N(x,y) = M(x,y), where g_N(x,y) is the nonmaxima suppressed image. The resulting image can have edges with single pixel width using an edge-thinning algorithm.

4. Hysteresis thresholding (upper and lower thresholds). Using double thresholding (two threshold values) will partition the ridge pixels to edges and non-edges. If two values for the threshold are t_1 and t_2 where $t_1>t_2$: pixels with gradient value above t_1 are definite edges,

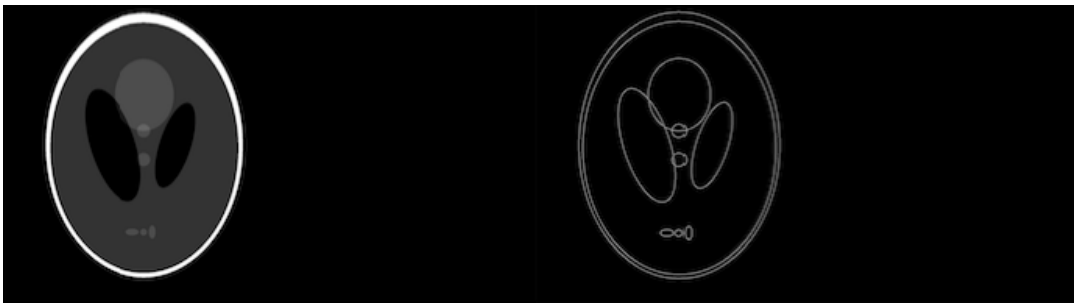
between t_1 and t_2 are potential edges and less than t_2 are non-edges. Connectivity analysis will detect and link edges. The potential edges that are connected to a definite edge using a neighboring potential edge will be considered definite edges.

4.2.4 Segmentation results

The following tests were implemented on Clemson's Palmetto supercomputer with Intel Xeon-E5 CPUs:

4.2.4.1 Canny edge detection

Figure 4.1(a) shows an original image of a HD phantom video and the 4.1(b) the segmented image using canny edge detection.



(a) (b)

Figure 4.1– (a) Original head phantom and (b) segmented images

The same algorithm was also applied to a 640 x 480 laparoscopic aorta video:

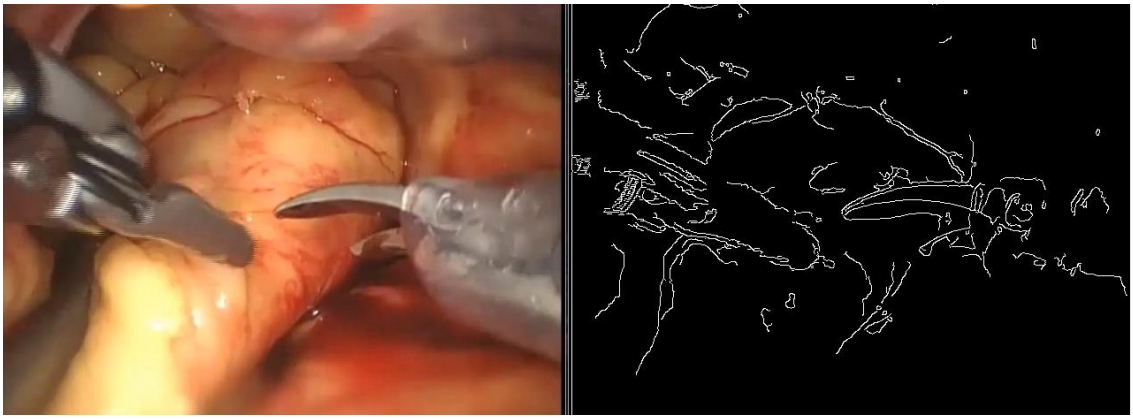


Figure 4.2– Original aorta laparo video and (left) segmented aorta from laparo video (right)

4.2.4.2 Thresholding

Thresholding was applied to the above aorta video:

Figure 4.3– Original aorta laparo video and (left) segmented threshold aorta (right)

4.3 Registration methods and results

4.3.1 Medical image registration

"Registration is the determination of a geometrical transformation that aligns points in one view of an object with corresponding points in another view of that object or another object." [68]. Medical image registration spatially aligns groups of images that can be acquired using different or same imaging modalities. For example: magnetic resonance imaging (MRI), computed tomography (CT) positron emission tomography (PET) or ultra-sound (US). Multi-modal imaging is challenging since the images may not have a lot of commonalities. Image registration can be defined as:

$$\hat{\mu} = \arg\min_{\mu} \mathcal{C}(I_F, I_M; \mu)$$

where I_F and I_M represent the moving and fixed images, μ vector of parameters that model the transformation. The cost function (C) consists of a comparison that defines quality of alignment. Optimization is usually gradient descent:

$$\mu_{k+1} = \mu_k - a_k \frac{\partial \mathcal{C}}{\partial \mu}$$

Merging these images allows physicians to visualize and compare structural and functional information. During surgery, organs may change shape or shift and pre-operative processing must be updated. This can be done using real-time image guidance and accurate registration of medical images. There is no unique registration algorithm that is suitable for all applications, thus the algorithms are tuned and adjusted tor specific

applications [49]. For a specific application, a balance between accuracy and computation time will lead to a suitable registration. For example, pre-operative processing and registration may take hours, while image guided surgery requires real-time or near real-time image registration. This intra-patient registration must provide reasonable alignment. Examples of different image registration methods include: Rigid (point based (Procrustes), surface based (Iterative Closest Point ICP), intensity based (Normalized Mutual Information NMI) and non-rigid (surface based, volume based and shape based) [46]. During non-rigid registration, images are spatially deformed to match each other.

The steps of registration include: image will be transformed to match or become more similar to another image, the similarity will be assessed, and finally the transformation will be optimized through iterations. Starting registration with lower complexity images will increase chances of successful registration. This can be done through smoothing or downsampling (resampling or shrinking) images. For example Gaussian filter: $\frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{x^2}{2\sigma^2}}$ will smooth an image in direction x. The same method can be used for y direction as well as x to achieve full smoothing.

Resampling can be done by looping over all pixels of the fixed image to calculate the mapped position, after that using interpolation to find the intensity of the moving image and copy to the output image. Several methods of interpolation include: nearest neighbor, linear and B-spline that differ in quality and speed.

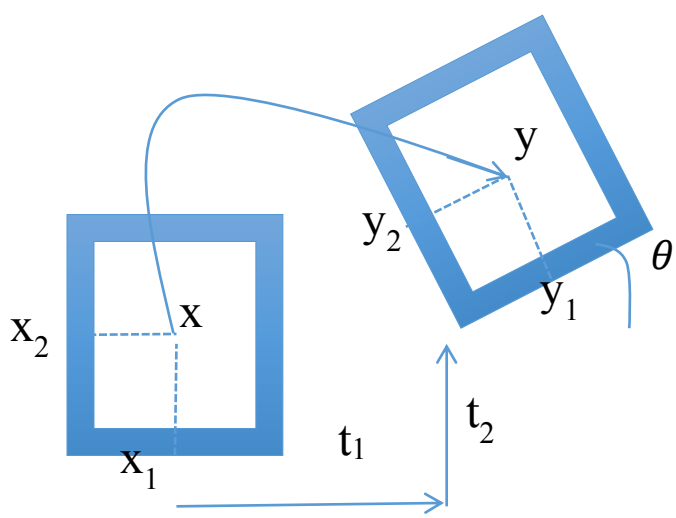
Transformation methods consist of rigid (moving or rotating object without changing shape), affine (same as rigid with additional scaling and shearing), non-rigid (more than 6 degrees of freedom and localized transformation). Measure of similarity can

be assessed using feature-based or intensity-based methods. Finally, optimizer will change the transformation model to obtain maximum similarity.

4.3.2 Rigid registration

Rigid registration moves an image in space (translate or rotate) but does not change the original shape. For three dimensional images, rigid transformation will have 6 degrees of freedom: 3 degrees of rotation and 3 degrees of translation. For example a 2-D rigid transformation with rotation θ and translation parameters t_1 and t_2 will result in:

$y_1 = \cos\theta \,.x_1 - \sin\theta.x_2 + t_1$
$y_2 = \sin\theta \,.x_1 + \cos\theta.x_2 + t_2$



A rotation along the z axis is equal to:

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.1.2 Affine registration

Affine registration consist of translation and rotation but also includes scaling and shearing parameters. An affine map is alignment of linear transformations with translations. Rigid and affine transformations are linear registration methods that can be described using a transformation matrix. The input image can be obtained by applying the reverse transform to the output image [14]. A 2-D affine transformation can be denoted by: T(x) = Dx + S, where D is a 2x2 matrix demonstrating rotation, scaling and shearing, S is 2x1 vector demonstrating translation or shift.

4.3.3 Non-rigid registration

Non-rigid registrations modify input images that will result in distortion in one or more dimensions. During non-rigid registration, one or several images are locally spatially deformed to match another image. The transformation model can be parametric or non-parametric. Parametric models use other transformation models to generate a deformation field. For non-parametric models each voxel has three degrees of freedom and regularization is used to smoothen deformation: $\vec{x}^{/} = u(\vec{x})$ (u is the deformation field). Similarity measures can be feature-based (landmarks such as points, lines or surfaces) or intensity-based. Intensity-based does not require pre-processing and it can be the sum of squared difference or absolute differences. For example the sum of squared difference between a reference image (R) and a deformed floating image F (T) is calculated using:

Two of the most common optimization methods are gradient descent and Gauss-Newton. During non-rigid registration the first and/or second derivatives of the objective function are necessary, since a large number of parameters require optimization [50].

4.3.4 Free-Form Deformation algorithm

One of the algorithms that we investigated is called Free-Form Deformation (FFD) [50]. The image domain consists of control points that translating these points will induce local deformations. The algorithm consists of three parts: transformation using cubic B-spline and an interpolation function, evaluation an objective function and optimization. The cubic B-spline guarantees a continuous deformation but it is the computationally expensive part of the algorithm. The equation to compute new coordinate of a point in one dimension is:

$$\mathbf{T}(\vec{x}) = \sum_{l=0}^{3} B_l \left(\frac{\vec{x}}{\delta} - \lfloor \frac{\vec{x}}{\delta} \rfloor \right) \mu_{i+l}$$

where μ is the first control point position, δ is the distance between two control points, and B functions are approximated three-order spline polynomials. In three dimensions the Free-Form Deformation can be written as 3-D tensor product of the 1-D cubic B-spline equation:

$$\mathbf{T}(\vec{x}) = \sum_{l=0}^{3} \sum_{m=0}^{3} \sum_{n=0}^{3} B_l(u) B_m(v) B_n(w) \mu_{i+l,j+m,k+n}$$

where u, v and w are relative positions of the index point along axis x, u and z. i, j and k

are the indices of the first control point and as a result the new coordinates are calculated from $4 \times 4 \times 4$ (64) surrounding control points. These 64 control points are used to compute one coordinate. For a 3-D image, each voxel has three coordinates so 192 degrees of freedom have to be checked. The higher number of control points results in better registration quality but also increases the processing time. As a result, an arbitrary cubic B-spline deformation can be used to refine a deformation by adding new control points at each half-spacing length [51]. As a result, the spacing will be divided by two for each axis. This will increase the control points by a factor of eight and closer to their optimal position with less iterations.

The three parts of the FFD algorithm can be processed independently. FPGA-based and supercomputer with 64 CPUs have been used to speed up this computation 3.2 and 50 times respectively [52, 53]. More recently, GPU based implementation of the NiftyReg package [54] has reduced the processing time and led to the speed up of 10 to 20 versus single-thread CPU implementation [55]. We have tested registration of brain MRI and abdominal (kidney) CT scans. The results are presented and discussed in the next two sections. We will discuss the GPU implementation in the next chapter.

4.3.5 Registration results

4.3.5.1 Brain MRI registration

Affine and non-rigid transformations were first tested on brain MRIs using NiftyReg. Figure 4.4 illustrates brain scans taken a year apart from a patient with Alzheimer's disease and the difference of these images prior to registration. The data set was downloaded from: https://www.ucl.ac.uk/drc/research/methods/miriad-scan-database

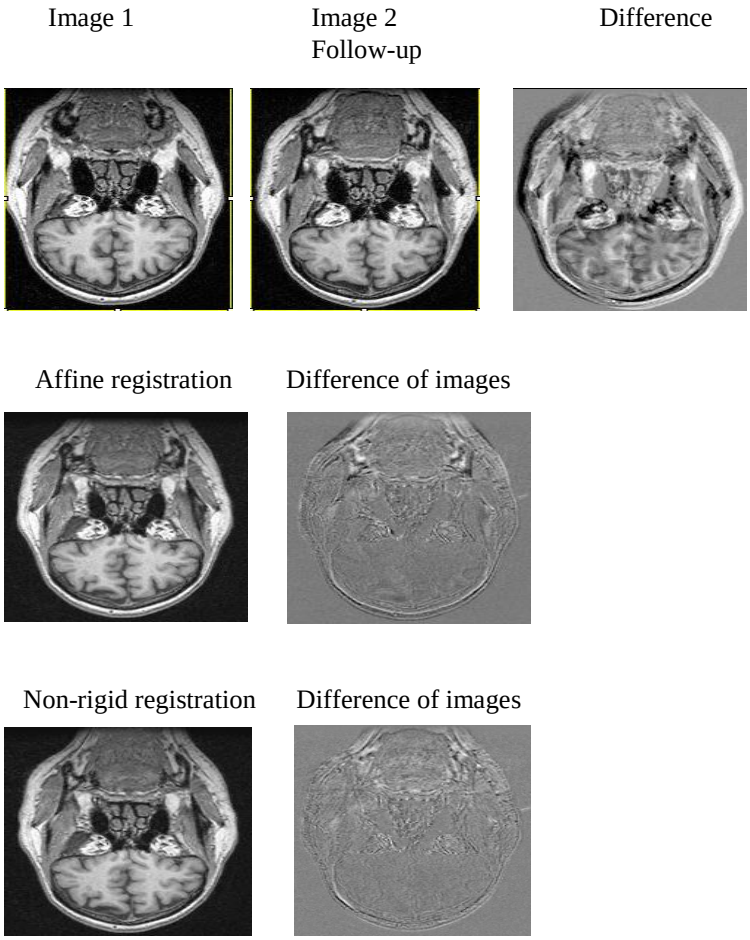


Figure 4.4– Brain MRI images with affine and non-rigid registrations

4.3.5.2 Abdominal registration for kidney

The transformations were then performed on abdominal CTs. Figure 4.5 illustrates CT scans taken 16 months apart from a patient with prostate cancer to examine his kidney. The data set was obtained from: http://www.cancerimagingarchive.net

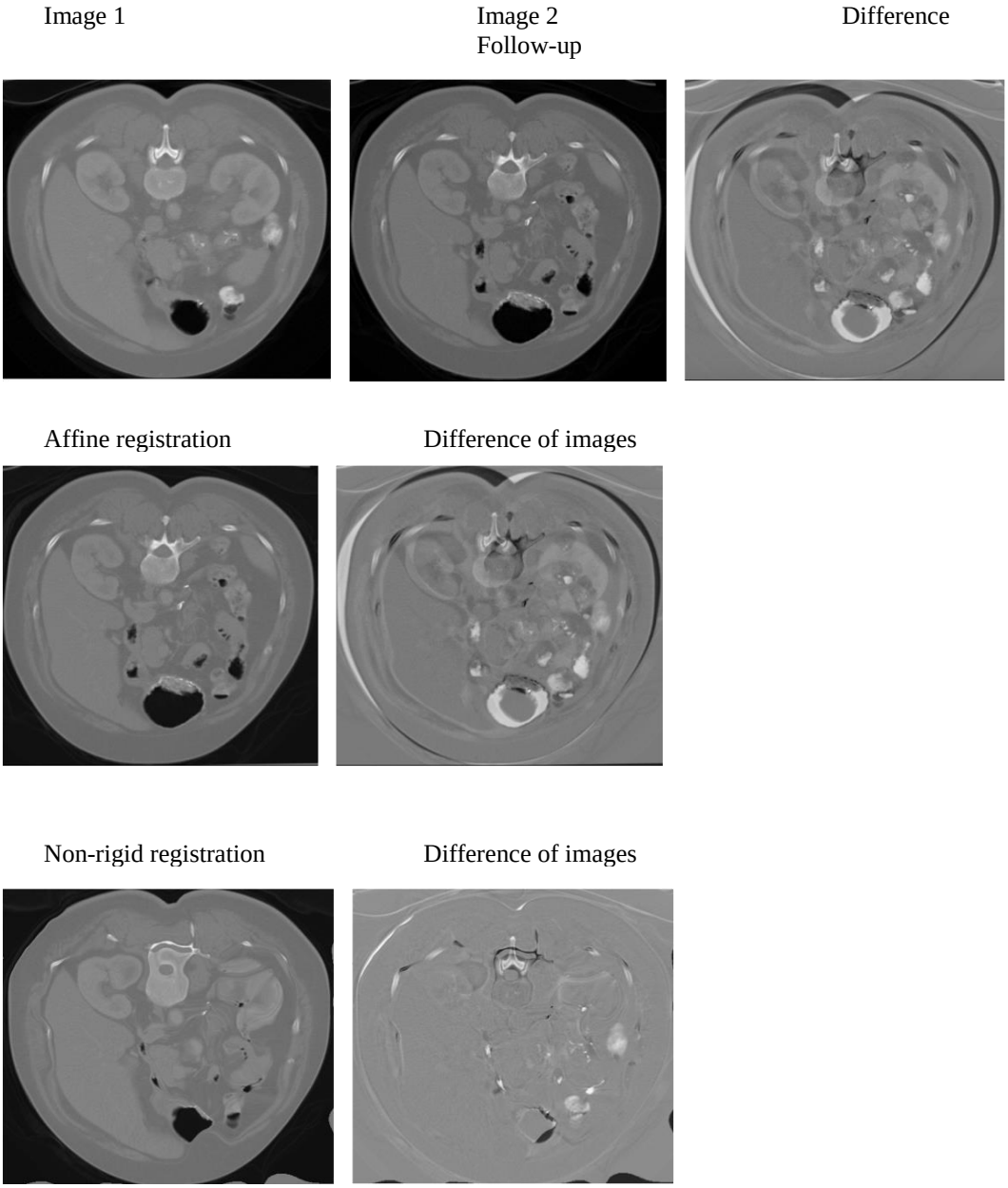


Figure 4.5– - Abdominal CT scans with affine and non-rigid registration

3.2.3 Laparoscopic video registration

Image 1 | Image 2 Follow-up | Difference

Non-rigid registration | Difference of images

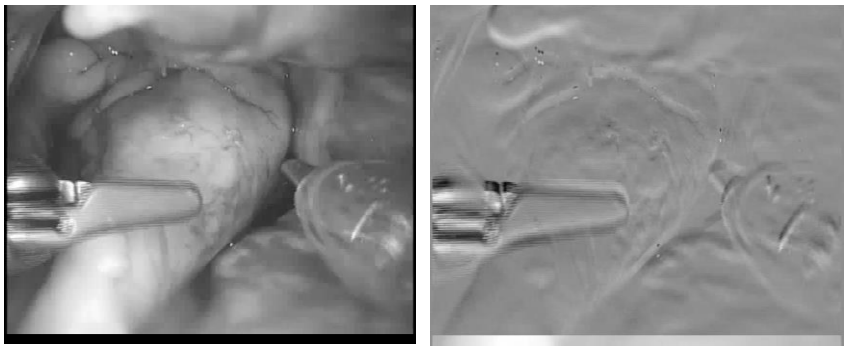


Figure 4.6– - Aorta with affine and non-rigid registration

4.4 Discussion and Conclusions

In the field of computer assisted-surgery, particularly in IGS, the ability to acquire, process, render, and visualize data in real time is essential for the performance of complex tasks, thus minimizing risk to the patient [29]. The contextual validity of data degrades quickly as tools within the body move, patient organs move, and resections are performed. It is critical that the data be acquired, processed, and returned in a timely manner [44].

Processing of live high-definition stereo laparoscopic data with preoperative or intraoperative medical images requires more computational and storage capability than is possible on a single workstation. More advanced visualizations and processing may require new or improved rendering techniques more suited to an HPC environment. Therefore, flexibility in the transmission of data to multiple sites is important. Furthermore, the ability to optimize the data transmission to take advantage of the specialized processing and rendering techniques at multiple endpoints is the key to allowing for rendering and processing as a service and also for providing optimal data.

Through the use of HPC resource, we believe that it will be possible to update models and allow for real-time modification of the previously generated object map, based on current surgical data. Optimization and parallelization of these algorithms will reduce the total processing time. We also plan to develop segmentation and registration algorithms, and send previously generated data to the cluster for adaption and deformation. Success in this portion would lead to the ability of performing surgery at one site (origin), receiving an object map from another site (segmentation/rendering), forwarding this data along with relevant force data to a third site for deformation (model update), and returning this data to the origin for display.

Chapter 5

HIGH PERFORMANCE COMPUTING ENABLED REAL-TIME PROCESSING OF LAPAROSCOPIC SURGERY

Published as: Ronaghi, Z., Sapra, K., Izard, R., Duffy, E., Smith, M. C., Wang, K. C., & Kwartowitz, D. M. (2016, March). HPC enabled real-time remote processing of laparoscopic surgery. In SPIE Medical Imaging (pp. 97861U-97861U). International Society for Optics and Photonics.

5.1 Introduction

Minimally invasive surgery (MIS) reduces patient trauma, hospitalization stay and recovery time through reducing incisions to small keyholes. Laparoscopic procedures use a camera for real-time visualization of the surgical field and guidance of the procedures. However, small field of view of the laparoscope as well as small incision size will result in a small overall visual field of the underlying tissues of interest. Image-guided surgery (IGS) uses images to map a surgical region of interest, thus providing surgeons with visualization of subsurface structures and accurate positional information during a surgical procedure [7]. The accurate co-registration of an IGS system with laparoscopic video can reduce the impact of limited surgical access and allow for a resection with higher specificity and tighter margins, thus sparing more healthy tissues [10].

Currently the primary robotic surgery devices on the market are the daVinci-si and daVinci-xi systems. The daVinci-si robot uses two parallel 1080p (1920x1080x3) High Definition video cameras to generate stereoscopic vision [12]. Robotic surgery uses stereo laparoscopy (one feed for left eye and one for right eye), thus the volume of data collected and required throughput is twice that of traditional laparoscopy. If captured raw, these video streams generate approximately 360 megabytes of data per second demonstrating a trend towards increased data sizes in medicine. Real-time processing this large stream of data on a bedside PC, single or dual node setup, has become challenging and a high-performance computing (HPC) environment may not always be available at the point of care. Processing of the volume of data generated by the stereo video feeds would be demanding for a local system, and thus HPC hardware is needed [14].

We have implemented and compared performance of compression, segmentation and registration algorithms on Clemson's Palmetto supercomputer using dual NVIDIA K40 GPUs per node. We implemented our algorithms on CPU and GPU to analyze execution time. Different size and resolution of images were used to compare run-time and overall performance.

5.2 Methods

5.2.1 Medical image processing on GPUs

Medical image processing, specifically medical image registration is a computationally expensive task. The computation of parallel friendly algorithms

(processing of blocks of data are performed in parallel) can be distributed over processors. Thus, multicore architectures could increase processing speed of these algorithms and process large amount of data in shorter time. For example, Graphic processors were designed for visual rendering of games and their computational power to cost ratio has made them popular for various computational tasks and applications. Graphic Processing Units (GPUs) have a large number of cores that can be used to reduce time and increase speed of image processing algorithms [16]. Algorithms that are data parallel can take advantage of the data parallel architecture of GPUs and task parallel algorithms are more suitable for CPUs. Until early 2000s GPUs were mainly used for programming graphics applications. Since 2003 codes for general-purpose applications such as molecular dynamics, data mining and signal processing have started taking advantage of GPUs.

One of the differences of CPUs and GPUs is the number of Arithmetic Logic Units (ALUs). The figure below from Nvidia's website [58] shows a higher number of ALUs for GPUs compared to CPUs for processors with the same size. As a result, GPUs are able to execute thousands of operations concurrently while multi-core CPUs can handle 2 to 16 simultaneously. For example, Nvidia's Tesla K40 provides 1.43 Tflops. However, GPUs may not follow the same accuracy standards of CPUs. This could results in accumulation of errors, thus verification of results using CPUs is recommended [56].

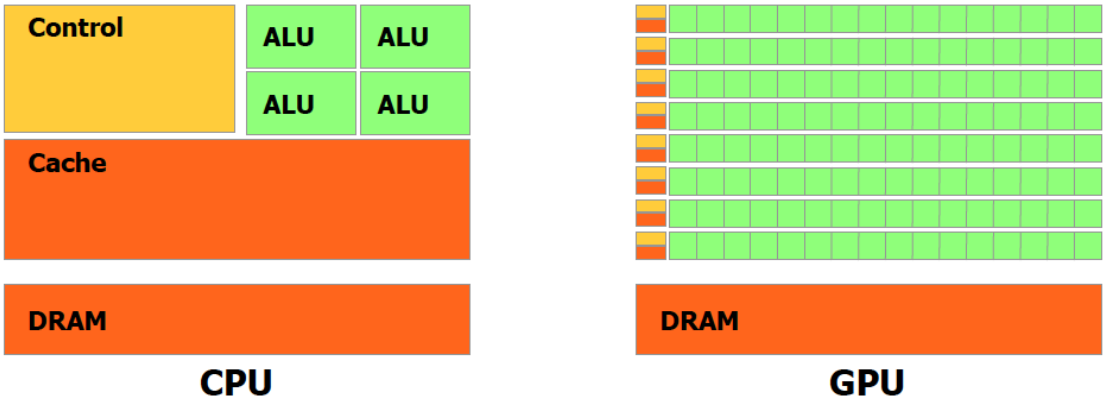


Figure 5.1– CPU vs. GPU Arithmetic Logic Units [57]

Compute **U**nified **D**evice **A**rchitecture (CUDA) and **Open C**omputing **L**anguage (OpenCL) are two of the programming interfaces that allow parallel programming on GPUs, which are described in the following sections:

5.2.1.1 CUDA

Compute Unified Device Architecture (CUDA) [58], was introduced by Nvidia in late 2006 and is specific to Nvidia hardware. It is an extension of C language that integrates CPU and GPU code. GPU acts as a coprocessor but with its own memory. CUDA enables communication and data transfer between CPU and GPU, which are called host and device. The function running on a device is called a kernel and consists of threads that can run simultaneously. The figure below indicates how a cuda program is executed.

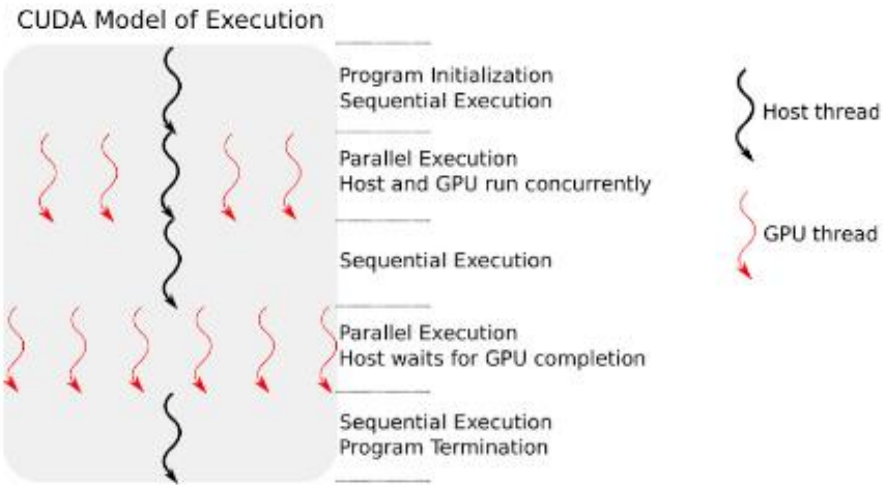


Figure 5.2 – CUDA execution model [56]

CUDA's architecture includes grids and blocks. This architecture allows CUDA code to be scalable and run on compatible hardware without recompilation. Grids (one, two or three dimensions) contain blocks and blocks (one, two or three dimensions) contain threads. Threads are also grouped in wraps that can be executed in unpredictable order. The size of each block (number of threads in the block) and grid (number of blocks in a grid) are defined using blockDim and gridDim and are set by the programmer. The position of thread within a block and block within a grid are indexed using threadIdx and blockIdx respectively. For example a thread's x position within grid of data can be calculated as: x = threadIdx.x + blockIdx.x * blockDim.x

The sizes of blocks and grids are determined by the device specifications and can be discovered by running "deviceQuery". Figure 5.3 indicates a 4x3x2 grid made of 3x2 blocks.

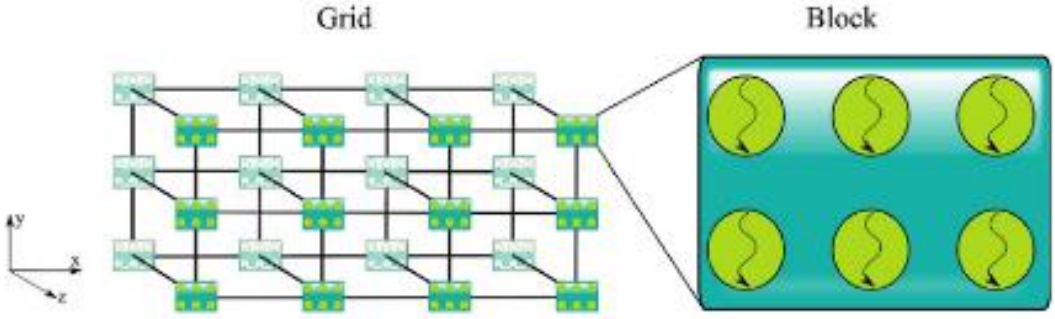


Figure 5.3 – CUDA Grid/block hierarchy [56]

GPU and host memory are typically disjoint and passing data through pointer to array in host memory is not possible. The GPU implementation of an algorithm can be optimized through memory access within a thread. Data will be transferred from host to device and copied back from device to host after kernel execution and computation. This data transfer is costly and should be minimized to maximize efficiency. CPU can only access global, constant and texture memory. Other memory types of the GPU include shared, registers and local memory. These memory types can be effectively incorporated in kernel structures to extract more performance from GPUs. The following figure shows CUDA memory model.

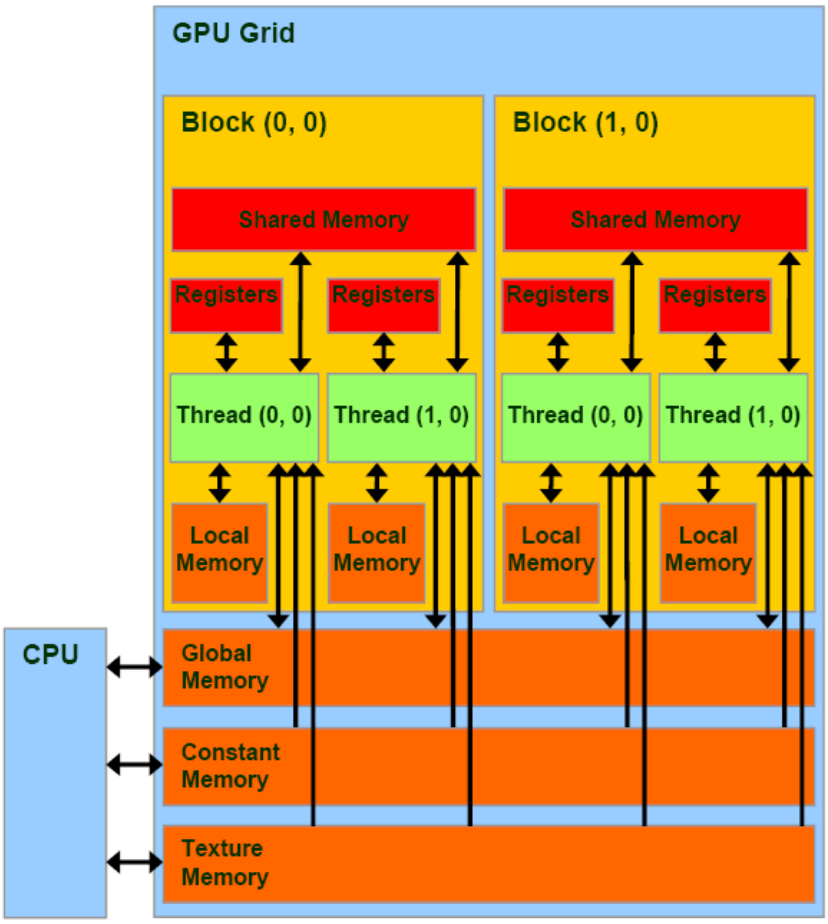


Figure 5.4 – CUDA memory model [58]

Registers are fast and individual threads have access to them. However, they are limited and usually automatic variables are registers. Shared memory is slower than registers but is accessible to all threads in thread block. They are commonly used for thread collaboration and synchronization. Constant memory is often used to provide input values to the kernel and is read-only. Texture memory is also read-only and often used to provide input values to the kernel and store textures. Local memory is local to a thread and resides in global memory. Global memory has a large reservoir but is very slow. It is visible to all threads in a grid and often used to pass information from one kernel to another.

We implemented our framework on Palmetto Supercomputer at Clemson University using two NVIDIA Tesla K40 GPUs per node, CUDA7.5 enabled with Intel Xeon-E5 CPUs and 32GB of Memory. Some of the specifications of K40 GPUs are 4.29 TF peak single precision, 1.43 TF peak double precision, 12 GB memory, 288 GB/s memory bandwidth and 2880 cores. Additional details on Palmetto cluster can be found in [39].

5.2.1.2 OpenCL

Open Computing Language (OpenCL) was introduced at the end of 2008 [59], developed by the Khronos group and is vendor independent. OpenCL is a general-purpose programming standard in heterogeneous systems that can run on different architectures, such as CPUs, GPUs and FPGAs. One of the main advantages of OpenCL is its portability. This enables connection of different devices to a host that run different parts of the code. It also allows efficient use of both CPU and GPUs and can be used for applications that compute parallel sections on GPU and sequential on CPUs. However, depending on the application it may not perform as well as CUDA on Nvidia's GPUs [60]. The device executes a kernel, which consists of points that execute an instance of the kernel. The instance is called a work-item (threads in CUDA) and is defined by points in the index space, also known as global IDs. Work-items are combined into work-groups. The index space that work-items operate is called NDRange, which is an N-dimensional index space (one, two or three).

As shown in figure 5.5, the memory model of OpenCL consists of global, local, constant and private memory. Global memory can be accessed by every work-item but it is the slowest to access. Local memory is faster but is only accessible for work-items within the same work-group. Constant memory is read-only but has low latency and can be accessed by all work-items. Private memory is the fastest memory but is accessed by a specific work-item.

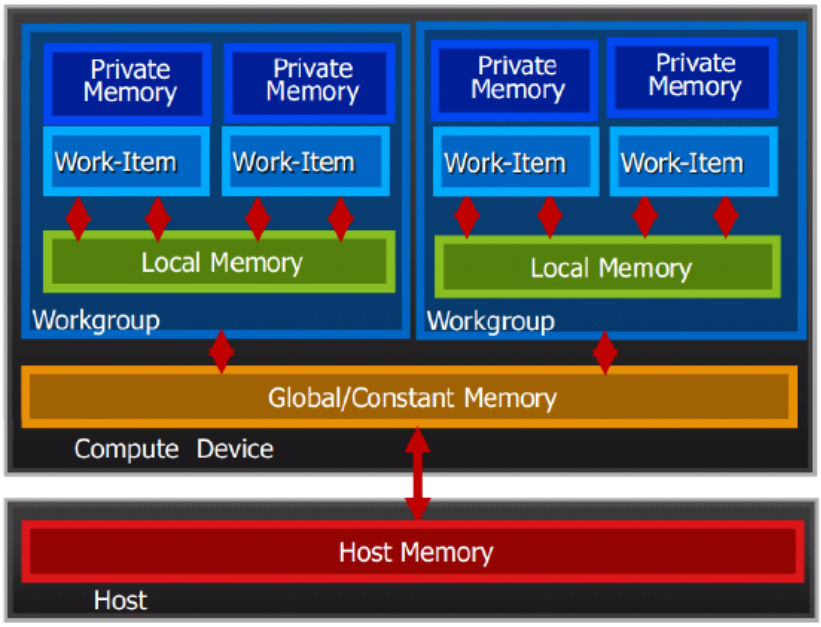


Figure 5.5 –OpenCL memory model [61]

OpenCL can be programmed through data parallel or task parallel models or a combination of both approaches. In data parallel programming model, each work-item is mapped to a data element to ensure aligned memory access. In task parallel, a compute unit creates one work-group with a work-item. This method is usually used to que multiple kernels or to efficiently implement vector data types. Most image processing algorithms

use the data parallel approach since it is difficult to take advantage of parallelism using the task parallel model.

5.3 Results and Discussion

Pre-surgery, our setup allows for transfer of per-operative images from a remote location to compute nodes and GPU memory. OpenCV, ITK and Elastix provide a GPU module for basid image processing algorithms. We perform warm-up of the networking protocol, followed by verification of compute nodes capabilities and devices.

The compute nodes are broken into three layers: Main Compute Layer (ML) and the two Duplication Compute Layer (DL). Each layer contains a node that performs duplicate computation. This computation is then verified with a quick checksum followed by verification using NVIDIA GPU Direct with compute nodes in the two DL. If the ML matches any one of DL calculation, the ML overlay information is sent back to surgeon. In case of a failure of GPU, the computation between the two DL are verified and sent to the surgeon and an idle node in the same layer overtakes the computation task of the failed compute node. This method enables reliability using replication of computation.

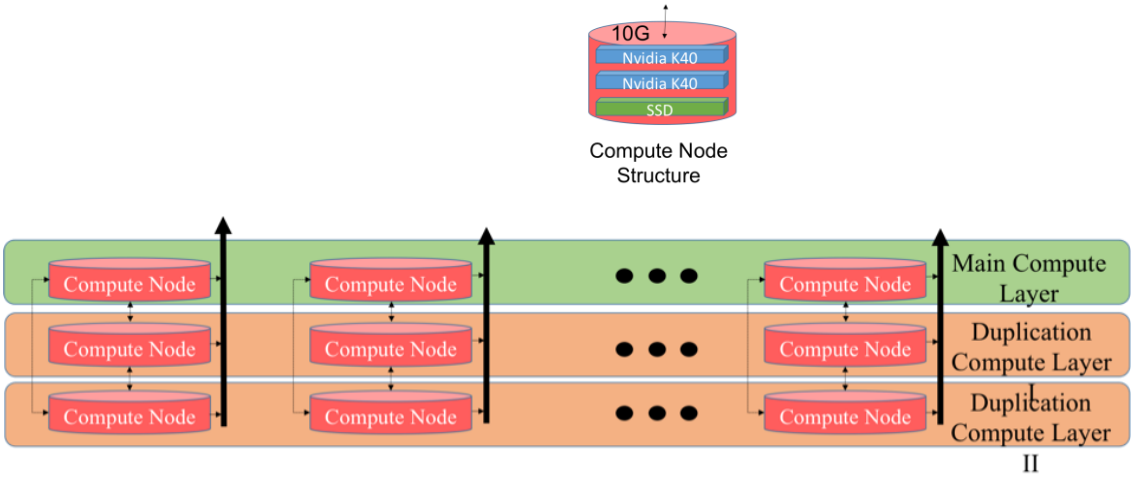


Figure 5.6 –Compute node structure

The following graphs compare CPU and GPU matrix multiplication using CUDA. Graph 1 indicates that the execution time for both CPU and GPU increase as the matrix dimensions increase. Adding more elements to the matrix requires more processing time. Graph 1 shows the results using different scales for CPU and GPU to observe the values and compare the graphs. Graph 2, indicates the speed up using GPU compared to serial implementation. Matrix sizes for testing include W = 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096. The benefit of using parallel implementation is more noticeable as the dimensions increase. For lower values (W=8, 16, 32) the GPU time is more than CPU. This is due to the communication time between host to device and device to host. As the matrix dimensions increase (W = 64, 128, 256, 512, 1024, 2048, 4096), the GPU version requires less time compared to serial. The calculation time on CPU is more than communication between host and device; as a result the overall GPU time is less than CPU.

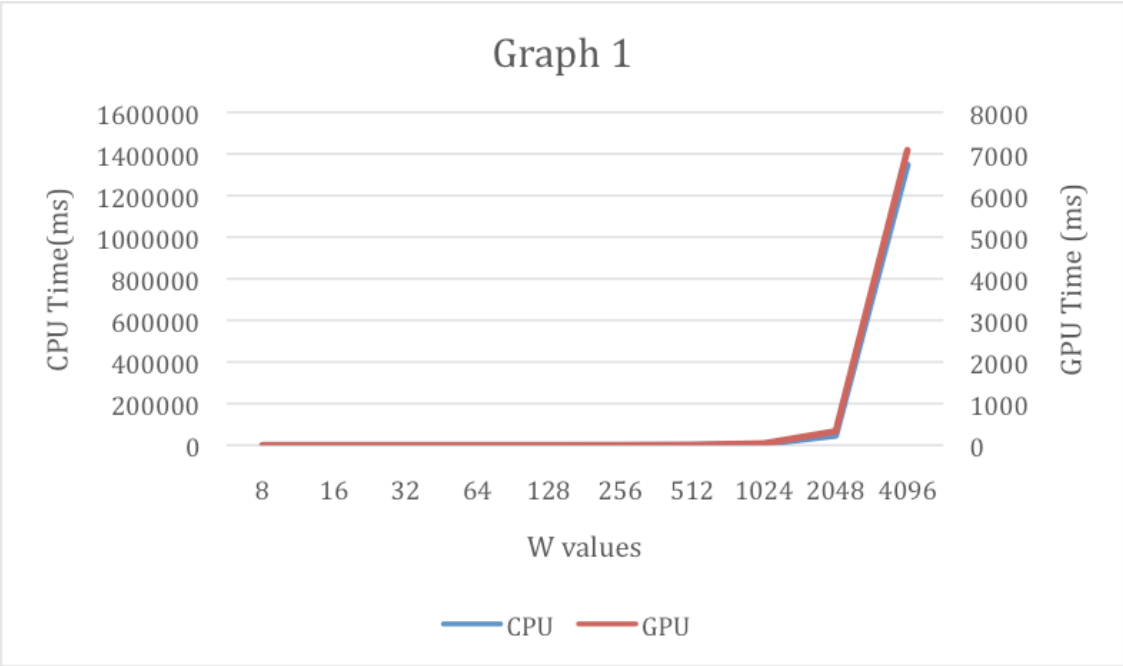


Figure 5.7 –Matrix multiplication time using CPU and GPU

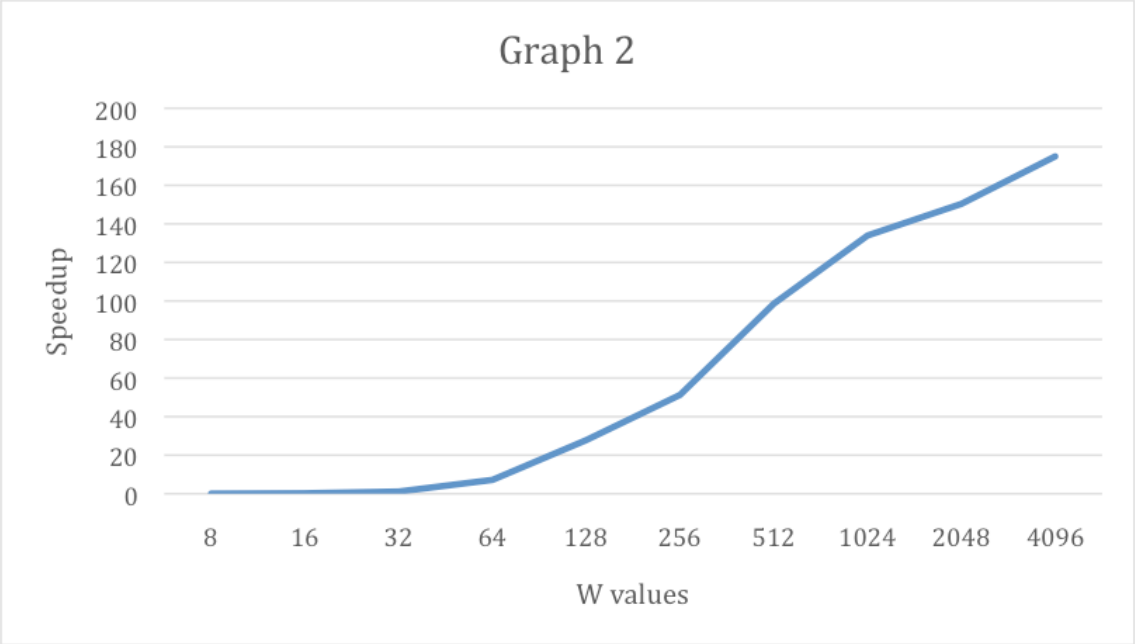


Figure 5.8 –GPU speed up compared to CPU

For larger matrices, i.e. W= 4096, kernel parameters (block and grid size) influence the overall performance. Matrix size = WxW; Block size = AxB; Grid size = CxD; where

C = W/A and D=W/B. The values do not show a significant difference for small dimensions. These values can change as the matrix sizes increase. Using more threads within a block will lower the occupancy. However, the number of threads is limited and dependent on the GPU model. If the input is more than the thread value, it can be divided into blocks. Table 1 shows matrix multiplication time for W = 4096:

Table 5.1 - Matrix multiplication time for W=4096

W = 4096	Time(ms)
2-D Block 2-D Grid: Block = 2x2, Grid = 1024x1024	30485
2-D Block 2-D Grid: Block = 4x4, Grid = 256x256	9217
2-D Block 2-D Grid: Block = 8x8, Grid = 512x512	2624
2-D Block 2-D Grid: Block = 16x16, Grid = 256x256	1619
2-D Block 2-D Grid: Block = 32x32, Grid = 128x128	1481

5.3.1 Image Processing using GPUs

5.3.1.1 GPU based registration

As discussed in chapter 4, image registration consists of reading and setting up images, pyramid construction, iterative computation to update parameter vector and resampling. Pyramid construction and iterative computation dominate the performance (profiled using valgrind) and were transferred to GPUs for parallel computation using Elastix and ITK [62].

During pyramid construction, images are smoothed and down sampled (resampling or shrinking) to reduce image complexity. The Gaussian pyramid is most commonly used for image registration and smooths images in a single direction. Gaussian filer in 1-D: (σ is standard deviation of the distribution): $G_{1D}(\mathcal{X}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$ [46]. Gaussian filter operates row-by-row for x direction or column-by-column for y. Each row can be assigned to a different thread for parallel computation. However, the column kernel will be queued and processed after row kernel since columns can be processed after rows.

Resampling is usually done using the fixed image domain. Mapped position of y = T(x) is calculated using voxels inside the fixed image domain (x). Moving image intensity is calculated using interpolation and copied to the output image [46].

We have tested affine and non-rigid transformation and the GPU implementation for non-rigid. Transformation and interpolation are performed in separate kernels and sequentially scheduled on the GPU.

We compared the run-time for image size of 1920x1080 and 640x360:

Table 5.2 - Affine and non-rigid registration

	640x480	1920x1080
Affine	21ms	129ms
Non-rigid	49ms	338ms
Non-rigid GPU	21ms	38ms

We have also investigated NiftyReg library and its CUDA implementation. Using the Nvidia visual profiler we observed the following results and compute percentages:

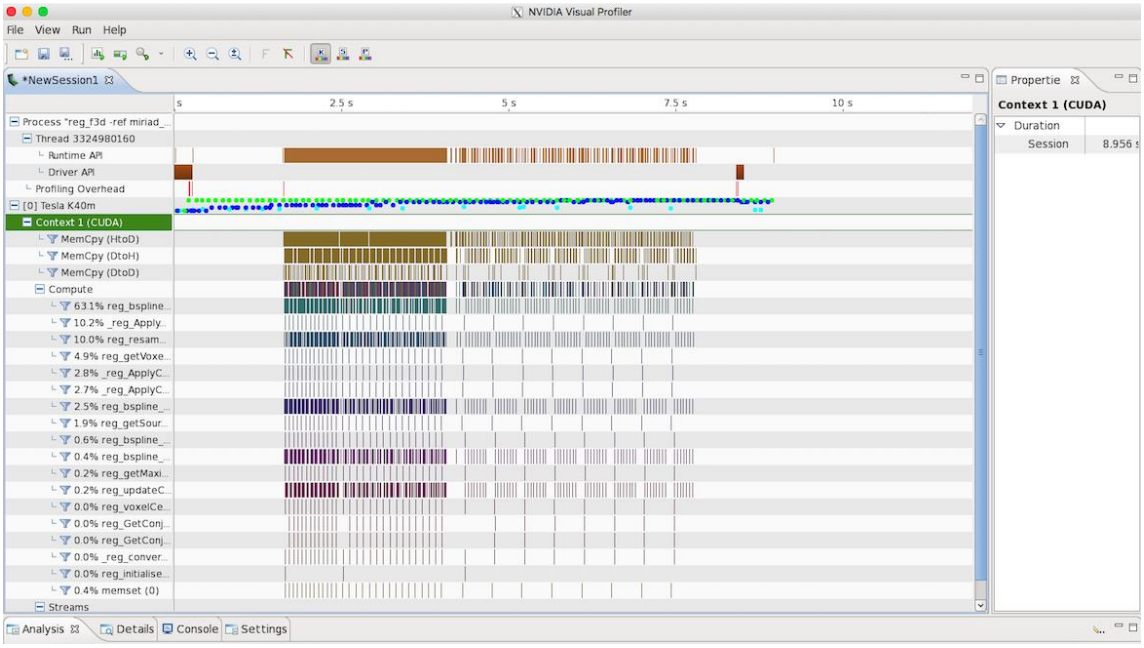


Figure 5.9 –Nvidia visual profiler

It can be observed that B-spline is the most computationally expensive part of this algorithm (as mentioned in chapter 4). The CUDA implementation was based on independent voxel displacement and intensity interpolation of the algorithm [53].

Optimization can be made by adjusting the kernels to use thread level parallelism (TLP) and instruction level parallelism (ILP). TLP can be achieved by selecting the appropriate block size for a specific application. ILP is parallelism among independent instructions and is dependent on the application or problem being solved. The optimization methods that we used include: static branch precalculation, loop unrolling and instruction reordering. For example the ApplyConvolution function includes a loop that was optimized

by precalculating and presetting windows sizes. Loop unrolling was applied to the main loop and the speed up was around 1.3 compared to the original implementation.

Another step of the optimization includes setting fixed image sizes, since these values are predetermined and stay fixed throughout a surgery or pre-operative image processing. Another limiting factor is the communication between CPU and GPU. Reducing data fetching and number of communications between CPU and GPU can speed up the algorithm. Uploading the pre-operative images to GPU memory prior to computation can reduce the communication time.

The following images show registration of Brian MRI scans using free form deformation with and without GPU.

Image 1

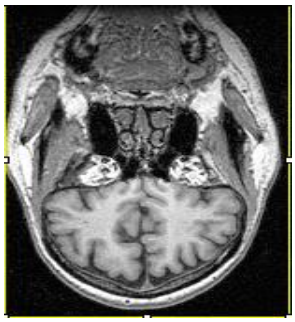
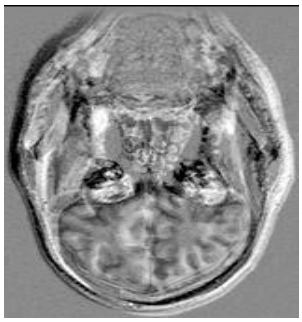
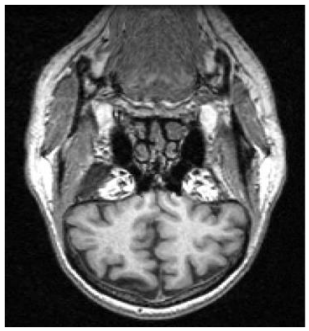


Image 2
Follow-up

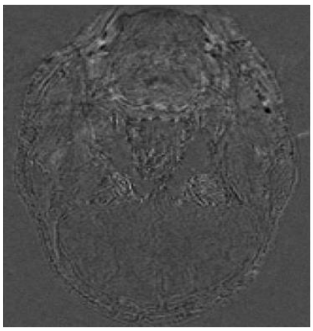
Difference



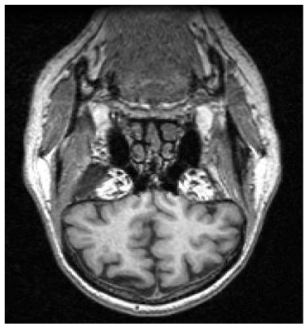
Non-rigid registration



Difference of images



Non-rigid registration with GPU



Difference of images

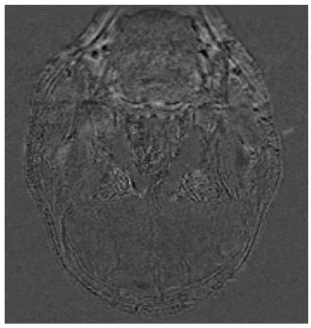


Figure 5.10 –Brain MRI registration

The images below indicate registration of abdominal CT scans using free form deformation with and without GPU.

Image 1

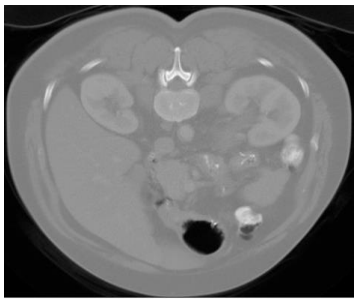
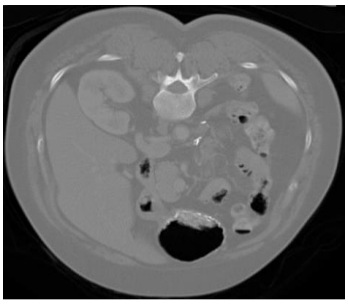
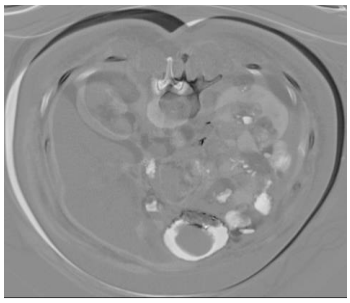


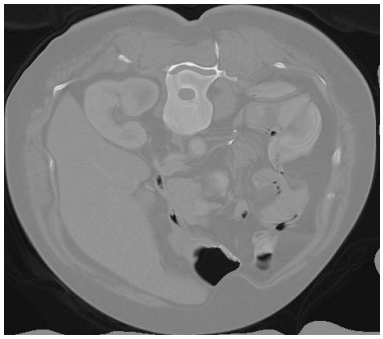
Image 2
Follow-up



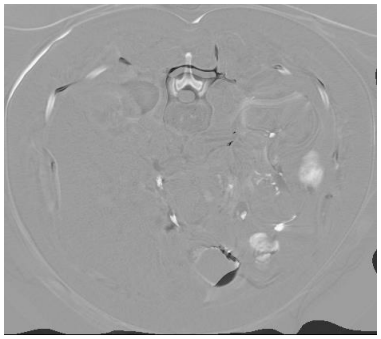
Difference



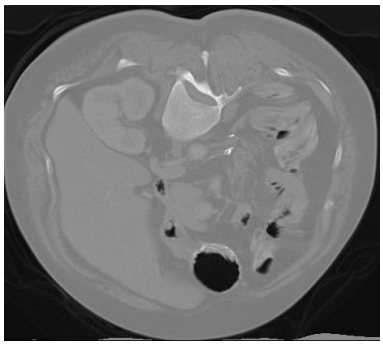
Non-rigid registration



Difference of images



Non-rigid registration with GPU



Difference of images

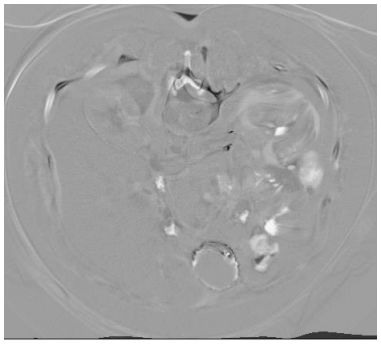


Figure 5.11 –Abdominal CT registration

5.3.1.2 GPU based segmentation

Thresholding and canny edge segmentations were transferred to the GPU for parallel computation using Open Source Computer Vision (OpenCV) [70]. The image mask was also constructed during this step. Thresholding is embarrassingly parallel since each pixel can be processed independently of others and there is no need for synchronization [23]. The number of threads is equal to the number of pixels and memory is only needed to store the segmentation result. For example, for single threshold T, the algorithm for kernel:

```
if   image ( threadIdx, threadIdy) ≥ T then
     result (threadIdx, threadIdy) ← 1
else
     result (threadIdx, threadIdy) ← 0
end if
```

For canny edge detection, Gaussian filtering, non-maximum suppression and hysteresis thresholding are performed in parallel for individual pixels. During Gaussian filtering the image is divided into subimages and after parallel computation the final gradient strength and direction are stored in memory. Non-maximum suppression uses these values to find the pixels (threads) that are local maximum and are stored in memory to be used as edge pixels. For hysteresis thresholding, each pixel in a subimage is processed to check whether a pixel is a strong edge pixel. If not, the thread pushes them to a stack

and the same operation is performed on each pixel. This step will continue until the stack becomes empty [63]. This process is shown in figure:

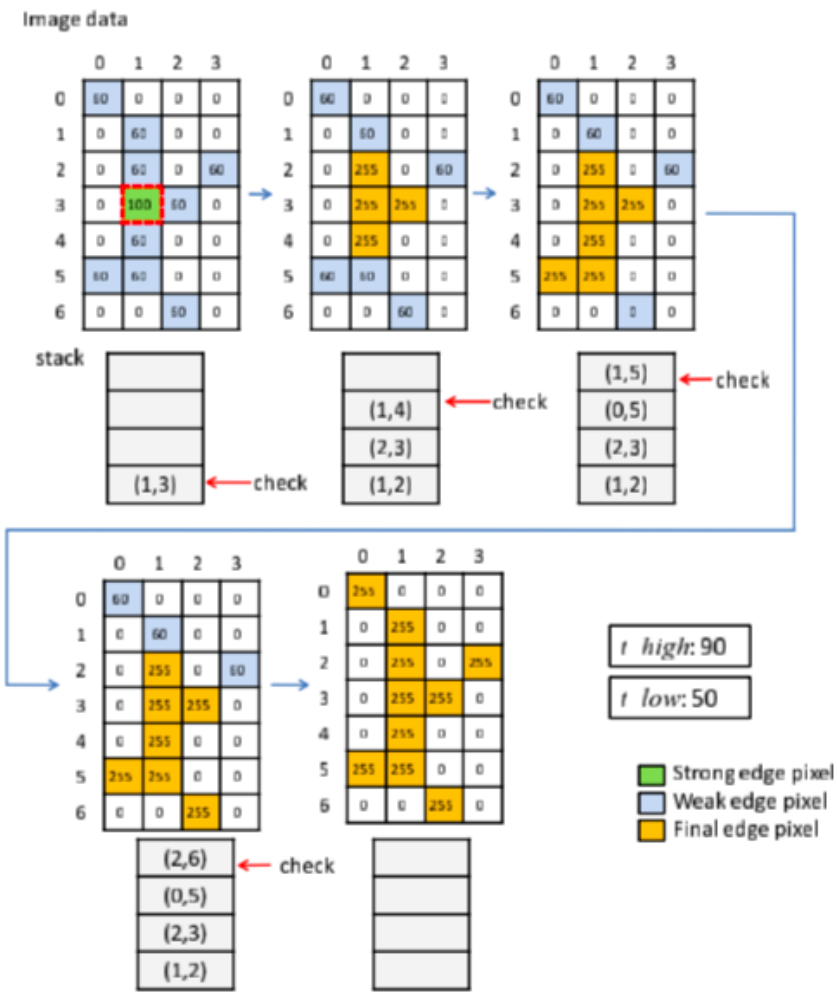


Figure 5.12 – Hysteresis thresholding [11]

The table below indicates segmentation time on CPU and GPU for different image sizes:

Table 5.3- Segmentation on CPU vs GPU

Segmentation	640x480	1920x1080
CPU	3ms	22ms
GPU	3ms	14ms

Figure 5.13 shows run-time for HD images time of various algorithm of interest to the surgeon in CUDA. As shown in the figure some algorithms have slower frame rates. To achieve a higher frame rate, we can enable GPU compression or image resizing to reduce image resolution prior to the operation.

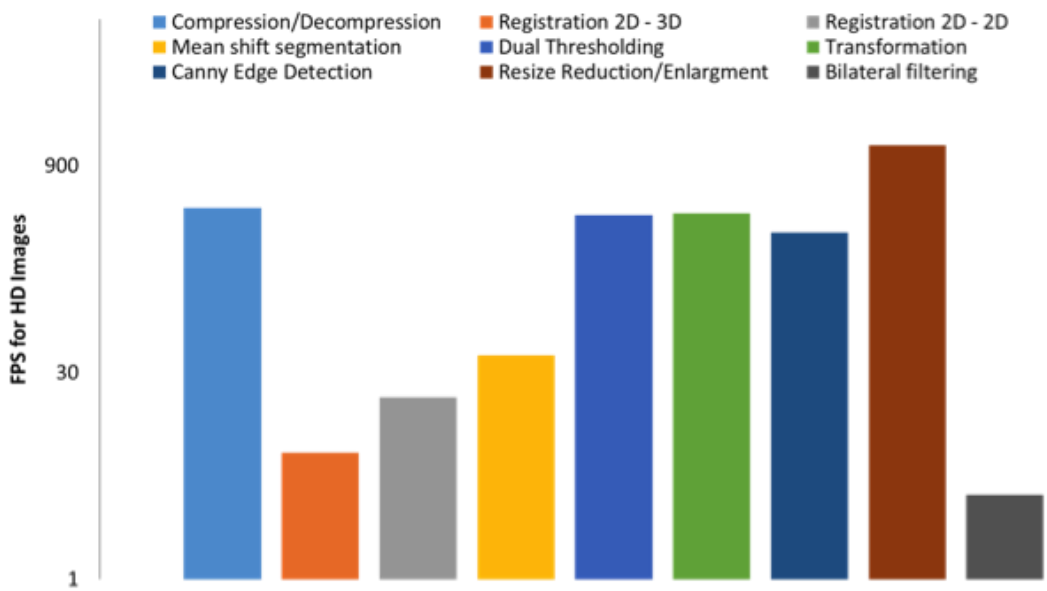


Figure 5.13 – GPU run-time of image processing algorithms

Different applications that benefit from image compression include: medical imaging, remote sensing, televideo conferencing, flash memory and disk storage [23]. For example the amount of data in a 30 frame per second (fps) HD movie (1920 x 1080 x 3byte) is equal to:

$$30\ \frac{frames}{second} \text{ x } (1920 \text{ x } 1080)\ \frac{pixels}{frame} \text{ x } (3 \text{ x } 8)\ \frac{bits}{pixel} = 1492992000\ \frac{bits}{second} = 1.49 \text{ Gb/s } (\frac{gigabits}{second})$$

Three types of data redundancies for two dimensional intensity arrays include: coding redundancy, spatial and temporal redundancy, and irrelevant information. Reducing or eliminating one or more redundancy factors will compress images. The image compression steps include:

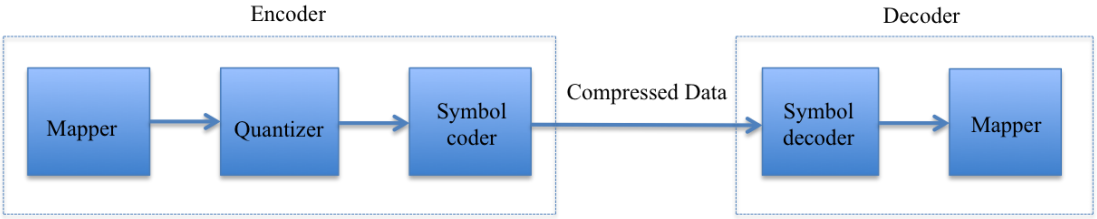


Figure 5.14 – Image compression

In [64] authors discuss parallel processing of image compression. This method can be used to reduce compression time and as a result speed up transmission and image processing. We measured the time that is required time to compress images. In order to reduce and resize to original resolution using GPUs took only 0.8msec. Furthermore if required, we can also enable compression of images, allowing 1.3msec to perform compression and decompression to achieve a higher frame rate. Figure 5.15 shows the performance of these algorithms. As seen in the figure certain algorithms such as bilateral filtering, mean-shift segmentation can achieve strong-scaling gain. However other operations require reduction in image size does not improve the performance.

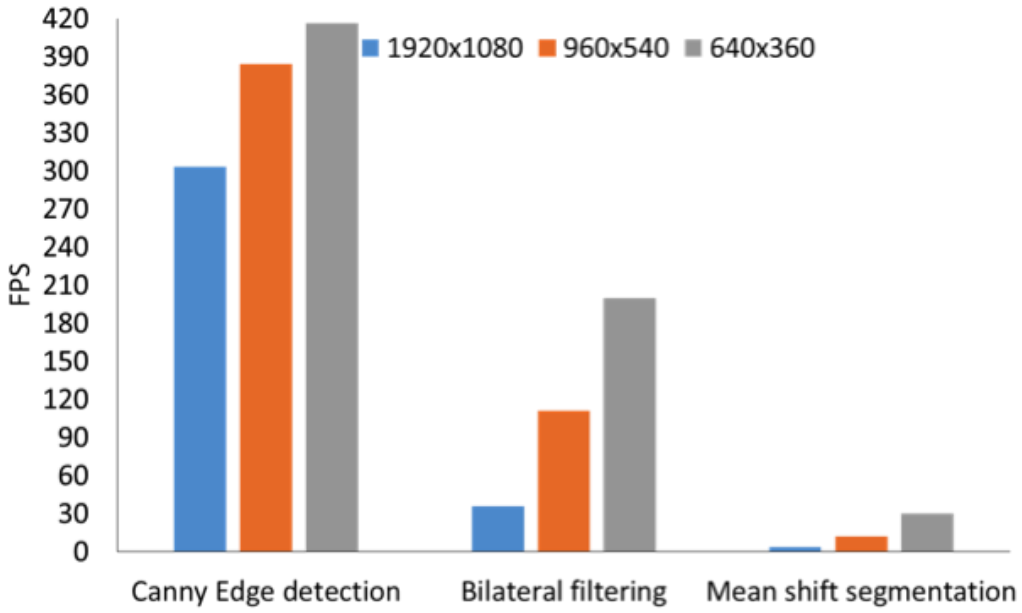


Figure 5.15 – GPU Run-time for varying image sizes

5.4 Conclusions

Processing of live high definition laparoscopic data with medical images requires more computational and storage capability than is possible on a single workstation. More advanced visualizations and processing may require new or improved rendering techniques more suited to a high-performance computing environment. Therefore, the ability to optimize the data transmission to take advantage of the specialized processing and rendering techniques at multiple endpoints is the key to allowing for rendering and processing as a service and also providing optimal data. Through the use of high-speed access to a high-performance computing resource, we believe that it will be possible to update models and allow for real-time modification of the previously generated object map, based on current surgical data.

We present a framework that enables reliable and secure processing of terabytes of information to aid in laparoscopic surgery using GPUs. The output of our framework is an overlay that can be enabled or disabled by the surgeon at the surgical console. Furthermore, our framework enables replication to prevent failure and allow for scalability for additional operations/algorithms. We plan to perform our segmentation algorithm for the laparoscopy video on the local multicore processors computer and compare the results with the algorithm running on HPC cluster. Through the development of a direct connection we hope to achieve more direct control over of the transmission of data as well as be able to develop a communication system between clients and computing nodes to perform real-time medical image processing.

Chapter 6

MPI-CUDA IMPLEMENTATION FOR REAL-TIME MEDICAL IMAGE PROCESSING

6.1 Introduction

Image-guided surgery (IGS) can provide surgeons with additional information during minimally invasive surgical procedures. The benefit of small incisions has a disadvantage of limited visualization of subsurface tissues. Accurate and real-time segmentation of pre-operative or intra-operative medical images as well as registration with laparoscopic video can provide surgeons with visualization of subsurface structures. Real-time processing of images using different algorithms will enable development of individual methods for specific surgeries and medical images.

I developed three separate applications that run simultaneously. These parts include video acquisition (video client), video display (video server), and image processing (video server palmetto). We used a high performance-computing (HPC) environment (Clemson's Palmetto cluster) to develop and run our applications in parallel. The image processing application runs on Palmetto's nodes for parallel processing, which contain dual Nvidia Tesla K40 graphics processing units (GPUs). Our setup allows nodes to communicate and transfer images through message passing interface (MPI). This will allow several image processing algorithms to run faster and simultaneously. We tested different setup and number of nodes to compared performance of our framework. Our computing framework

will also enable reliability using replication of computation. Utilizing high-speed network to access computing clusters with GPUs will improve surgical procedures by providing real-time medical image processing and laparoscopic data.

6.2 Methods

6.2.1 Message Passing Interface

Message passing Interface (MPI) is a language independent communications library that allows message exchange between processes and can be used for a variety of languages or compilers. MPI enables high performance, scalability and portability for different applications. It can be implemented on parallel computers, clusters and heterogeneous systems. MPI execution is achieved through process identification, message routing, message buffering and data marshaling [56]. MPI transfers data from sender to receiver through synchronization. MPI provides a selection of send and receive primitives to select based on the programmer's application. For example the syntax of MPI_Send is:

```
int MPI_Send( void          *buf,       //Address of the send buffer
              int           count,      //Number of items in message
              MPI_Datatype  datatype,   //Type of data being
             communicated
              int           dest,       //Destination process rank
              int           tag,        //Type of message
              MPI_Comm      comm        //Communicator context of
             'dest'
             )
```

On the receiving side, MPI_Recv is used that mirrors the syntax of MPI_Send but also includes MPI_Status structure pointer. The syntax of MPI_Recv:

```
int MPI_Recv( void          *buf,       //Address of the receive buffer
              int           count,      //Buffer capacity in items
              MPI_Datatype  datatype,   //Type of data being
              communicated
              int           source,     //Rank of sending process
              int           tag,        //Type of message
              MPI_Comm      comm        //Communicator context of
              'source'
              MPI_Status    *status     //Pointer to structure holding
                                        //message parameters
              )
```

MPI_Send and MPI_Recv are point-to-point and use blocking communication, they block operations until message is delivered. MPI_Send does not return until buffer is empty and available for reuse, and MPI_Recv does not return until buffer is full and available for use. Blocking communication is simple to use but is prone to deadlocks. However, MPI_Isend and MPI_Irecv use non-blocking communication that return immediately and allow overlapping computation and communication. MPI also allows collective communication for operations that involve more than two nodes. For example MPI_Bcast or MPI_Scatter can be used to send messages to a subset of processes and on the receiving side an operation such as MPI_Gather or MPI_Reduce is used to collect data from all the processes. They can also be used to send or receive a message to a subset of processes by creating a custom communicator [56]. We have tested MPI_Send and MPI_Isend in our application. MPI_Bcast will be tested in future applications based on the number of algorithms and available nodes.

6.2.2 MPI and CUDA

We investigated a MPI approach with CUDA on a HPC GPU cluster (Palmetto). MPI and CUDA can be used to control multiple hosts and more GPUs. MPI can transfer data between host buffers using two different approaches. If MPI is not CUDA-aware, data will be transferred from device to host before the MPI call. For MPI CUDA-aware, device buffers can be accessed directly through pointers to device memory in MPI calls [56]. This can be done through GPUDirect to reduce dependence on CPU for managing transfers.

6.3 Results and Discussion

6.3.1 MPI and CUDA

The following graphs show matrix vector multiplication using MPI. Matrix sizes for our testing include: 128, 256, 2048 and 4096. Graph 1 indicates

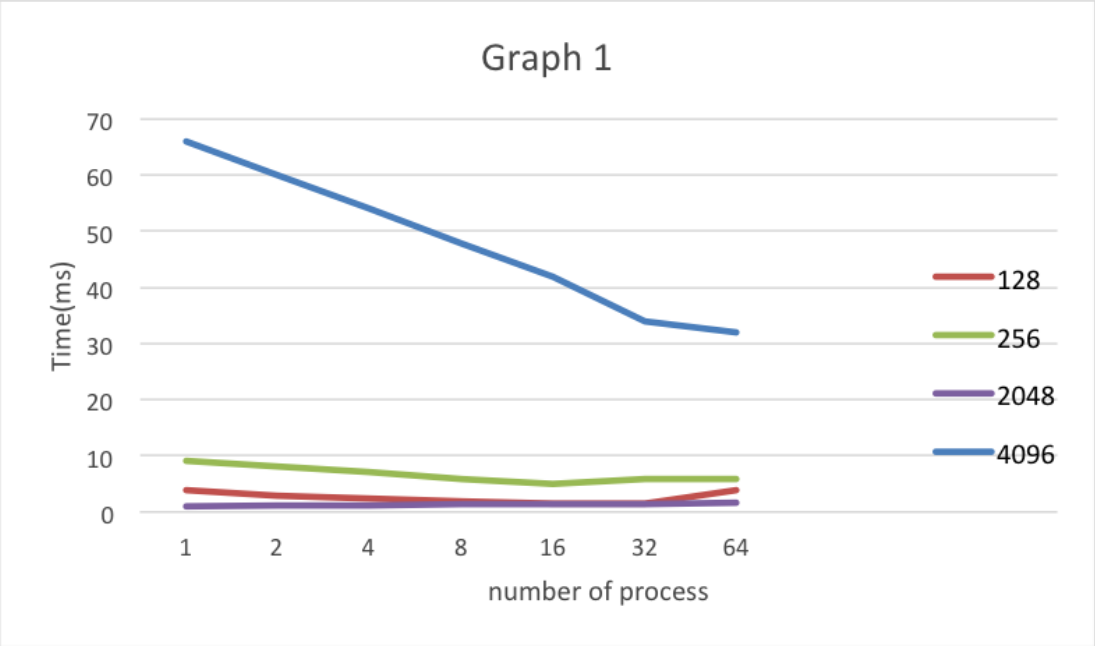


Figure 6.1 – MPI/CUDA block size 2

Graph 1 uses block size of 2 for GPU implementation. The results indicate that the total time for 128, 256 and 2048 stays flat and around the same number, therefore is scalable and with increasing the number of nodes results will stay the same. Graph 2 uses block size of 16 for the GPU implementation. The total time for different size matrices are less than Graph 1 and faster processing time.

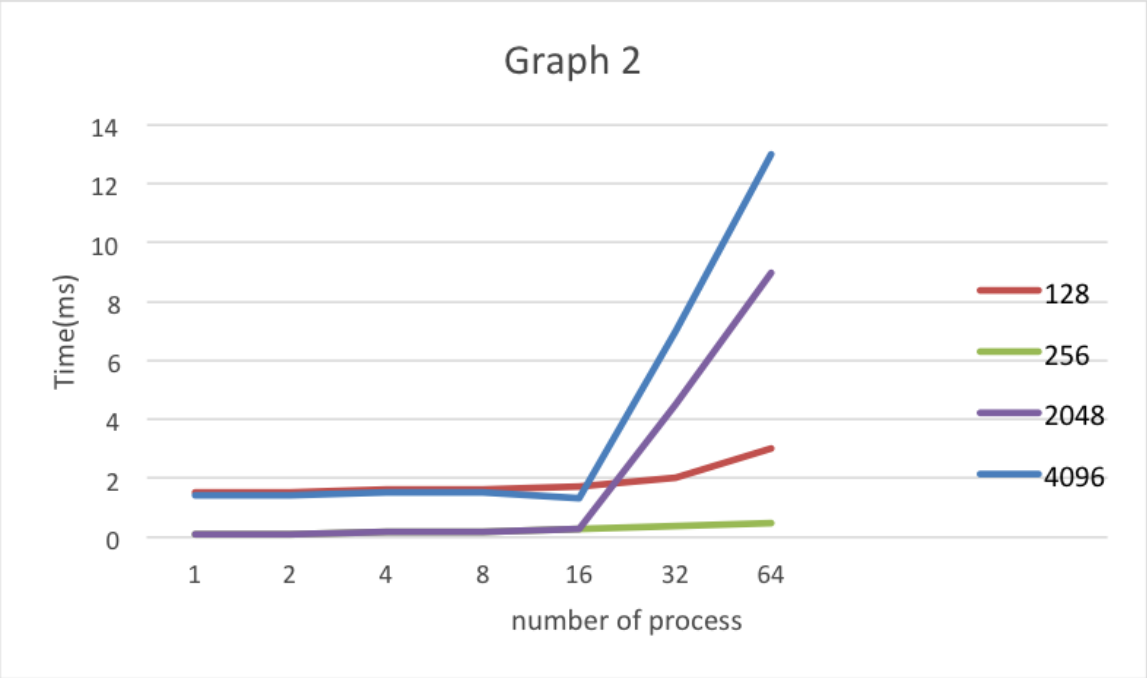


Figure 6.2 – MPI/CUDA block size 16

In order to determine the optimal way to process our images we needed to determine programming and hardware architecture. The algorithms that were used have different computation and memory utilization characteristics. The complete execution time was profiled using Nvidia's visual profiler ("nvvp"). The varying communication and computation time were measured to determine the optimal configuration of the algorithms. Based on this configuration, the most efficient architecture and node configuration can be determined.

We implemented our image processing algorithms using CUDA/MPI on Palmetto cluster. We performed video segmentation as well as registration of the pre-operative images to the video. The algorithms were executed simultaneously on varying amounts of nodes to process the images received from the laparoscopic camera. The following graph shows MPI ranks and connections:

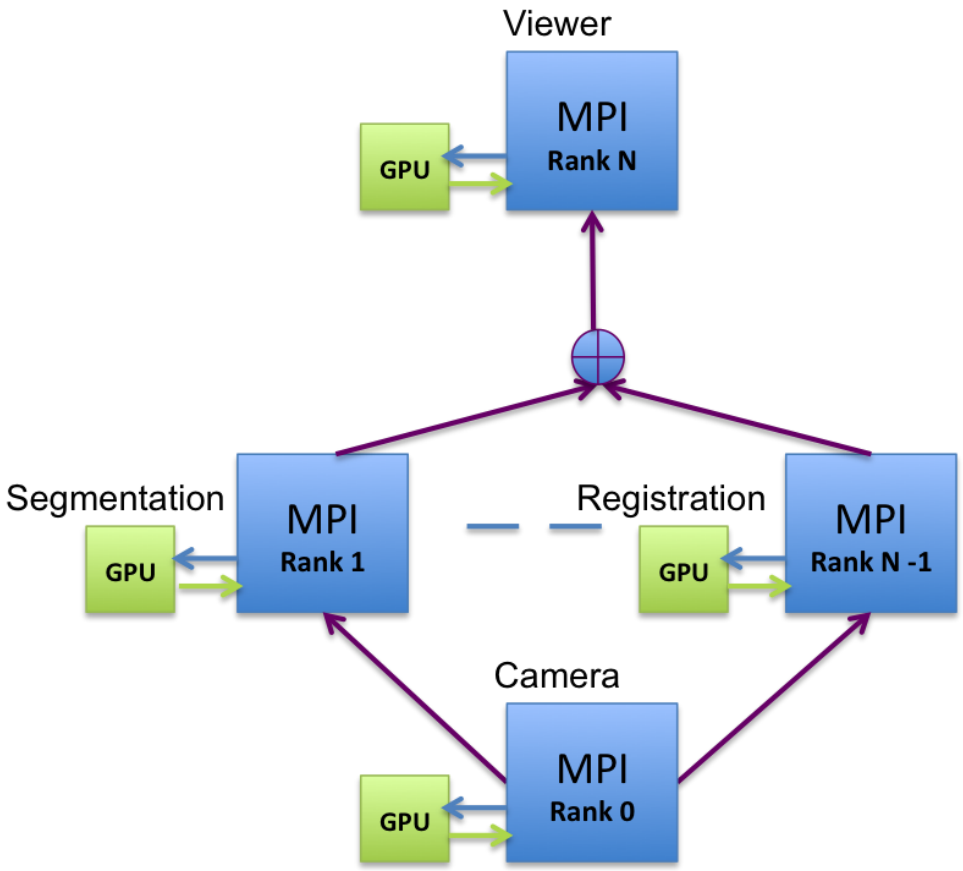


Figure 6.3 – MPI and CUDA on Palmetto

We have tested one or two MPI ranks per node. For two ranks, two algorithms were performed with each applied on an individual GPU. The table below indicates total run-time depending on the number of nodes:

Table 6.1-MPI/CUDA

# of Nodes	1	2
Time (CPU)	3ms	4ms
Time (CPU) HD	23ms	29ms
Time (GPU)	3ms	4ms
Time (GPU) HD	11ms	16ms

Chapter 7

CONCLUSIONS AND FUTURE WORK

We have implemented and compared performance of compression, segmentation and registration algorithms on Clemson's Palmetto supercomputer using dual Nvidia Tesla K40 GPUs per node and CUDA 7.5 enabled with Intel Xeon-E5 CPUs. We developed three separate applications that run simultaneously: video acquisition (video client), video display (video server), and image processing (video server palmetto).

We transferred and compared transmission time of medical images and video using different network protocols. These protocols include SFTP, UDP, TCP and SOS. SFTP runs over TCP, which uses acknowledgements and retransmission to prevent packet loss and causes delays, the encryption step is also a limiting factor. UDP is faster than TCP, since there is no acknowledgment mechanism. The UDP tests resulted in a lower round-trip time and no packet loss due to availability of high bandwidth. However, UDP and TCP have a packet size limit of 64 KB and HD or larger files have to get reassembled before processing. SOS runs on TCP and enabled transfer of larger files. The performance can be tuned to the environment and the quantity of parallel connections to utilize, agent buffer sizes, and queuing for lost/out of order data can be customized.

The image processing application was tested on the cluster to compare segmentation, rigid and non-rigid registration on CPU and GPU for different size images: 640 x 480 and 1920 x 1080. The segmentation algorithms resulted in an acceleration factor

of ~ 2 times compared to the CPU implementation. Registration is computationally more expensive than the segmentation algorithms, therefore higher performance and speed up was observed. Non-rigid registration was performed ~2 times faster for 640 x 480 images and ~ 9 times for HD images. To achieve a higher frame rate, we also resized images and reduced the overall processing time. We observed that algorithms such as bilateral filtering and mean-shift segmentation can achieve strong-scaling gain and for operations such as canny edge detection the improvement was not significant.

We also implemented and executed the algorithms mentioned above to run simultaneously on different cluster nodes instead of sequential on a single node. The nodes communicate and transfer images through message passing interface (MPI) and CUDA was used for parallel implementation on GPUs. As a result, MPI-CUDA was used to control multiple hosts and perform GPU algorithms and it enabled reliability using replication of computation.

In conclusion, previously generated images and videos were transferred to the cluster and processed for adaption and deformation. This could lead to the ability of performing surgery at one site (origin), receive an object map from another site (segmentation/rendering), forward this data to a third site for deformation (model update), and returning this data to the origin for display. As a result, utilizing high-speed network to access computing clusters with GPUs will improve surgical procedures by providing real-time medical image processing and laparoscopic data.

Future research can investigate and perform the image processing tests using GPUDirect peer-to-peer for the GPUs on the same node and GPUDirect-RDMA for the GPUs on different nodes. This will enable CUDA-aware MPI for further optimization. The tests can also be performed using CloudLab [65] to transfer the files between Clemson and Wisconsin and enable SDN to measure the overall time of the framework. The video feed can be augmented at the remote site (Wisconsin) with the inclusion of an overlay and rendered in a "real-time" manor. SOS can securely transfer the files to remote HPC clusters utilizing an OpenFlow-based network service and increase performance of large data transfers over long-distance and high bandwidth networks.

The system accuracy can be tested using phantoms and mock surgical experiments. These phantoms can be constructed using plastics, rubbers or natural materials. A tissue analog will have some or all properties of the specified tissue such as compressibility, strength, imaging properties and diffusion characteristics. Polyvinyl alcohol (PVA) gels have been used as phantoms for MR and ultrasound imaging [66] and a sharp object, scalpel, can apply a high strain force to cut the tissue [67]. Measurement of the resection ratio can be used as a metric for quality of resection. The location, size and depth of the cavity can be calculated by subtracting the difference between image intensities of pre-operative and post-operative images. These tests could indicate whether the clinician can resect the tumor region with equal or better accuracy when compared to augmented video without high performance computing cluster.

APPENDICES

A: Example for UDP socket testing

```
  /* connect to viewer */
  if((sock_view = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    die("socket()");

  struct hostent *view_host = gethostbyname(argv[1]);
  struct in_addr  view_addr;
  view_addr.s_addr = *(unsigned long *)view_host->h_addr_list[0];
  memset((char *)&si_view, 0, sizeof(si_view));
  si_view.sin_family = AF_INET;
  si_view.sin_port = htons(atoi(argv[2]));
  if(inet_aton(inet_ntoa(view_addr), &si_view.sin_addr) == 0)
    die("inet_aton\n");

/* listen to camera */
  if((sock_camera = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
    die("socket()");
  }
  memset((char *)&si_me, 0, sizeof(si_me));
  si_me.sin_family = AF_INET;
  si_me.sin_port = htons(PORT);
  si_me.sin_addr.s_addr = htonl(INADDR_ANY);
  bind(sock_camera, &si_me, sizeof(si_me));

  printf("augmenter listening on %s:%d\n", self, PORT);

  for(;;) {
    if(recvfrom(sock_camera, f, fsz, 0, &si_camera, &slen) == -1)
      die("recv()");
      fsz = frame_size(f);  /* get actual frame size */
      if(f->frameNo % 1000 == 0) {
        printf("recieved from %s:%d; frame %05lu (%lu bytes)\n",
              inet_ntoa(si_camera.sin_addr),
ntohs(si_camera.sin_port),
              f->frameNo, fsz);
      }
      if(sendto(sock_view, f, fsz, 0, &si_view, slen) == -1)
        die("send()\n");
  }
```

B: Thresholding

```
split(img.mat, planes); // Partition image into three channel planes
blue = planes[0];
green = planes[1];
red = planes[2];
threshold(red, dst, threshold_value, max_BINARY_value, threshold_type);


findContours(dst, contours, hierarchy,CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);

int largest_area = 0;
int largest_contour_index = 0;
for (size_t i = 0; i< contours.size(); i++) // iterate through each contour.
{
    double a = contourArea(contours[i], false);  //  Find the area of contour
    if (a>largest_area){
        largest_area = a;
        largest_contour_index = i;              //Store the index of largest contour
Scalar color(rand() & 255, rand() & 255, rand() & 255);
    Scalar colorcode = Scalar(0, 0, 0);
    Mat dsts(dst.rows, dst.cols, CV_8UC1, colorcode);
    drawContours(dsts, contours, largest_contour_index, color, CV_FILLED, 8,
hierarchy);

    Mat imageROI;
    img.mat.copyTo(imageROI, dsts); contours

    frame overlay;
    beta = (1.0 - alpha);
    addWeighted(img.mat, alpha, imageROI, beta, 0.0, overlay.mat);

    frame overlays;
    cvtColor(img.mat, gray, CV_BGR2GRAY);

    Canny(gray, edge, 50, 150, 3);
    edge.convertTo(overlays.mat, CV_8U);

    frame adds;
    overlay.mat.copyTo(adds.mat, overlays.mat);
    frame finals;
    finals.timestamp = img.timestamp;
    add(overlay.mat, adds.mat, finals.mat);
```

BIBLIOGRAPHY

[1] Vitiello, V., Lee, S. L., Cundy, T. P., & Yang, G. Z. (2013). Emerging robotic platforms for minimally invasive surgery. IEEE reviews in biomedical engineering, 6, 111-126.

[2] Mirota, D. J., Ishii, M., & Hager, G. D. (2011). Vision-based navigation in image-guided interventions. Annual review of biomedical engineering, 13, 297-319.

[3] Linte, C. A., White, J., Eagleson, R., Guiraudon, G. M., & Peters, T. M. (2010). Virtual and augmented medical imaging environments: Enabling technology for minimally invasive cardiac interventional guidance. Biomedical Engineering, IEEE Reviews in, 3, 25-47.

[4] Krempien, R., Hoppe, H., Kahrs, L., Daeuber, S., Schorr, O., Eggers, G., ... & Harms, W. (2008). Projector-based augmented reality for intuitive intraoperative guidance in image-guided 3d interstitial brachytherapy. International Journal of Radiation Oncology* Biology* Physics, 70(3), 944-952.

[5] Graham, S. D., Keane, T. E., & Glenn, J. F. (Eds.). (2010). Glenn's urologic surgery. Lippincott Williams & Wilkins.

[6] Grimson, W. E. L., Kikinis, R. J. F. A., Jolesz, F. A., & Black, P. M. (1999). Image-guided surgery. Scientific American, 280(6), 54-61.

[7] Peters, T. M. (2006). Image-guidance for surgical procedures. Physics in medicine and biology, 51(14), R505.

[8] Fuchs, H., Livingston, M. A., Raskar, R., Keller, K., Crawford, J. R., Rademacher, P., ... & Meyer, A. A. (1998, October). Augmented reality visualization for laparoscopic surgery. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 934-943). Springer Berlin Heidelberg.

[9] Baumhauer, M., Feuerstein, M., Meinzer, H. P., & Rassweiler, J. (2008). Navigation in endoscopic soft tissue surgery: perspectives and limitations. Journal of Endourology, 22(4), 751-766.

[10] Van Krevelen, D. W. F., & Poelman, R. (2010). A survey of augmented reality technologies, applications and limitations. International Journal of Virtual Reality, 9(2), 1.

[11] Lee, S. L., Lerotic, M., Vitiello, V., Giannarou, S., Kwok, K. W., Visentini-Scarzanella, M., & Yang, G. Z. (2010). From medical images to minimally invasive intervention: computer assistance for robotic surgery. Computerized Medical Imaging and Graphics, 34(1), 33-45.

[12] Kwartowitz, D. M., Miga, M. I., Herrell, S. D., & Galloway, R. L. (2009). Towards

image guided robotic surgery: multi-arm tracking through hybrid localization. International journal of computer assisted radiology and surgery, 4(3), 281-286.

[13] Kwartowitz, D. M., Herrell, S. D., & Galloway, R. L. (2006). Toward image-guided robotic surgery: determining intrinsic accuracy of the da Vinci robot. International Journal of Computer Assisted Radiology and Surgery, 1(3), 157-165.

[14] Shams, R., Sadeghi, P., Kennedy, R. A., & Hartley, R. I. (2010). A survey of medical image registration on multicore and the GPU. IEEE Signal Processing Magazine, 27(2), 50-60.

[15] Samant, S. S., Xia, J., Muyan-Özçelik, P., & Owens, J. D. (2008). High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy. Medical physics, 35(8), 3546-3553.

[16] Fluck, O., Vetter, C., Wein, W., Kamen, A., Preim, B., & Westermann, R. (2011). A survey of medical image registration on graphics hardware. Computer methods and programs in biomedicine, 104(3), e45-e57.

[17] Open Networking Foundation, "Software defined networking (sdn) definition," https://www. opennetworking.org/sdn-resources/sdn-definition (2015).

[18] Brocade, "100 gigabit ethernet: The way forward for research networks," http://info.brocade.com/rs/brocade/images/CDE14%20100GbE%20BRIEF%20Brocade.PDF (2014).

[19] Galloway, Robert, S. Herrell, and Michael Miga. "Image-guided abdominal surgery and therapy delivery." Journal of healthcare engineering 3.2 (2012): 203-228.

[20] Hughes-Hallett, A., Mayer, E. K., Marcus, H. J., Cundy, T. P., Pratt, P. J., Darzi, A. W., & Vale, J. A. (2014). Augmented reality partial nephrectomy: examining the current status and future perspectives. Urology, 83(2), 266-273.

[21] Su, L. M., Vagvolgyi, B. P., Agarwal, R., Reiley, C. E., Taylor, R. H., & Hager, G. D. (2009). Augmented reality during robot-assisted laparoscopic partial nephrectomy: toward real-time 3D-CT to stereoscopic video registration. Urology, 73(4), 896-900.

[22] Oliveira, Francisco PM, and João Manuel RS Tavares. "Medical image registration: a review." Computer methods in biomechanics and biomedical engineering 17.2 (2014): 73-93.

[23] Smistad, E., Falch, T. L., Bozorgi, M., Elster, A. C., & Lindseth, F. (2015). Medical image segmentation on GPUs–A comprehensive review. Medical image analysis, 20(1), 1-18.

[24] Shams, R., Sadeghi, P., Kennedy, R., & Hartley, R. (2010). Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. Computer methods and programs in biomedicine, 99(2), 133-146.

[25] Brounstein, A., Hacihaliloglu, I., Guy, P., Hodgson, A., & Abugharbieh, R. (2011, September). Towards real-time 3D US to CT bone image registration using phase and curvature feature based GMM matching. In International Conference on Medical Image Computing and Computer-Assisted Intervention

[26] Ruijters, Daniel, Bart M. ter Haar Romeny, and Paul Suetens. "GPU-accelerated elastic 3D image registration for intra-surgical applications." Computer methods and programs in biomedicine 103.2 (2011): 104-112.

[27] Plishker, W., Dandekar, O., Bhattacharyya, S. S., & Shekhar, R. (2010). Utilizing hierarchical multiprocessing for medical image registration. IEEE Signal Processing Magazine, 27(2), 61-68.

[28] Hwang, Kyung Hoon, Haejun Lee, and Duckjoo Choi. "Medical image retrieval: past and present." Healthcare informatics research 18.1 (2012): 3-9.

[29] Eklund, A., Dufort, P., Forsberg, D., & LaConte, S. M. (2013). Medical image processing on the GPU–Past, present and future. Medical image analysis, 17(8), 1073-1094.

[30] Perrot, G., Domas, S., Couturier, R., & Bertaux, N. (2011, August). GPU implementation of a region based algorithm for large images segmentation. In Computer and information technology (CIT), 2011 IEEE 11th international conference on (pp. 291-298). IEEE.

[31] Novotny, P. M., Stoll, J. A., Vasilyev, N. V., Pedro, J., Dupont, P. E., Zickler, T. E., & Howe, R. D. (2007). GPU based real-time instrument tracking with three-dimensional ultrasound. Medical image analysis, 11(5), 458-464.

[32] Körbes, A., Vitor, G. B., de Alencar Lotufo, R., & Ferreira, J. V. (2011, July). Advances on watershed processing on GPU architecture. In International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing (pp. 260-271). Springer Berlin Heidelberg.

[33] Lefohn, A. E., Cates, J. E., & Whitaker, R. T. (2003, November). Interactive, GPU-based level sets for 3D segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 564-572). Springer Berlin Heidelberg.

[34] Tokuda, J., Fischer, G. S., Papademetris, X., Yaniv, Z., Ibanez, L., Cheng, P., ... & Kapur, T. (2009). OpenIGTLink: an open network protocol for image-guided therapy

environment. The International Journal of Medical Robotics and Computer Assisted Surgery, 5(4), 423-434.

[35] Clarkson, M. J., Zombori, G., Thompson, S., Totz, J., Song, Y., Espak, M., ... & Ourselin, S. (2015). The NifTK software platform for image-guided interventions: platform overview and NiftyLink messaging. International journal of computer assisted radiology and surgery, 10(3), 301-316.

[36] Lasso, A., Heffter, T., Rankin, A., Pinter, C., Ungi, T., & Fichtinger, G. (2014). PLUS: open-source toolkit for ultrasound-guided intervention systems. IEEE Transactions on Biomedical Engineering, 61(10), 2527-2537.

[37] Klemm, M., Kirchner, T., Gröhl, J., Cheray, D., Nolden, M., Seitel, A., ... & Franz, A. M. (2016). MITK-OpenIGTLink for combining open-source toolkits in real-time computer-assisted interventions. International Journal of Computer Assisted Radiology and Surgery, 1-11.

[38] Kurose, James F. Computer Networking: A Top-Down Approach Featuring the Internet, 3/E. Pearson Education India, 2005.

[39] Cyberinfrastructure Technology Integration (CITI) group at Clemson University, "Palmetto User's Guide," 2016, https://www.palmetto.clemson.edu/palmetto/pages/userguide.html

[40] Rowshanrad, S., Namvarasl, S., Abdi, V., Hajizadeh, M., & Keshtgary, M. (2014). A survey on SDN, the future of networking. Journal of Advanced Computer Science & Technology, 3(2), 232.

[41] Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials, 16(3), 1617-1634.

[42] Rosen, A. and Wang, K.-C., Steroid OpenFlow Service: Seamless Network Service Delivery in Software Defined Networksg," (2012).

[43] Gropp, W., Lusk, E., & Skjellum, A. (1999). Using MPI: portable parallel programming with the message-passing interface (Vol. 1). MIT press.

[44] Kawasaki, Y., Ino, F., Mizutani, Y., Fujimoto, N., Sasama, T., Sato, Y., ... & Hagihara, K. (2004). High-performance computing service over the Internet for intraoperative image processing. IEEE transactions on information technology in biomedicine, 8(1), 36-46.

[45] Ali, S., & Cherif, A. (2015). Performances analysis of image encryption for medical applications. Journal of Information Sciences and Computing Technologies, 1(1), 78-87.

[46] Gonzalez, Rafael C., and Richard E. Woods. "Digital image processing." (2002).

[47] Huang, Deng-Yuan, and Chia-Hung Wang. "Optimal multi-level thresholding using a two-stage Otsu optimization approach." Pattern Recognition Letters 30.3 (2009): 275-284.

[48] Luo, Y., & Duraiswami, R. (2008, June). Canny edge detection on NVIDIA CUDA. In Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on (pp. 1-8). IEEE.Chicago

[49] Fischer, Bernd, and Jan Modersitzki. "Ill-posed medicine—an introduction to image registration." Inverse Problems 24.3 (2008): 034008.

[50] Rueckert, D., Sonoda, L. I., Hayes, C., Hill, D. L., Leach, M. O., & Hawkes, D. J. (1999). Nonrigid registration using free-form deformations: application to breast MR images. IEEE transactions on medical imaging, 18(8), 712-721.

[51] Lee, Seungyong, George Wolberg, and Sung Yong Shin. "Scattered data interpolation with multilevel B-splines." IEEE transactions on visualization and computer graphics 3.3 (1997): 228-244.

[52] Jiang, Jun, Wayne Luk, and Daniel Rueckert. "FPGA-based computation of free-form deformations in medical image registration." Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on. IEEE, 2003.

[53] Modat, M., Ridgway, G. R., Taylor, Z. A., Lehmann, M., Barnes, J., Hawkes, D. J., ... & Ourselin, S. (2010). Fast free-form deformation using graphics processing units. Computer methods and programs in biomedicine, 98(3), 278-284.

[54] https://sourceforge.net/projects/niftyreg/

[55] Rohlfing, T., Maurer, C. R., Bluemke, D. A., & Jacobs, M. A. (2003). Volume-preserving nonrigid registration of MR breast images using free-form deformation with an incompressibility constraint. IEEE transactions on medical imaging, 22(6), 730-741.

[56] Barlas, Gerassimos. Multicore and GPU Programming: An integrated approach. Elsevier, 2014.

[57] https://docs.nvidia.com/cuda/cuda-c-programming-guide/#introduction

[58] https://developer.nvidia.com/cuda-zone

[59] https://www.khronos.org/opencl/

[60] Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., & Dongarra, J. (2012). From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. Parallel Computing, 38(8), 391-407.

[61] Munshi, A., Gaster, B., Mattson, T. G., & Ginsburg, D. (2011). OpenCL programming guide. Pearson Education.

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

[63] Ogawa, K., Ito, Y., & Nakano, K. (2010, November). Efficient Canny edge detection using a GPU. In Networking and Computing (ICNC), 2010 First International Conference on (pp. 279-280). IEEE.

[64] Holub, P., Šrom, M., Pulec, M., Matela, J., & Jirman, M. (2013). GPU-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2K, and 4K video. Future Generation Computer Systems, 29(8), 1991-2006.

[65] https://www.cloudlab.us

[66] Surry, K. J. M., Austin, H. J. B., Fenster, A., & Peters, T. M. (2004). Poly (vinyl alcohol) cryogel phantoms for use in ultrasound and MR imaging. Physics in medicine and biology, 49(24), 5529.

[67] Chanthasopeephan, T., Desai, J. P., & Lau, A. C. (2007). Modeling soft-tissue deformation prior to cutting for surgical simulation: finite element analysis and study of cutting parameters. Biomedical Engineering, IEEE Transactions on, 54(3), 349-359.

[68] Fitzpatrick, J. M., Hill, D. L., & Maurer Jr, C. R. (2000). Image registration. Handbook of medical imaging, 2, 447-513. Chicago

[69] http://globalconfig.net/software-defined-networking-vs-traditional/

[70] http://opencv.org/