

SOLVING SPARSE LINEAR SYSTEM USING ITERATIVE SOLVERS ON GPU FOR IMPROVED IMAGE RECONSTRUCTION

a project report submitted by

SAMUEL ROY (UR14CS007)

in partial fulfilment for the award of the degree of

**BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING**

under the supervision of

Mr. JEBAN CHANDIR MOSES M.E., (Ph.D.)



DEPARTMENT OF COMPUTER SCIENCES TECHNOLOGY

KARUNYA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Declared as Deemed-to-be-University under Sec-3 of the UGC Act, 1956)

Karunya Nagar, Coimbatore - 641 114, India.

NOVEMBER 2017

BONAFIDE CERTIFICATE

Certified that this project report **“SOLVING SPARSE LINEAR SYSTEM
USING ITERATIVE SOLVERS ON GPU FOR IMPROVED
IMAGE RECONSTRUCTION”** is the bonafide work of **“SAMUEL ROY
(REG. NO: UR14CS007)”** who carried out the project work under my
supervision.

SIGNATURE

Dr. Immanuel John Raja

Programme Coordinator

Department of Computer Sciences Technology,

SIGNATURE

Mr. Jeban Chandir Moses

Supervisor

Assistant Professor

Department of Computer Sciences Technology

Submitted for the Project Viva Voce held on...14.11.2017...

Examiner

Dr. rer. medic. Dipl. -Inf. Felix Gremse
Institute of Experimental Molecular Imaging
RWTH Aachen University Hospital
Pauwelsstraße 30, D-52074 Aachen
Phone: +49 (0) 241 8089761
E-Mail: fgremse@ukaachen.de
Aachen, October 24th 2017

Internship of Samuel Roy

To whom it may concern,

Samuel Roy, born 21st May 1996, performed an internship at our institute as part of his computer science studies at the Karunya University in India for duration of 12 weeks in the time period from 1st August 2017 to 24th October 2017. In this time, Samuel learned to work with Microsoft Visual Studio and the NVIDIA Cuda Toolkit to compile and execute C++/Cuda code about numerical and mathematical problems. In particular, he studied the performance of GPU-accelerated iterative solvers of sparse linear systems which is an important task for many scientific problems such as tomographic fluorescence reconstruction, a research focus of my group.

I appreciate that Samuel was very willing to learn new things and would like to see him involved in further projects of my group. He successfully completed every aspect of his work and the tasks to the highest possible standard. In summary, I would definitely work with Samuel again, and would recommend him to other groups without hesitation. I wish him all the best for his future career.

Sincerely yours,



Felix Gremse

(Group leader, Applied Medical Informatics)

ACKNOWLEDGEMENT

First and foremost, I praise and thank **ALMIGHTY GOD** whose blessings have bestowed in me the will power and confidence to carry out my project.

I am grateful to our beloved founders Late **Dr. D.G.S. Dhinakaran, C.A.I.I.B, Ph.D.**, and **Dr. Paul Dhinakaran, M.B.A., Ph.D.**, for their love and always remembering me in their prayers.

I extend my thanks to our Vice Chancellor **Dr. P. Mannar Jawahar, Ph.D**, and our Registrar **Dr. Elijah Blessing Vinoth, M.E., Ph.D**, for giving me this opportunity to carry out the project.

I would like to thank **Dr. Prince Arulraj, M.E., Ph.D.**, Director, School of Engineering and Technology for his direction and invaluable support to complete the same.

I would like to place my heart-felt thanks and gratitude to **Dr. E. Kirubakaran, M.E., Ph.D.**, Head of the Department, Computer Sciences Technology for his encouragement and guidance.

I would also like to thank **Dr. Immanuel John Raja, M.E., Ph.D.**, Programme Coordinator- UG Computer Science and Engineering for his guidance and support.

I am grateful to my guides, **Dr. rer. medic. Dipl.-Inf. Felix Gremse**, Group Leader, Applied Medical Informatics and **Mr. Jeban Chandir Moses**, Assistant Professor, Department of Computer Sciences Technology for their valuable support, advice and encouragement.

I also thank all the staff members of the Department for extending their helping hands to make this project work a success.

I would also like to thank all my friends and my parents who have prayed and helped me during the project.

ABSTRACT

In Medical imaging Fluorescence-mediated tomography (FMT) enables longitudinal and quantitative determination of the fluorescence distribution in vivo to assess disease progression using established molecular probes or reporter genes. The combination with an anatomical modality, *e.g.*, micro computed tomography (μ CT), is called Hybrid μ CT-FMT imaging which is beneficial for image analysis and for fluorescence reconstruction. The main aim of this project is to leverage the computation capabilities of GPU in order to solve an image reconstruction problem. GPU accelerated computing is the use of a graphical processing unit along with the CPU to accelerate the applications. Statistical image reconstruction methods are replacing traditional methods in commercial μ CT scanners. Statistical methods offer many advantages over traditional methods, including incorporating physical effects and physical constraints, modeling of complex imaging geometries, and imaging at lower X-ray doses. The usage of GPU accelerated computing in the Statistical image reconstruction has enabled it to overcome one of the practical limitations *i.e.* high reconstruction time. To reduce the reconstruction time, several novel iterative algorithms are being used such as the Conjugate Gradient Method. This project has been implemented with C++ using the CUDA Toolkit. Mathematical libraries such as CUBLAS, CUSPARSE and CUSP which are compatible with the CUDA framework have been used. Compressed Sparse Row format which is an efficient way to store sparse matrices is used.

KEYWORDS: CUDA, GPU-Computing, Image-reconstruction, Conjugate Gradient, Sparse Matrix

CONTENTS

| | |
|--|------|
| Certificate | iii |
| Acknowledgement | iv |
| Abstract | v |
| List of Figures | viii |
| List of Tables | ix |
| List of symbols and abbreviations | x |
| 1. Chapter 1: Introduction | |
| 1.1 Problem Statement | 1 |
| 1.2 Objective of the Project | 1 |
| 1.3 Profile and Objective of the Organization | 2 |
| 1.4 Overview of the Internship | 2 |
| 1.5 Chapter wise summary | 2 |
| 2. Chapter 2: System Analysis | |
| 2.1 Study of μ CT- FMT process | 3 |
| 2.2 Protocol | 6 |
| 2.3 X-ray Generation | 8 |
| 2.4 μ CT instrumentation | 11 |
| 2.5 μ CT reconstruction | 11 |
| 2.6 Understanding the absorption reconstruction and optical scattering | 12 |
| 2.7 Existing System | 14 |
| 2.8 Proposed System | 15 |
| 2.9 Use Case Analysis | 17 |
| 2.10 Sequence Diagram | 19 |
| 2.11 Detailed study of functionalities | 21 |
| 2.12 Requirements Specification | 21 |
| 2.13 Constraints | 22 |

| | |
|--|----|
| 3. Chapter 3: System Design | |
| 3.1 Architecture Overview | 21 |
| 3.2 Heterogeneous Computing | 22 |
| 3.3 Processing Flow of GPU | 23 |
| 3.4 CSR format for Computation | 25 |
| 4. Chapter 4: System Implementation | |
| 4.1 Module Implementation | 30 |
| 4.2 Graphical Processing Unit | 30 |
| 4.3 Microsoft Visual C++ | 31 |
| 4.4 CUDA Toolkit | 31 |
| 4.5 Libraries Used | 32 |
| 4.6 Testing and Results | 35 |
| 5. Chapter 4: System Implementation | |
| 5.1 Conclusion | 37 |
| 5.2 Future Scope | 37 |
| Appendix A (Source Code) | 38 |
| Appendix B (Output Screenshot) | 45 |
| References | 48 |

LIST OF FIGURES

| FIG NO. | TITLE | PAGE NO. |
|----------------|--|-----------------|
| 2.1 | Fluorescence Mediated Tomography | 3 |
| 2.2 | FMT Image | 3 |
| 2.3 | μ CT imaging | 4 |
| 2.4 | Anatomical images from μ CT | 4 |
| 2.5 | Features of FMT, μ -CT and Hybrid μ -CT- FMT | 5 |
| 2.6 | Hybrid μ -CT- FMT image | 6 |
| 2.7 | Flow Chart summarizing the Protocol | 7 |
| 2.8 | Organ segmentation using Imalytics Preclinical | 8 |
| 2.9 | Complete process from Scanning to the reconstruction | 8 |
| 2.10 | Working of X-ray Tube | 9 |
| 2.11 | Absorption or Scattering of the Cathode electron by | 9 |
| 2.12 | Tungsten X-ray generation energy graph | 10 |
| 2.13 | Figure showing μ -CT instrumentation | 11 |
| 2.14 | Radon reconstruction using the Shepp-Logan phantom image | 12 |
| 2.15 | Working of X-ray computed tomography | 12 |
| 2.16 | $Ax=b$ represented in matrix format | 13 |
| 2.17 | Graph showing convergence rate of different iterative algorithms | 16 |
| 2.18 | Use Case Diagram | 18 |
| 2.19 | Sequence Diagram | 19 |
| 2.20 | Flow Chart showing the Functionalities involved | 21 |
| 3.1 | CUDA Architecture | 23 |
| 3.2 | NVIDIA GPU –DEVICE | 25 |
| 3.3 | CPU RAM –HOST | 25 |
| 3.4 | Device and Host Memory | 25 |
| 3.5 | Data copied from CPU to GPU | 26 |
| 3.6 | Loading and execution of GPU program | 26 |
| 3.7 | Copy Results to CPU memory | 27 |

| | | |
|-----|---|----|
| 3.8 | Example of CSR stored Sparse Matrix | 28 |
| 4.1 | Poisson Matrix used for testing | 37 |
| 4.2 | Graph plotting the different methods and number of iterations | 38 |

LIST OF TABLES

| FIG NO. | TITLE | PAGE NO. |
|---------|-------------------------------------|----------|
| 2.1 | Hardware Requirements | 19 |
| 3.1 | CUDA Toolkit | 22 |
| 4.1 | Number of Iterations in each method | 35 |

LIST OF SYMBOLS AND ABBREVIATIONS

1. FMT - Fluorescence-mediated tomography
2. μ -CT - micro X-ray computed Tomography
3. GPU – Graphical Processing Unit
4. CUDA - Compute Unified Device Architecture

CHAPTER 1

INTRODUCTION

Fluorescence-mediated tomography, also called fluorescence molecular tomography (FMT), is a technique to quantitatively assess the fluorescence distribution in diffuse tissues, such as anaesthetized mice or even human body tissues. FMT strongly benefits from the combination with an anatomical modality such as μ -CT, as the fluorescence images are difficult to interpret without anatomical reference information.

In the micro X-ray Computed Tomography (μ -CT) Statistical methods especially iterative methods are more suitable for the reconstruction of images with high contrast and precision in the noisy conditions when compared to the traditional methods. But these methods are not widely used due to the high computational cost of implementation. However, nowadays these methods can be used due to the high computational capability of GPU accelerated computing.

This project is an attempt to implement a few iterative methods for image reconstruction on GPU and evaluate their performance. This project has been implemented in C++ using the CUDA Toolkit. Mathematical libraries such as CUBLAS, CUSPARSE and CUSP which are compatible with the CUDA framework have been used. Compressed Sparse Row format which is an efficient way to store sparse matrices is used.

1.1 PROBLEM STATEMENT

To Solve the Sparse linear system in the form of $Ax=b$ using iterative solvers using most efficient Math Libraries on Graphical Processing Unit for improved image reconstruction in μ -CT imaging.

1.2 OBJECTIVES OF THE PROJECT

- Study of Hybrid μ CT-FMT imaging Process (Multimodal imaging)
- Understanding the absorption reconstruction and optical scattering in μ -CT
- Study of Sparse Matrix Linear System and Solving Methods
- Usage of Graphical Processing Unit for faster computation of solution of Sparse Matrix Linear system for application in image reconstruction
- Implementation of the iterative Method using CUDA C++

1.3 PROFILE AND OBJECTIVE OF THE ORGANIZATION

The Uniklinikum Aachen, full German name Universitätsklinikum Aachen is the university hospital of the city of Aachen, Germany. The Institute for Experimental Molecular Imaging (ExMI) at the University Hospital Aachen(Uniklinik) at RWTH Aachen University is headed by Univ.-Prof. Dr. med. Fabian Kiessling. ExMI focuses on the development of novel contrast agents, imaging techniques and therapeutic approaches to characterize and treat cancer, cardiovascular and inflammatory disorders.

The Applied Medical Informatics group develops methods to quantitatively analyze clinical and preclinical image data in the scope of interdisciplinary research. The aim is to automate certain image segmentation or analysis steps, or at least minimize the required user input, to achieve a high level of user independence and therefore reproducibility.

1.4 OVERVIEW OF THE INTERNSHIP

The Experimental Molecular imaging (ExMi) department offers research opportunities especially to Masters and Doctoral students. Bachelor students in their final year of study are also provided with internship opportunities. This internship program was arranged, stipend funded by DAAD-IAESTE as a part of Practical Traineeships for Foreign Students of Natural and Technical Sciences, Agriculture and Forestry.

The internship was carried out under the guidance of Dr. rer. medic. Dipl.-Inf. Felix Gremse, Group Leader, Applied Medical Informatics, Experimental Molecular Imaging, Aachen, Germany. The internship program has contributed to the improvement in professional and practical understanding of the concepts and development of intercultural skills.

1.5 CHAPTER-WISE SUMMARY

Chapter 2 deals with the analysis of the use case diagram used in the project and describe the hardware and software requirements of the project. Chapter 3 is dealt with the fabrication of the problem-solving methodology with a module-wise explanation on the significant functions. Chapter 4 explains how the system is implemented and the various function used along with testing. Chapter 5 gives a conclusive assertion of the uses of this system and also the future scope is discussed.

CHAPTER 2

SYSTEM ANALYSIS

The chapter deals with the Study of Hybrid μ CT-FMT imaging Process (Multimodal imaging), analysis of the architecture of the system used in the project, describes the hardware and software requirements of the project.

2.1 STUDY OF μ CT-FMT IMAGING PROCESS

The study of μ CT-FMT imaging process is important to understand the work carried out at ExMI and also to introduce the working of Hybrid FMT and μ CT process.

2.1.1 FMT imaging

Fluorescence-mediated tomography, also called fluorescence molecular tomography (FMT), is a technique to quantitatively assess the fluorescence distribution in diffuse tissues, such as anaesthetized mice or even human body tissues. Fig 2.1 shows an FMT imaging scanner and Fig 2.2 shows an FMT image.

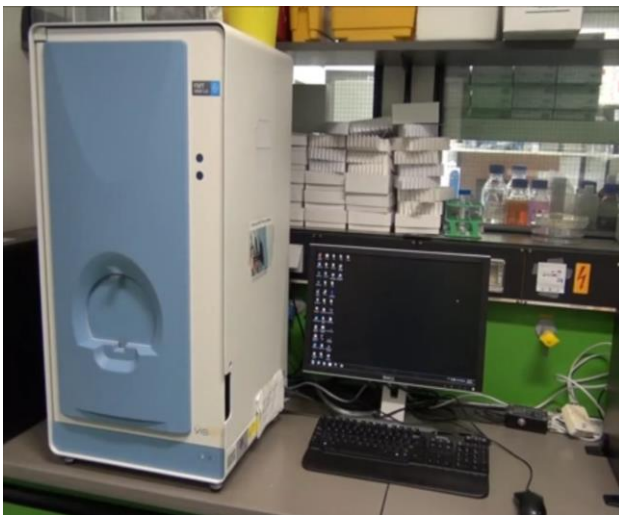


Fig 2.1: Fluorescence Mediated Tomography

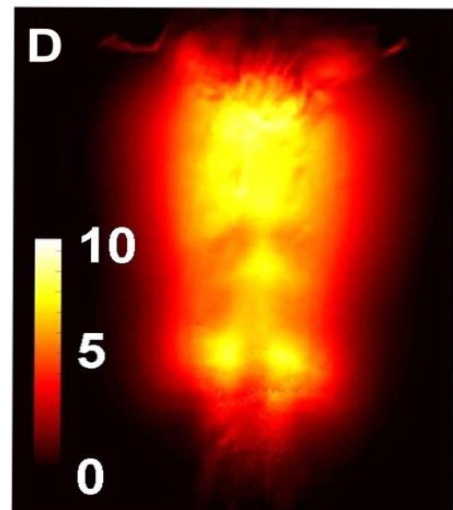


Fig 2.2: FMT Image

Many targeted fluorescent probes are available to image angiogenesis, apoptosis, inflammation, fluorescent proteins can be imaged, e.g., to track tumour cell. FMT allows three-dimensional reconstruction of fluorescent sources in depths of several centimetres, although at a

lower resolution. Fluorescence-mediated tomography holds potential for accelerating diagnostic and theranostic drug development. However, for proper quantitative fluorescence reconstruction, knowledge on optical scattering and absorption, which are highly heterogeneous in different (mouse) tissues, is required.

2.1.2 μ CT imaging

A CT scan, also known as computed tomography scan, makes use of computer-processed combinations of many X-ray measurements taken from different angles to produce cross-sectional (tomographic) images (virtual "slices") of specific areas of a scanned object, allowing the user to see inside the object without cutting but on a small scale with massively increased resolution. CT produces a volume of data that can be manipulated in order to demonstrate various bodily structures based on their ability to absorb the X-ray beam.



Fig 2.3 μ CT imaging

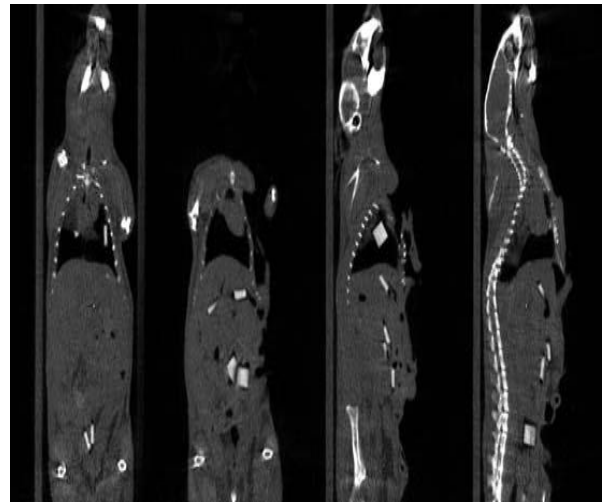


Fig 2.4 Anatomical images from μ CT

X-ray microtomography also uses x-rays to create cross-sections of a physical object that can be used to recreate a virtual model (3D model) without destroying the original object. The prefix micro- (symbol: μ) is used to indicate that the pixel sizes of the cross-sections are in the micrometre range. Fig 2.3 shows a μ CT scanner and Fig 2.4 shows an Anatomical Modality. Micro-CT has applications both in medical imaging and in industrial computed tomography. In a

clinical CT scanner setup, is gantry based where the animal/specimen is stationary in space while the X-ray tube and detector rotate around.

These scanners are typically used for small animals (in vivo scanners), biomedical samples, foods, microfossils, and other studies for which minute detail is desired. The sample is rotated by a fraction of a degree and another projection image is taken at the new position. This procedure is iterated until the sample has rotated 180 or 360 degrees producing a series of projection images. The projection images are then processed using computer software to show the internal structure of the object nondestructively. This series of images is typically called the reconstructed images or cross sections

2.1.3 Hybrid μ -CT- FMT imaging

FMT strongly benefits from the combination with an anatomical modality. While stand-alone FMT devices are commercially available, the fluorescence images are difficult to interpret without anatomical reference information, the fused anatomical image data enables a more robust analysis.

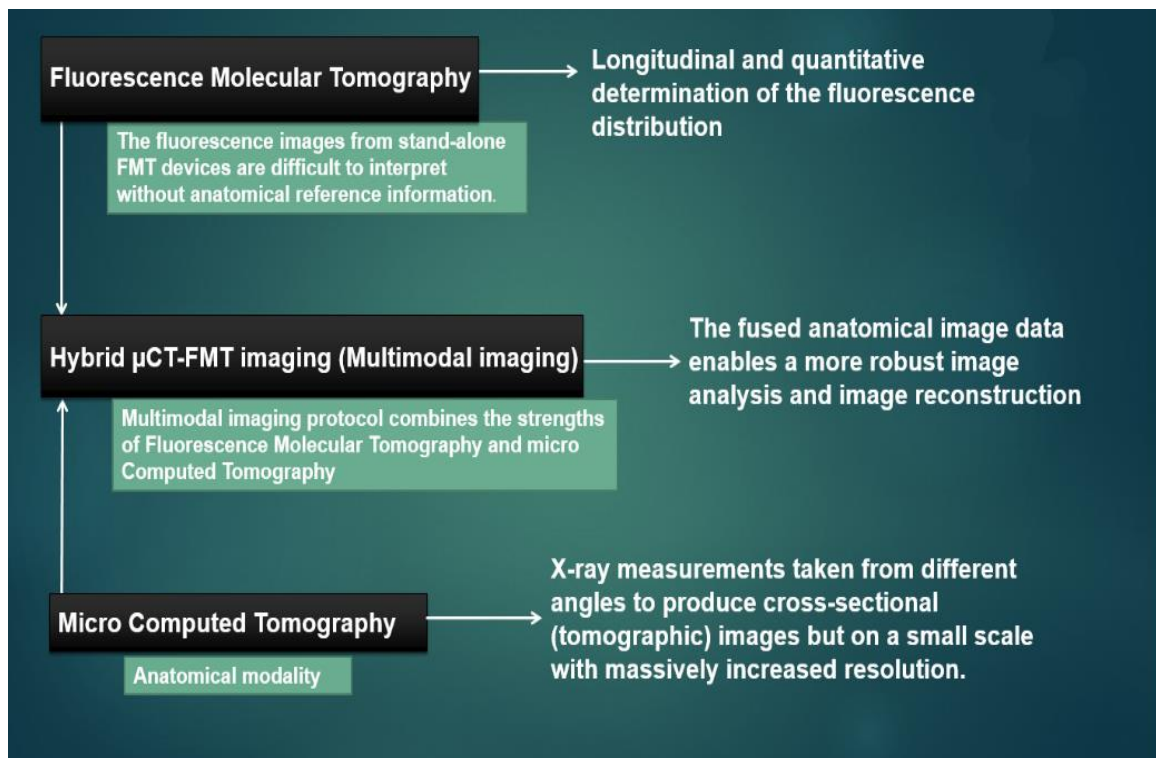


Fig 2.5 Features of FMT, μ -CT and Hybrid μ -CT- FMT

The anatomical data can also be used to provide prior knowledge, such as the outer shape of the mouse, which is important for accurate optical modelling and fluorescence reconstruction. Optical scattering and absorption maps can be estimated using segmentation of tissue types and by assigning class specific coefficients.

After sequential μ CT-FMT imaging, the μ CT images are automatically segmented to find markers built into the mouse bed for automated fusion. Furthermore, an automated segmentation of the outer shape and of several tissue regions for the scattering map can generated. The whole-animal absorption map was reconstructed by iterative minimization of the difference between predicted and measured excitation boundary values.

The absorption reconstruction and the effect on the fluorescence reconstruction are evaluated using phantom experiments. In vivo, the whole-animal absorption reconstruction can be validated using the μ CT-derived relative blood volume, because the latter is known to essentially determine the absorption in many organs. A scattering map from the μ CT-based segmentation and reconstruct a whole-animal absorption map from the FMT measurements.

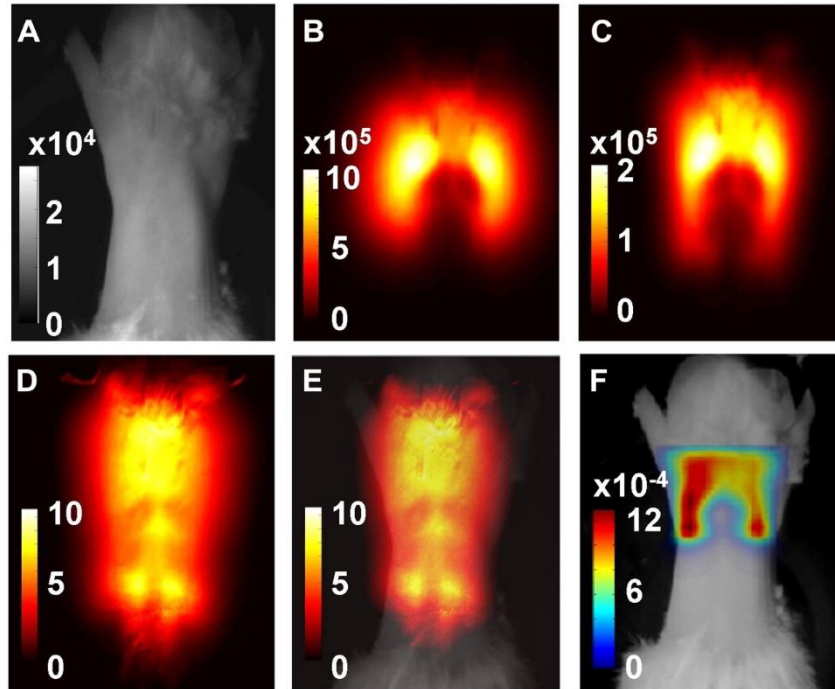


Fig 2.6 Hybrid μ -CT- FMT image

2.2 PROTOCOL

In the Video Article Hybrid μ CT-FMT imaging and image analysis performed at Uniklinik RWTH Aachen the protocol has been summarized as below and as a flowchart in Fig 2.7:

- At first, phantoms or mice and the multimodal mouse bed are prepared for imaging. Then a whole-body scan is acquired in the μ CT.
- Subsequently, the mouse bed is transferred to the FMT where two scans are acquired (up and upside down). This can be repeated for multiple mice at multiple time points.
- After completion of the data acquisition, the data needs to be exported and sorted to enable automated segmentation (requiring a Definiens software license), as well as image fusion and fluorescence reconstruction (requiring an Analytics Preclinical software license).
- Finally, it is shown how the multimodal datasets are visualized and how organs are interactively segmented to quantify the biodistribution of fluorescent probes.

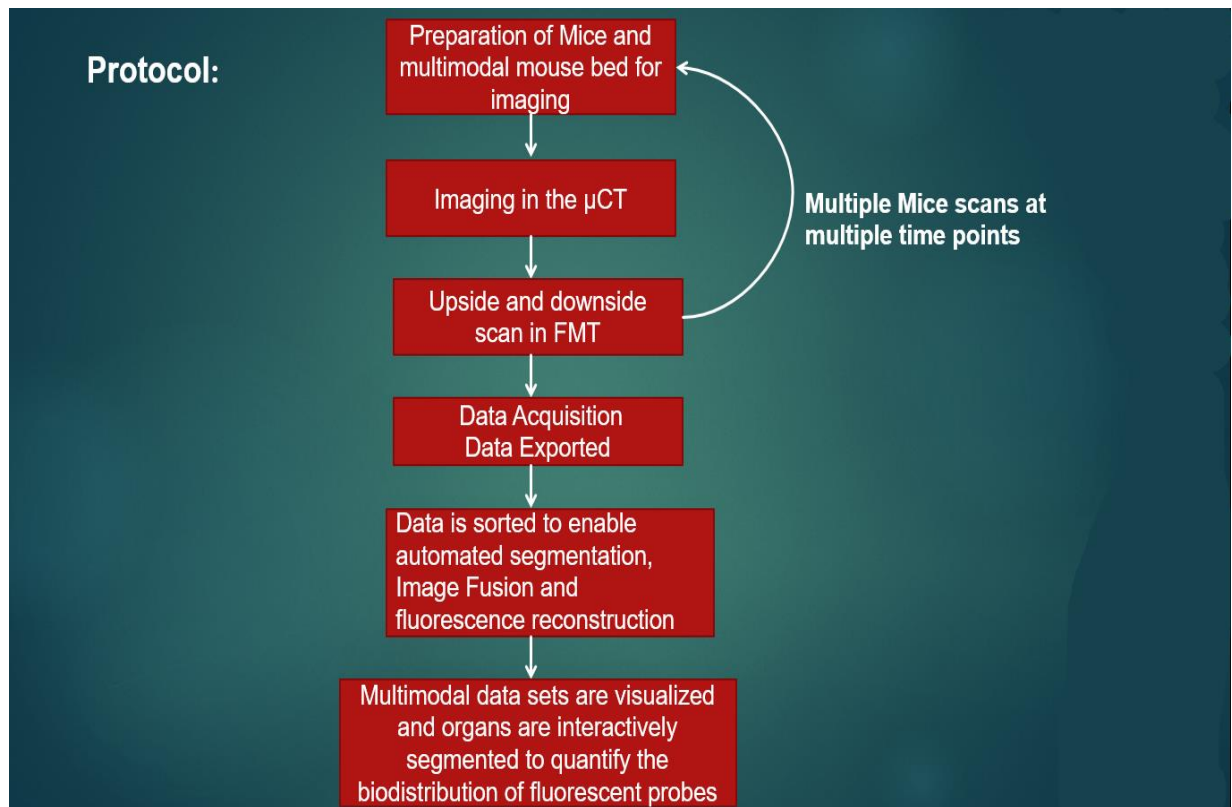


Fig 2.7 Flow Chart summarizing the Protocol

2.2.1 Image Fusion and Reconstruction

The acquired data sets are sorted to enable the automated image fusion and fluorescence reconstruction. Reconstruct group (FMT) in the Imalytics Preclinical software to perform the fluorescence reconstruction including the generation of absorption and scattering maps the processing is GPU-accelerated. Knowledge on optical scattering and absorption, which are highly heterogeneous in different (mouse) tissues will help in proper quantitative fluorescence reconstruction.

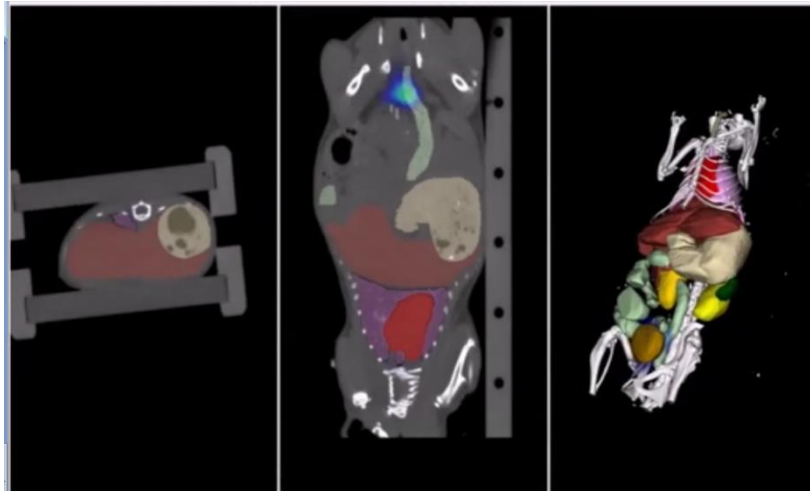


Fig 2.8 Organ segmentation using Imalytics Preclinical

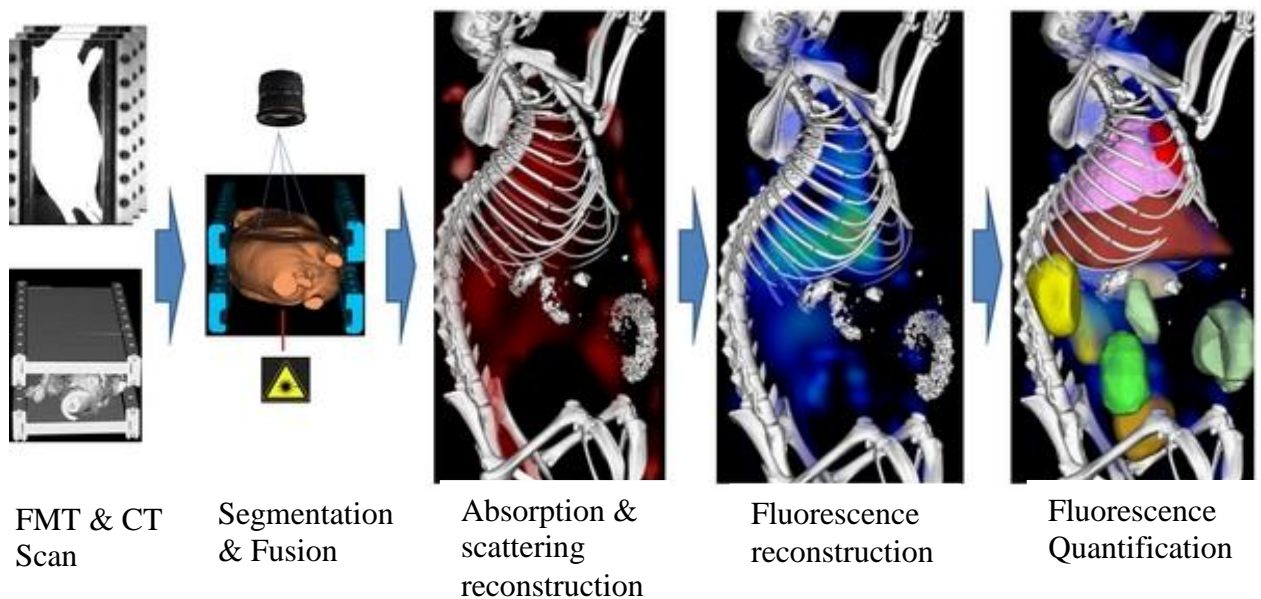


Fig 2.9 Complete process from Scanning to the reconstruction

The shape of the mouse and the heterogeneous absorption and scattering maps are automatically estimated using the micro-CT Data. These parameters are required for quantitative fluorescence reconstruction. To measure the biodistribution of the fluorescence and to extract quantitative measurements from the image data, organ segmentation is needed and it is done using Imalytics Preclinical from the μ -CT Data. Fig 2.8 shows the organ segmentation and Fig 2.9 shows the complete process from CT scanning to reconstruction.

2.3 X-RAY GENERATION

X-ray Tube is where the x-ray is actually generated. X-ray generation is an indirect process unlike tungsten filament lamp or LED lamps in the circuit. There is a cathode that gets heated up from which a lot of electrons are flowing out. As these electrons go towards the anode, they strike the anode surfaces which result in the emission of X-rays. But unlike lasers, they are not a concentrated beam instead they are a Divergent beam. This actually has an advantage as we can produce different levels of magnification and it behaves as a point source of light which emits in all directions isotopically. When an image is placed in front of the X-ray we get a magnified image of the film behind it. Fig 2.10 shows the X-ray Tube working.

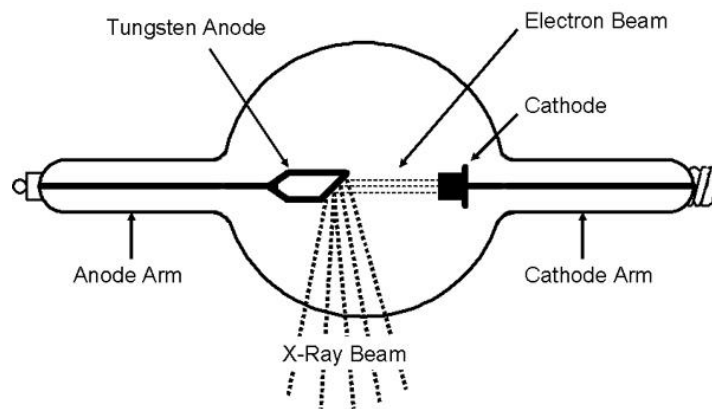


Fig 2.10 Working of X-ray tube

An X-ray is usually generated when the electron gets absorbed in the nucleus of the atom of the anode which generates the highest amount of energy, else it goes close to the nucleus of the atom and gets deflected. But for that also to happen the electron must strike a very heavy metal

atom. The electron has to pass through multiple levels of valence electrons, only then the absorption or the deflection takes place.

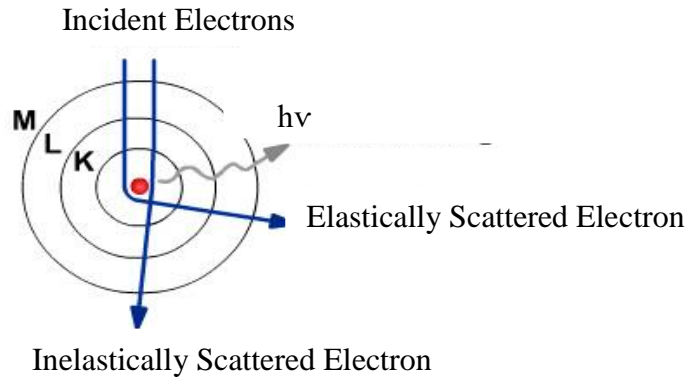


Fig 2.11 Absorption or Scattering of the Cathode electron by the anode atom and release of photon

Since the Kinetic energy of the electron is suddenly reduced, it has to be dissipated in some way to follow the law of conservation of energy. That is the unit of X-ray photon that is released. Fig 2.8 shows the electron from the cathode getting absorbed or getting deflected when it reaches the anode atom.

$$E_{\text{kin}} = \frac{1}{2} m_e v^2 = e \cdot U \quad (1)$$

$$E_{\text{kin}} = E_{\text{max}} = hc/\lambda_{\text{min}} \quad (2)$$

The X-ray photon released is guided based on the above two equations. The electron travels at a velocity 'v' due to the electric field 'U'. So the total Kinetic field equals the electric field times the charge of the electron. If the electron is completely absorbed in the anode atom then the X-ray photon that is generated has the wavelength λ_{min} which actually corresponds to the maximum energy release by the photon. If the electron is partially absorbed or deflected the energy of the photon released is much lower than the which means that the wavelength will be more than the λ_{min} .

If the anode is made out of complete Tungsten and if the electron strikes and the X-rays are generated then the trend of relative photon flux is as shown in the graph in Fig 2.8

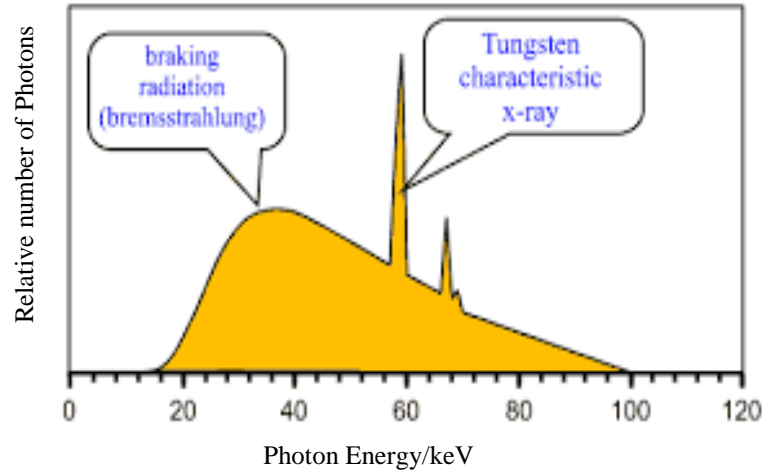


Fig 2.12 Tungsten X-ray generation energy graph

A modal maximum is observed and two peaks are observed that are greater than the modal maximum. The first peak corresponds to the K^{th} shell energy which means that the electron just goes to the K^{th} shell and takes a place in K^{th} shell and it releases energy equivalent to 69.5keV and next transition is between K^{th} shell to $K-1$ the shell and releases energy equivalent to 57.4keV. This observation is important because different kinds of matter can be discriminated using different energy bands.

2.4 μ -CT INSTRUMENTATION

X-ray produces a very divergent beam, not a single focused beam. Due to this, we have an array of detectors each of which has a line equation which can be solved so as to get the projection. This can be performed on 2D space. After which the source and detector array can be rotated around the object which will result in multiple projections in order to reconstruct the whole object. The projections that are derived can be used to backtrack and iteratively reconstruct the object.

2.5 μ -CT RECONSTRUCTION

Image reconstruction in CT is a mathematical process that generates tomographic images from X-ray projection data acquired at many different angles around the object. The whole object that is being imaged is assumed as a grid with different points and each grid has its own attenuation.

Now if there is a whole beam that is passing through the grid, the projection is calculated. The end result is a set of linear equations which have to be inverted and solved.

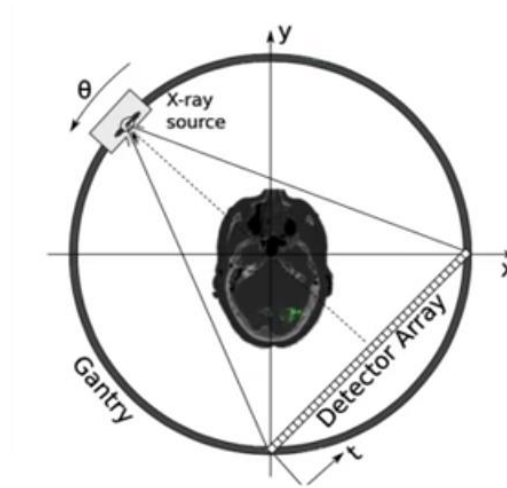


Fig 2.13 Figure showing μ -CT instrumentation

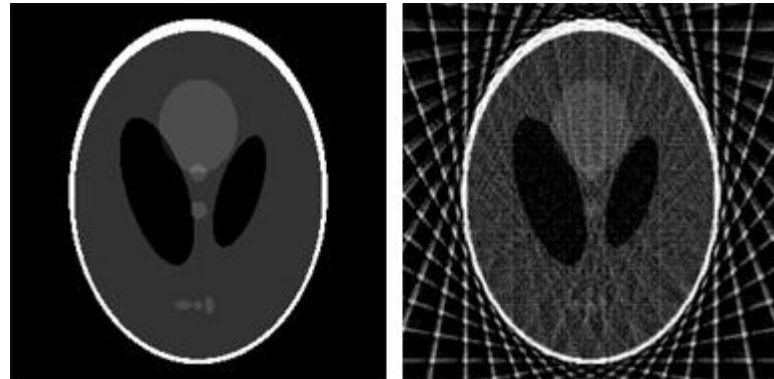


Fig 2.14 Radon reconstruction using the Shepp-Logan phantom image

2.6 UNDERSTANDING THE ABSORPTION RECONSTRUCTION AND OPTICAL SCATTERING IN μ -CT

It is very important to understand the absorption reconstruction and optical scattering for effective design of the solution.

2.6.1 ABSORPTION RECONSTRUCTION

μ -CT relies on the X-ray flux measurements (projections) from different directions (angles). An X-ray beam passes through a planar slice of the body only in directions that are parallel to the plane of the slice, and the intensities of the attenuated X-rays are measured by a detector in the opposite side. Those measures are called projections as shown in Fig 2.7.

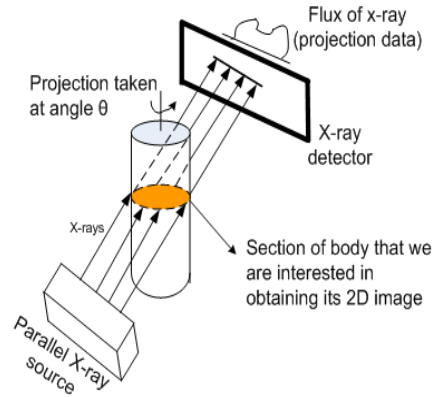


Fig 2.15 Working of X-ray computed tomography

The intensity of the X-ray is attenuated when it goes through a material following a decreasing exponential law. Attenuation coefficient depends on the density and characteristics of the material, hence it is possible to obtain an image of the body's slice by reconstructing the attenuation coefficient function from the projections and then imaging the attenuation coefficient.

The problem of image reconstruction from projections can be considered as a system of linear equations of the form:

$Ax \approx b$, where the system matrix A simulates CT operation and is a sparse symmetric square matrix; its elements depend on the projection number and the angle at which the projections have been acquired. The values of the column matrix x represent the intensities of the image, and the column matrix b represents projections collected by a scanner as shown in Fig 2.8.

For a given angle, we assume that the number of projections ranges from 1 to m . For k different angles, in the above equation, b has $M = mxk$ elements, x has N elements, and A is an $M \times N$ sparse positive symmetric square matrix.

In the paper 'Absorption Reconstruction Improves Biodistribution Assessment of Fluorescent Nanoprobes Using Hybrid Fluorescence-mediated Tomography' [2], the whole-

animal absorption map was reconstructed by iterative minimization of the difference between predicted and measured excitation boundary values.

$$\begin{bmatrix} A11 & A12 & . & . & . & A1n \\ A21 & A22 & . & . & . & A2n \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ An1 & An2 & . & . & . & Ann \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ . \\ . \\ . \\ xn \end{bmatrix} = \begin{bmatrix} b1 \\ b2 \\ . \\ . \\ . \\ bn \end{bmatrix}$$

Fig 2.16 $Ax=b$ represented in matrix format

The absorption map was reconstructed using a model based iterative image reconstruction. The theoptical forward model was implemented using a finite difference mesh (mesh size 0.56mm) which approximates the diffusion equation. This results in a sparse symmetric positive matrix (μa), which depends on the absorption map μa and the fixed pre-determined scattering map. The three-dimensional absorption map μa , treated as a flattened vector, was reconstructed by iteratively minimizing a differentiable nonlinear cost function $f(\mu a)$ expressing the squared difference between predicted and measured boundary values for all K source points. The cost function is a mathematical function that maps the reconstructed image into the Real number line.

2.7. EXISTING SYSTEM

Filtered back-projection is one of the oldest and a technology still in use. The following section is a study of this method.

2.7.1 FILTERED BACK PROJECTION

Filtered back-projection was the algorithm used first by almost all commercially available CT scanners. It uses the Fourier transform method, where at one particular projection it is backtracked in order to get the inverse of the Fourier transform. It comes close to the actual attenuation map at every backtracking.

Filtered back-projection demands fewer computational resources, while iterative methods generally produce less number of errors in the reconstruction at a higher computational

cost. Filtered back-projection methods are based on analytical algorithms which use the inverse Fourier transform and they need a set of projections equally separated. However, in CT it is common to find undersampled set of not equally spaced projections. In these cases, images reconstructed with conventional FBP algorithm are highly degraded due to insufficient and noisy projections. However, the advance in new computer architectures is doing that algebraic methods can be considered as reconstruction method in commercial scanners.

2.7.2 DISADVANTAGE OF FBP

- Since it is in the Fourier domain it has a higher computational complexity
- Since the Fourier transforms are not infinite, they just perform a fast Fourier transform or discrete Fourier transforms which lead to limiting the number of frequencies considered.
- In CT it is common to find undersampled set of not equally spaced projections. In these cases, images reconstructed with conventional FBP algorithm are highly degraded due to insufficient and noisy projections.

2.8. PROPOSED SYSTEM

After exploring the Filtered back-projection. The following section is a study of the proposed effective method. Iterative reconstruction refers to iterative algorithms used to reconstruct 2D and 3D images in certain imaging techniques.

For example, in computed tomography an image must be reconstructed from projections of an object. Here, iterative reconstruction techniques are usually a better, but computationally more expensive alternative to the common filtered back projection (FBP) method, which directly calculates the image in a single reconstruction step. In recent research works, scientists have shown that extremely fast computations and massive parallelism is possible for iterative reconstruction, which makes iterative reconstruction practical for commercialization.

2.8.1 SOLVING A SPARSE LINEAR SYSTEM THROUGH MINIMIZATION

A system of linear equations is called sparse if only a relatively small number of its matrix elements a_{ij} are nonzero. It is wasteful to use general methods of linear algebra on such problems because most of the $O(N^3)$ arithmetic operations devoted to solving the set of equations or inverting the matrix involve zero operands.

Cost functions in μ -CT are considered to have the property that their minimum/minima are close to the true nature of patient/phantom anatomy. The algorithm is a numerical method that finds the image that minimizes the cost function. Absorption reconstruction is a nonlinear problem, particularly for strong absorption, requiring a computationally demanding iterative method. They start with a coarse initial guess and refine it over and over in such a way that the refinement process lowers values of the Cost function.

The iterative algorithms execute iterations until convergence criteria are met. The number of iterations required by an iterative algorithm to meet certain convergence criteria roughly defines its convergence rate. Iterative methods generally produce less number of errors in the reconstruction at a higher computational cost. These methods are more flexible and provide more accuracy. Iterative methods do not require complete data collection and to provide the optimal reconstruction in noisy conditions in the image.

2.8.2 KRYLOV METHODS

Modern iterative methods for finding one (or a few) eigenvalues of large sparse matrices or solving large systems of linear equations avoid matrix-matrix operations, but rather multiply vectors by the matrix and work with the resulting vectors. Starting with a vector, b , one computes, then one multiplies that vector by to find and so on.

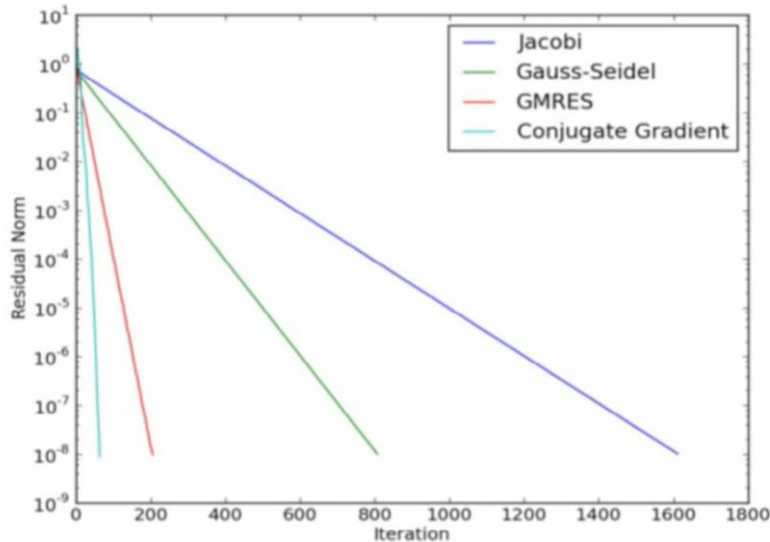


Fig 2.17 Graph shows the convergence rate of different iterative algorithms

All algorithms that work this way are referred to as Krylov subspace methods; they are among the most successful methods currently available in numerical linear algebra. Krylov methods are the most popular iterative method for solving large systems of linear equations and are effective for systems of the form $Ax=b$ where x is an unknown vector, b is a known vector, and A is a known, square, symmetric, positive-definite.

Krylov Methods:

- Conjugate Gradient method: In the conjugate gradient method, a better descent direction is produced by combining the gradient at the current iterate with the descent direction at the previous iterate. Solves the symmetric, positive-definite linear system $A x = b$ with preconditioner M .
- Biconjugate Gradient method: Solves the linear system $A x = b$ with preconditioner M . Biconjugate Gradient Stabilized method: Solves the linear system $A x = b$ with preconditioner M .
- Conjugate Residual method: A and M must be symmetric and semi-definite.
- GMRES method: Solves the nonsymmetric, linear system $A x = b$ with preconditioner M .

2.8.3 USING GPU TO ACCELERATE COMPUTATION OF SOLUTION FOR SPARSE MATRIX LINEAR SYSTEM

High convergence rate alone is not expected to be sufficient to make iterative algorithms practical. Acceleration of iterative reconstruction is an active area of research, especially on graphical processing units. Our aim is to take advantage of the massive computing power of GPU in order to reconstruct μ -CT images with a higher resolution without losing quality.

The cost function $f(\mu_a)$ has to be minimized with respect to the absorption map using the nonlinear conjugate gradient method and the required gradient computations should be performed using algorithmic differentiation with GPU accelerated sparse vector and matrix operations. Due to the computational load, to perform 3D reconstructions of whole animals, we compute the gradients using algorithmic differentiation with GPU-accelerated sparse vector and matrix operations, where the derivatives are propagated backwards through a linear solver at each iteration.

2.9 USECASE ANALYSIS

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. The purpose of use case diagram is to capture the dynamic aspect of a system. A use case is a list of steps, typically defining interactions between a role (known in UML as an "actor") and a system, to achieve a goal. The actor can be a human or an external system the purpose of use case diagrams can be as follows:

- Used to gather requirements for a system.
- Used to get an outside view of a system.
- Identify external factors influencing the system.
- Identify internal factors influencing the system.
- Show the interacting among the requirements are actors.

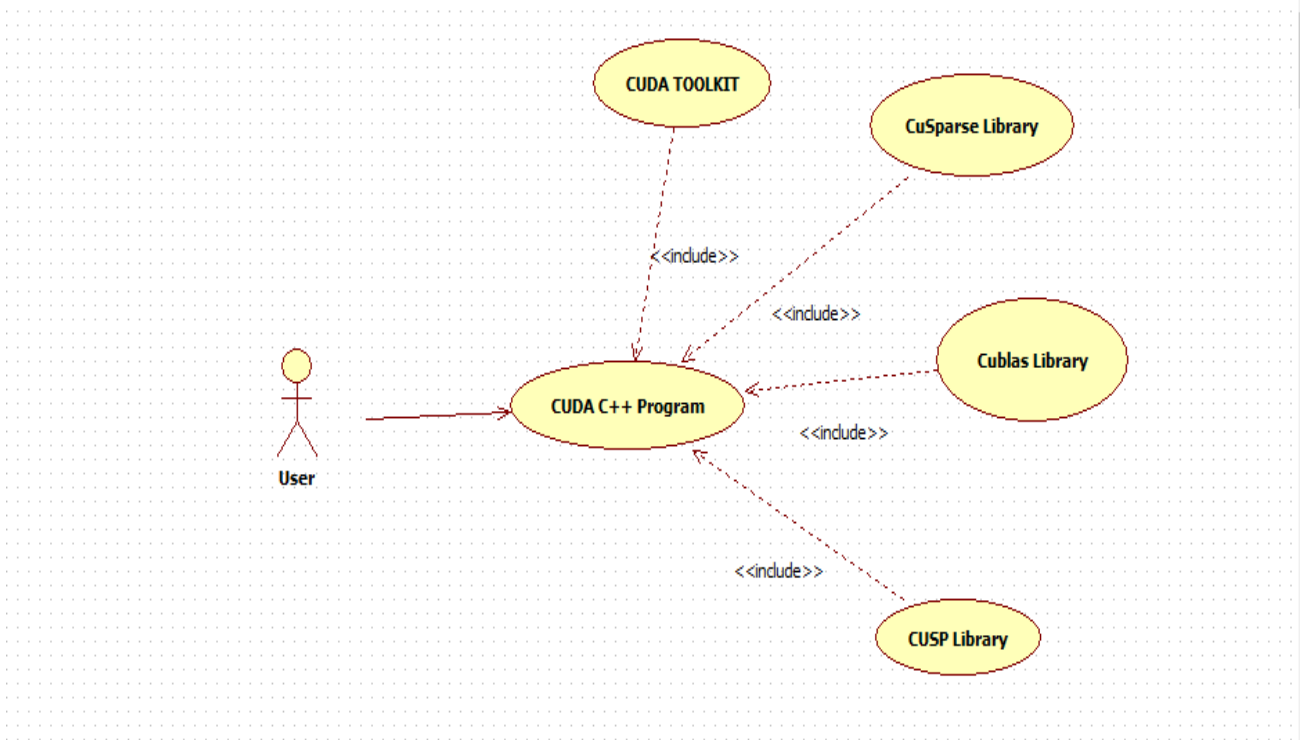


Fig 2.18 Use Case Diagram of the libraries and modules used for the CUDA program

UML Use Case Diagrams can be used to describe the functionality of a system in a horizontal way. That is, rather than merely representing the details of individual features of your system, UCDs can be used to show all of its available functionality. It is important to note, though, that UCDs are fundamentally different from sequence diagrams or flow charts because they do not make any attempt to represent the order or number of times that the actions and sub-actions of the system should be executed. That is, rather than merely representing the details of individual features of your system, UCDs can be used to show all of its available functionality. Use case diagrams are considered for high-level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases.

2.10 SEQUENCE DIAGRAM

The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. The main purpose of a sequence diagram is to define event sequences that result in some desired outcome. The focus is less on messages themselves and more on the order in which messages occur.

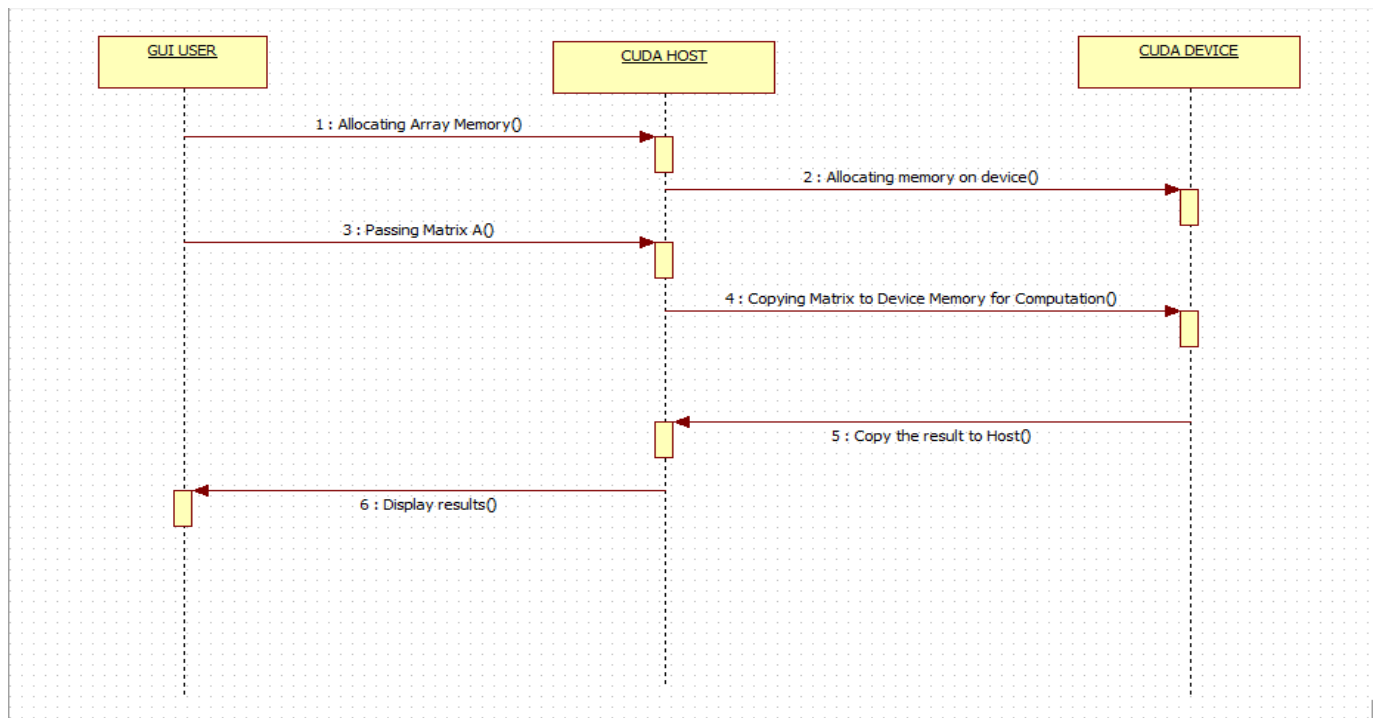


Fig 2.19 Sequence Diagram

Most sequence diagrams will communicate what messages are sent between a system's objects as well as the order in which they occur. The diagram conveys this information along the horizontal and vertical dimensions: the vertical dimension shows, top-down, and the time sequence of messages/calls as they occur, and the horizontal dimension shows, left to right, and the object instances that the messages are sent to. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event

Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. When an object is destroyed (removed from memory), an X is drawn on top of the lifeline, and the dashed line ceases to be drawn below it (this is not the case in the first example though).

It should be the result of a message, either from the object itself, or another. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response.

Asynchronous calls are present in multi-threaded applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML). Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. A message sent from outside the diagram can be represented by a message originating from a filled-in circle (found a message in UML) or from a border of the sequence diagram (gate in UML). The sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. Much like the class diagram, developers typically think sequence diagrams were meant exclusively for them.

However, an organization's business staff can find sequence diagrams useful to communicate how the business currently works by showing how various business objects interact. Besides documenting an organization's current affairs, a business-level sequence diagram can be used as a requirements document to communicate requirements for a future system

implementation. During the requirements phase of a project, analysts can take use cases to the next level by providing a more formal level of refinement.

Sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between a numbers of lifelines. UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes.

2.11 DETAILED STUDY AND ANALYSIS OF FUNCTIONALITIES

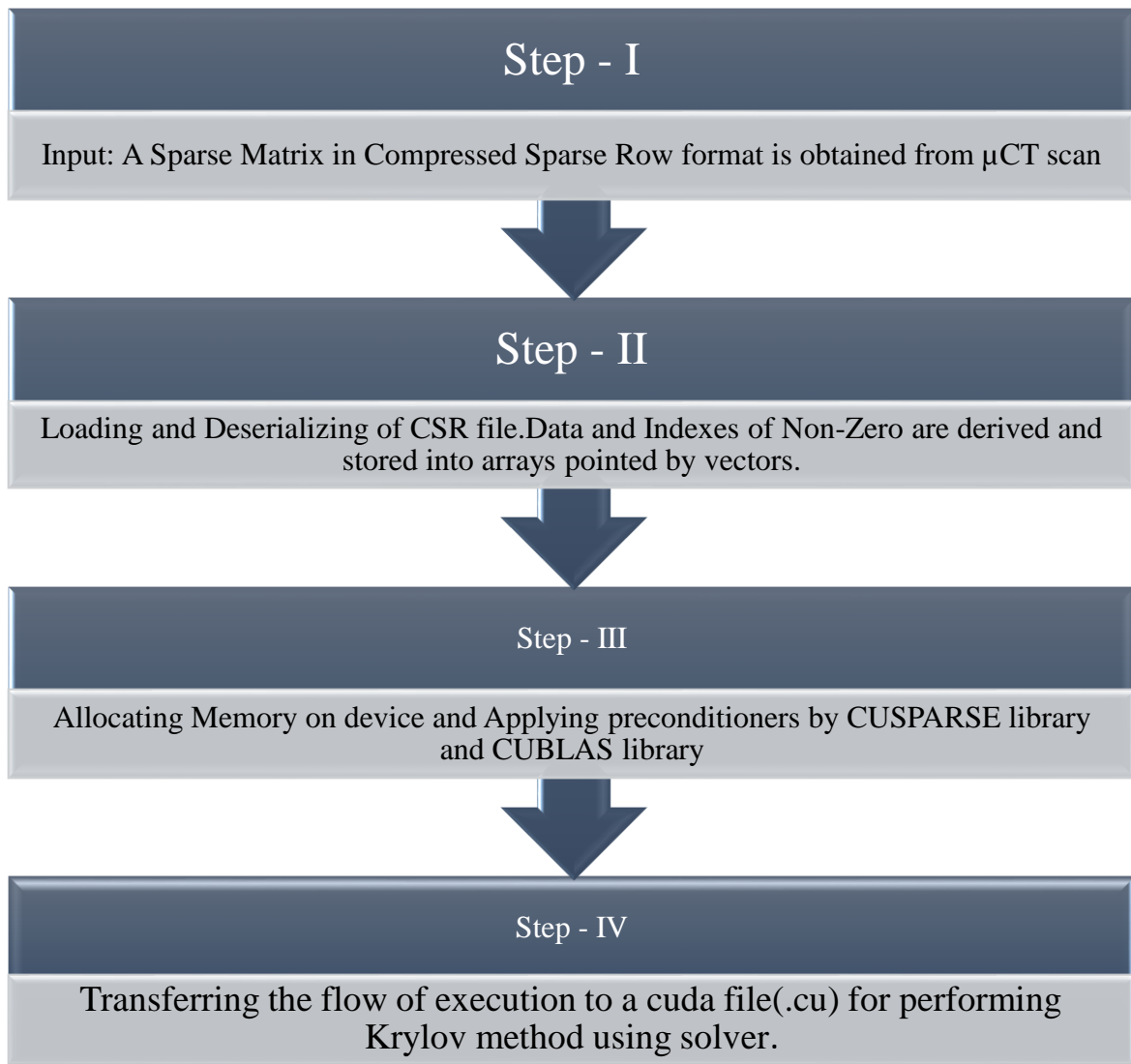


Fig 2.20 Flow Chart showing the Functionalities involved

2.12 REQUIREMENTS SPECIFICATION

Requirement specification gives a list of expectations out of the created system.\

2.12.1 User Requirements

- Industry standard programming practices on CUDA C++
- The system must allow easy access to information
- The system categorizes the different tools in a module based on their functionalities
- The system provides help relating to tools used in each module

2.12.2 Functional Requirements

- Anyone who understands the CUDA C++ should be able to use this system
- The System should be able to perform computation for CSR format
- The System should conform to all specifications and Display Results

2.12.3 Non Functional Requirements

- The System should be efficient, reliable and secure throughout
- The System should be easy to modify to suit the user changing demands

2.12.4 Software Requirements

- Development tool: Microsoft Visual Studio-2015
- Operating System: Windows 8

2.12.5 Hardware Requirements

Table 2.1 Hardware Requirements

| Processor | RAM | Hard Disk Memory | GPU |
|-------------------------------------|-------------------|------------------|-----------------------------|
| Intel Core i5 Processor, 3.20GHz | 8GB (recommended) | >1GB | NVIDIA GeForce GTX Titan |

2.13 CONSTRAINTS

- The Input Matrix File format can only be compressed sparse row (CSR)
- The PC must have an NVIDIA Graphic Card of at least 2GB
- Visual Studio 2015 or after and CUDA 2.0 Toolkit or after is recommended
- This solution only performs minimization using the conjugate gradient method.

CHAPTER 3

SYSTEM DESIGN

This Chapter deals with the fabrication of solution with a module-wise explanation of the significant functions.

3.1 ARCHITECTURE OVERVIEW

NVIDIA CUDA technology leverages the massively parallel processing power of NVIDIA GPUs. The CUDA architecture is a revolutionary parallel computing architecture that delivers the performance of NVIDIA's world-renowned graphics processor technology for general purpose GPU Computing. With the CUDA architecture and tools, developers are achieving dramatic speedups in fields such as medical imaging and natural resource exploration and creating breakthrough applications in areas such as image recognition and real-time HD video playback and encoding. CUDA enables this unprecedented performance via standard APIs and high-level programming languages such as C/C++, Fortran, Java, Python, and the Microsoft .NET Framework. CUDA Architecture is shown in Fig 3.1.

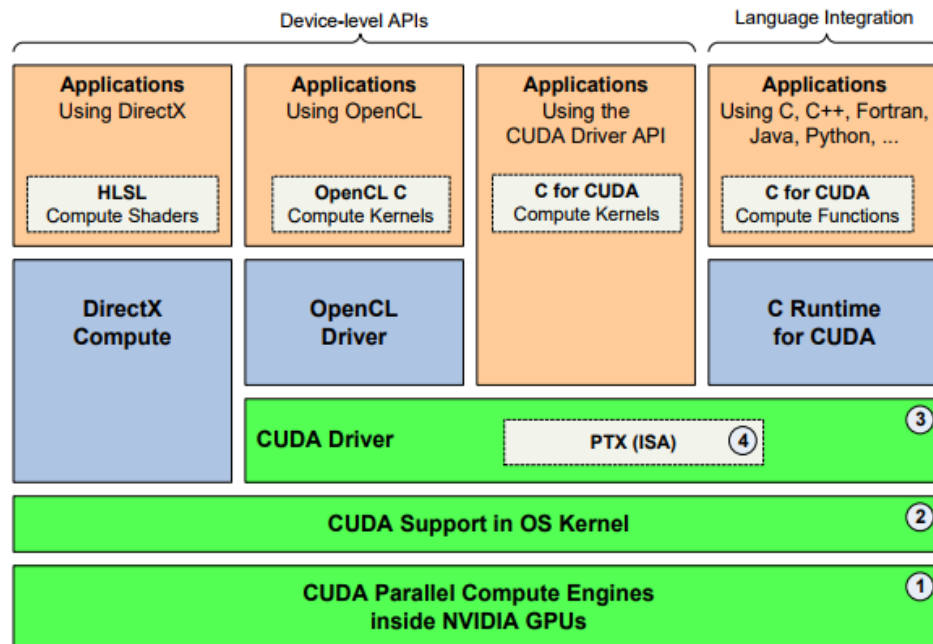


Fig 3.1 CUDA Architecture

The CUDA Architecture consists of several components:

- Parallel compute engines inside NVIDIA GPUs
- OS kernel-level support for hardware initialization, configuration, etc
- User-mode driver, which provides a device-level API for developers
- PTX instruction set architecture (ISA) for parallel computing kernels and functions

The CUDA Software Development Environment provides all the tools, examples and documentation necessary to develop applications that take advantage of the CUDA architecture.

Table 3.1 CUDA Toolkit

| | |
|---------------|---|
| Libraries | Advanced libraries that include BLAS, FFT, and other functions optimized for the CUDA architecture |
| C++ Runtime | The C++ Runtime for CUDA provides support for executing standard C++ functions on the GPU |
| Tools | NVIDIA C Compiler (nvcc), CUDA Debugger (cudagdb), CUDA Visual Profiler (cudaprof), and other helpful tools |
| Documentation | Includes the CUDA Programming Guide, API specifications, and other helpful documentation |

3.2 HETEROGENEOUS COMPUTING

The CUDA programming model is a heterogeneous model in which both the CPU and GPU are used. In CUDA, the host refers to the CPU and its memory shown in Fig 3.3, while the device refers to the GPU and its memory shown in Fig 3.2. Code run on the host can manage memory on both the host and device and also launches kernels which are functions executed on the device. These kernels are executed by many GPU threads in parallel. Given the heterogeneous nature of the CUDA programming model, a typical sequence of operations for a CUDA C program is:

- Declare and allocate host and device memory.
- Initialize host data.
- Transfer data from the host to the device.
- Execute one or more kernels.
- Transfer results from the device to the host.



Fig 3.2 NVIDIA GPU -DEVICE

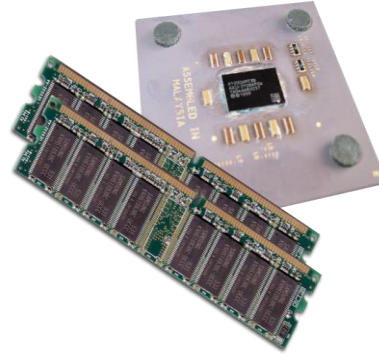


Fig 3.3 CPU RAM -HOST

3.3 PROCESSING FLOW OF GPU

The process flow of the GPU can be summarized as below Fig 3.4 to Fig 3.7

Step1: Device and Host Memory:

There are two types of Memory used and hence it is called heterogeneous computing. The CPU memory and the GPU memory.

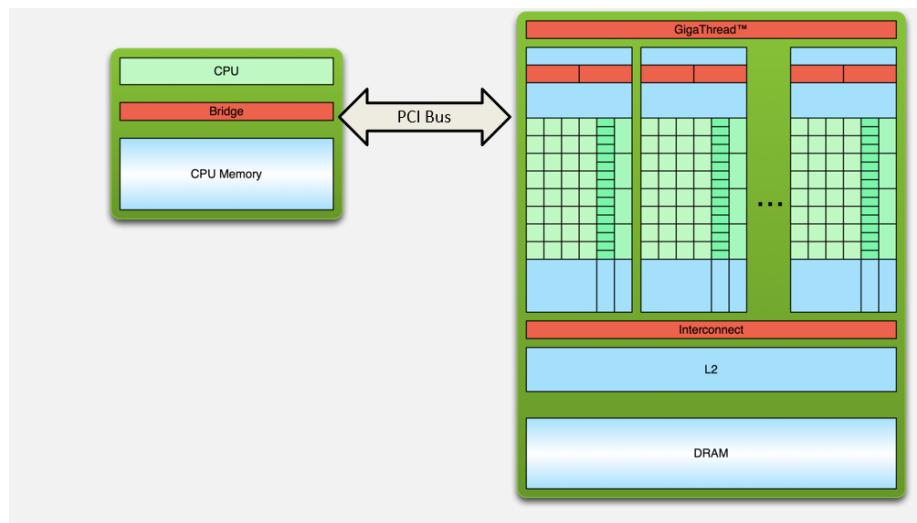


Fig 3.4 Device and Host Memory

Step 2: Copy of Input Data from Host to Device

Copying the data and the program from the host to the device is performed after allocating enough memory on the Device.

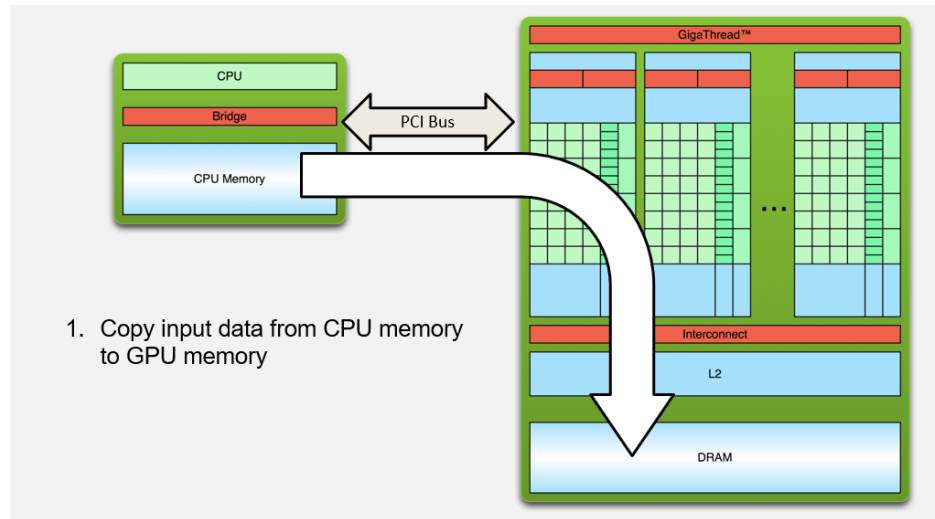


Fig 3.5 Data copied from CPU to GPU

Step 3: Load Device program and execute

The program is loaded on to the GPU and the execution of the program takes place on the GPU.

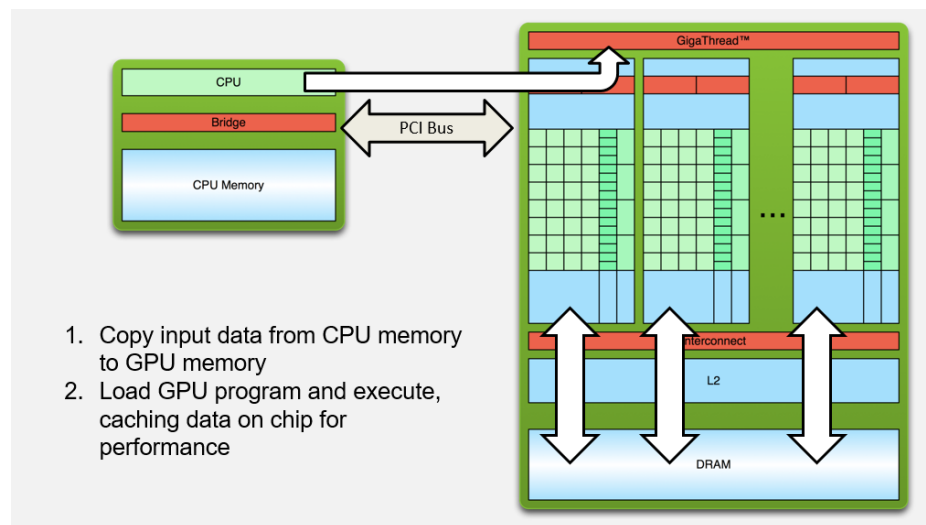


Fig 3.6 Loading and execution of GPU program

Step 4: Copy Resulting Data back to Host memory

The result after computation is copied back on to the CPU memory.

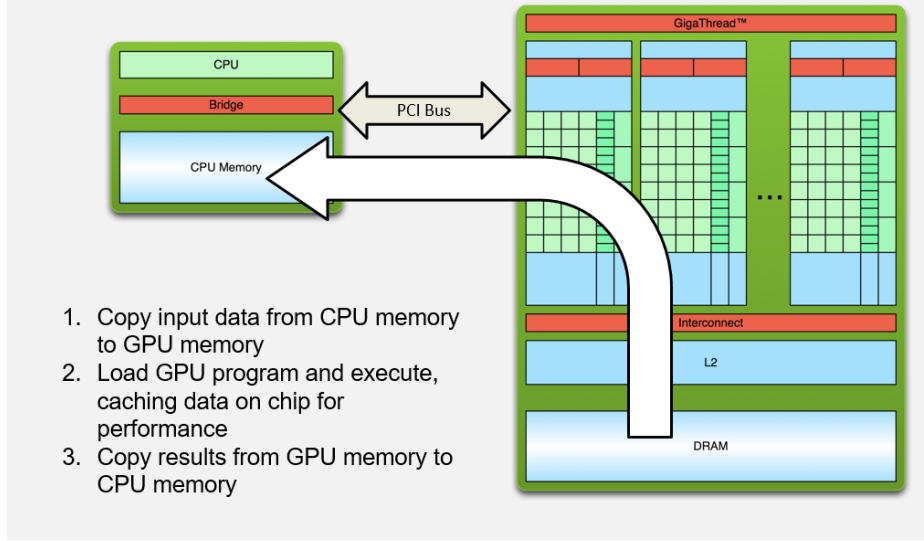


Fig 3.7 Copy Results to CPU memory

3.4 COMPRESSED SPARSE ROW FORMAT FOR SPARSE MATRIX COMPUTATION

The compressed sparse row (CSR) or compressed row storage (CRS) format represents a matrix M by three (one-dimensional) arrays, that respectively contain nonzero values, the extents of rows, and column indices. It is similar to COO, but compresses the row indices, hence the name. This format allows fast row access and matrix-vector multiplications (Mx).

The CSR format has been in use since at least the mid-1960s, with the first complete description appearing in 1967. The CSR format stores a sparse $m \times n$ matrix M in row form using three (one-dimensional) arrays (A , IA , JA). Let NNZ denote the number of nonzero entries in M . The compressed row storage (CRS) format puts the subsequent nonzeros of the matrix rows in contiguous memory locations.

Assuming we have a nonsymmetric sparse matrix A , we create three vectors: one for floating point numbers (val) and the other two for integers (col_ind , row_ptr). The val vector stores the values of the nonzero elements of the matrix A as they are traversed in a row-wise fashion. The col_ind vector stores the column indexes of the elements in the val vector. That is, if $val(k)=a_{ij}$, then $col_ind(k)=j$. The row_ptr vector stores the locations in the val vector that start a row; that is, if $val(k)=a_{ij}$, then $row_ptr(i) < k < row_ptr(i+1)$. By convention, we define $row_ptr(n+1) = nnz+1$, where nnz is the number of nonzeros in the matrix A . The storage savings for this approach is significant. Instead of storing n^2 elements, we need only $2nnz+n+1$ storage locations.

The example in Fig 3.8 shows how the non-zero values in Matrix A is stored in value array and the column indices stored in col_ind and row pointer stored in row_ptr.

$$\mathbf{A} = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}$$

| | | | | | | | | | | | | | | | | | | | |
|----------------|-----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|----------|----------|-----------|
| val | 10 | -2 | 3 | 9 | 3 | 7 | 8 | 7 | 3 | 8 | 7 | 5 | 8 | 9 | 9 | 13 | 4 | 2 | -1 |
| col_ind | 0 | 4 | 0 | 1 | 5 | 1 | 2 | 3 | 0 | 2 | 3 | 4 | 1 | 3 | 4 | 5 | 1 | 4 | 5 |

| | | | | | | |
|----------------|----------|----------|----------|----------|-----------|-----------|
| row_ptr | 0 | 2 | 5 | 8 | 12 | 16 |
|----------------|----------|----------|----------|----------|-----------|-----------|

Fig 3.8 Example of a Sparse Matrix stored Sparse Matrix

3.5 SERIALIZATION AND DESERIALIZATION

In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later (possibly in a different computer environment).

When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object. For many complex objects, such as those that make extensive use of references, this process is not straightforward. Serialization of object-oriented objects does not include any of their associated methods with which they were previously linked. This process of serializing an object is also called marshalling an object. The opposite operation, extracting a data structure from a series of bytes, is deserialization (which is also called unmarshalling).

3.5 ITERATIVE SOLVERS USED

There are many types of iterative solvers, but for this application, only Krylov methods are used because they are effective for sparse, symmetric, square, positive matrix.

3.5.1 Conjugate Gradient

In mathematics, the conjugate gradient method is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is symmetric and positive-definite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition.

Large sparse systems often arise when numerically solving partial differential equations or optimization problems. The conjugate gradient method can also be used to solve unconstrained optimization problems such as energy minimization. It was mainly developed by Magnus Hestenes and Eduard Stiefel. In the conjugate gradient method, a better descent direction is produced by combining the gradient at the current iterate with the descent direction at the previous iterate. Solves the symmetric, positive-definite linear system $Ax = b$ with preconditioner M .

3.5.2 Preconditioned Conjugate Gradient

In mathematics, preconditioning is the application of a transformation, called the preconditioner that conditions a given problem into a form that is more suitable for numerical solving methods. Preconditioning is typically related to reducing a condition number of the problem. The preconditioned problem is then usually solved by an iterative method.

In most cases, preconditioning is necessary to ensure fast convergence of the conjugate gradient method. The preconditioner matrix M has to be symmetric positive-definite and fixed, i.e., cannot change from iteration to iteration. If any of these assumptions on the preconditioner is violated, the behaviour of the preconditioned conjugate gradient method may become unpredictable.

3.5.2 GMRES Method

In mathematics, the generalized minimal residual method (usually abbreviated GMRES) is an iterative method for the numerical solution of a nonsymmetric system of linear equations. The method approximates the solution by the vector in a Krylov subspace with minimal residual. The Arnoldi iteration is used to find this vector.

The GMRES method was developed by Yousef Saad and Martin H. Schultz in 1986. GMRES is a generalization of the MINRES method developed by Chris Paige and Michael Saunders in 1975. GMRES also is a special case of the DIIS method developed by Peter Pulay in 1980. DIIS is also applicable to non-linear systems.

3.5.2 Conjugate Residual Method

The conjugate residual method is an iterative numeric method used for solving systems of linear equations. The conjugate residual method differs from the closely related conjugate gradient method primarily in that it involves more numerical operations and requires more storage, but the system matrix is only required to be Hermitian, not symmetric positive definite.

3.5.3 Bi-Conjugate Gradient Method

In mathematics, more specifically in numerical linear algebra, the biconjugate gradient method is an algorithm to solve systems of linear equations. Unlike the conjugate gradient method, this algorithm does not require the matrix A to be self-adjoint, but instead one needs to perform multiplications by the conjugate transpose A^* .

3.5.4 Bi-Conjugate Stabilized Method

In numerical linear algebra, the biconjugate gradient stabilized method, often abbreviated as BiCGSTAB, is an iterative method developed by H. A. van der Vorst for the numerical solution of nonsymmetric linear systems. It is a variant of the biconjugate gradient method (BiCG) and has faster and smoother convergence than the original BiCG as well as other variants such as the conjugate gradient squared method (CGS). It is a Krylov subspace method.

CHAPTER 4

SYSTEM IMPLEMENTATION

The Chapter deals with the implementation of a solution with a module-wise explanation of the significant functions.

4.1 MODULE IMPLEMENTATION

Modules involved in the implementation are summarized below:

- Mouse Matrix imaged in a CT scan is taken as Matrix A
- Importing Sparse Matrix A in CSR format
- Deserializing the CSR object to get the three arrays which have all the CSR data
- Dense vector be provided by the user
- Allocation of device vector and copy the matrix and vectors from host to device
- Initialize the cuSparse library for Matrix operations
- Create and set up matrix descriptor
- Exercise the Krylov methods from CUSP library in separate routine
- Display the results and number of iterations required
- Destroy the matrix descriptor
- Release the resources allocated on the GPU

4.2 GRAPHICAL PROCESSING UNIT

Computer graphic cards, such as the NVIDIA GeForce series and the GTX series, are conventionally used for display purpose on desktop computers. Special GPUs card dedicated to scientific computing, like the NVIDIA GeForce GTX Titan card is used in the project to carry out the experiment. Utilizing such a GPU card with tremendous parallel computing ability can considerably elevate the computation efficiency of our algorithm.

4.3 MICROSOFT VISUAL C++

Microsoft Visual C++ (often abbreviated to MSVC) is an integrated development environment (IDE) product from Microsoft for the C, C++, and C++/CLI programming

languages. MSVC is proprietary software; it was originally a standalone product but later became a part of Visual Studio and made available in both trialware and freeware forms. It features tools for developing and debugging C++ code, especially code written for Windows API, DirectX and .NET Framework.

Many applications require redistributable Visual C++ packages to function correctly. These packages are often installed independently of applications, allowing multiple applications to make use of the package while only having to install it once. These Visual C++ redistributable and runtime packages are mostly installed for standard libraries that many applications use.

4.4 CUDA TOOLKIT

CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming. Also, CUDA supports programming frameworks such as OpenACC and OpenCL. When it was first introduced by Nvidia, the name CUDA was an acronym for Compute Unified Device Architecture, but Nvidia subsequently dropped the use of the acronym.

CUDA has several advantages over traditional general-purpose computation on GPUs:

- Scattered reads – code can read from arbitrary addresses in memory
- Unified virtual memory (CUDA 4.0 and above)
- Unified memory (CUDA 6.0 and above)
- Shared memory – CUDA exposes a fast shared memory region that can be shared among threads. This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups.^[14]

- Faster downloads and readbacks to and from the GPU
- Full support for integer and bitwise operations, including integer texture lookups

4.5 LIBRARIES USED

CUDA comes with a software environment that allows developers to use C or C++ as high-level programming languages and overcome the challenge to develop application software that transparently scales its parallelism to leverage the increasing number of processor cores.

CUSP Library, CUBLAS and CUSPARS libraries that allow the user to access the computational resources of NVIDIA Graphical Processing Unit (GPU). The CUBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA®CUDATM runtime.

To use the CUBLAS library, the application must allocate the required matrices and vectors in the GPU memory space, fill them with data, call the sequence of desired CUBLAS functions, and then upload the results from the GPU memory space back to the host. The CUBLAS library also provides helper functions for writing and retrieving data from the GPU.

The NVIDIA® CUDATM CUSPARSE library contains a set of basic linear algebra subroutines used for handling sparse matrices and is designed to be called from C or C++. These subroutines include operations between vector and matrices in sparse and dense format, as well as conversion routines that allow conversion between different matrix formats.

Cusp is a library for sparse linear algebra and graph computations based on Thrust. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems.

The following piece of the code represents the library functions used for computation:

Cusp library:

- `cusparseCreate (&handle_s);`
- `cusparseCreateMatDescr(&descra);`
- `cusparseSetMatType(descra,CUSPARSE_MATRIX_TYPE_GENERAL);`
- `cusparseSetMatIndexBase(descra,CUSPARSE_INDEX_BASE_ZERO); cusparseScsrmmv (handle_s, CUSPARSE_OPERATION_NON_TRANSPOSE, ncol, nrow, 1.0, descra, csc_values, cscColPtr, cscRowInd, d_U, 0.0, d_V);`

- `cublasGetVector (ncol, sizeof(float), d_V, 1, h_V, 1)`

Cusplibrary:

- `void cusp::krylov::bicg(LinearOperator &A, LinearOperator &At, Vector &x, Vector &b, Monitor& monitor, Preconditioner &M, Preconditioner &Mt)`
- `void cusp::krylov::bicgstab(LinearOperator &A, Vector &x, Vector &b, Monitor& monitor, Preconditioner &M)`
- `void cusp::krylov::cg(LinearOperator &A, Vector&x, Vector &b, Monitor &monitor, Preconditioner &M);`
- `void cusp::krylov::cr(LinearOperator &A, Vector&x, Vector &b, Monitor &monitor, Preconditioner &M);`
- `void cusp::krylov::gmres(LinearOperator &A, Vector&x, Vector &b, const size_t restart, Monitor &monitor, Preconditioner &M);`

4.6 IMPLEMENTATION

The CUDA C++ Code for the solution was written using Visual C++. The headers recommended for CUDA programming are included in the program. The matrix given as input is in the form of CSR(Compressed Sparse Row) format for reducing the operations required for Matrix operations. The CSR file is read as input using File operation and using the SparseHostMatrix class the file is deserialized to get the values of three vector from CSR format ie, row_ptr, col_indices and values. These three vectors form the major points of access for Matrix computation.

The Host pointers to each of these vectors are initialized and then again the Matrix A is initialized using LoadSparseHostMatrixCSR routine. They are converted into device pointers. Matrix A is also converted into a device Matrix B.

The values of width, height ie the number of rows and columns and number of nonzeros in the sparse matrix is stored on the host in int variables. A few variables are created on the host for preconditioning steps using CUDA Toolkit. `cudaGetDevice()` is used to get the best possible available CUDA device. Statistics about the CUDA device can be printed for checking if the device is working.

A CUBLAS instance has to be created to handle the CUBLAS context. A CUSPARSE instance has to be created to handle the CUSPARSE context. A Matrix descriptor is created and the properties of the matrix are defined as Sparse, Positive, Symmetric, and Square. Host Memory is allocated to the device using the cudaMalloc. Preconditioning steps such as Matrix-vector multiplication was performed for better computation.

The Device vector with the matrices is sent to a routine called DoSomethingWithCusp(). In this routine, the cusp library is used in the majority of operations. The object of SparseHostMatrix is converted into cusp::csr_matrix. The csr_matrix object is allocated the memory again in the host based on the number of rows, columns and no. of non-zeros.

Values from the device vector B is copied onto the Matrix of cusp format. Then the arrays required for dense vectors x, b are allocated in the device memory using cusp library. After that one by one, the Krylov methods are called using different methods available like cg, bistab etc.

4.7 TESTING AND RESULTS

The Mouse matrix collected from the in-vivo studies from μ CT-FMT at Uniklink is used to perform the testing and to record the results. The CUDA C++ program developed in Microsoft Visual C++ using the libraries such as CUSP compiles perfectly and builds. Now for conducting a study on the performance of different algorithms on the same, the Krylov methods implemented on cusp library is used. For every separate method, the program is modified but the input matrix has remained the same throughout the process.

Initially after the code is built and debugged, the testing had to be done if the Krylov solvers are working and the GPU code is in place and if the libraries are compatible with each other. In such situation using the laboratory generated matrix is a bad idea to test and conform the results. Usage of a standard symmetric matrix such as Poisson matrix is a good idea to test if the solver is working. Thus a Poisson matrix is generated using the cusp library command and given as an input to the routine solving with the solver.

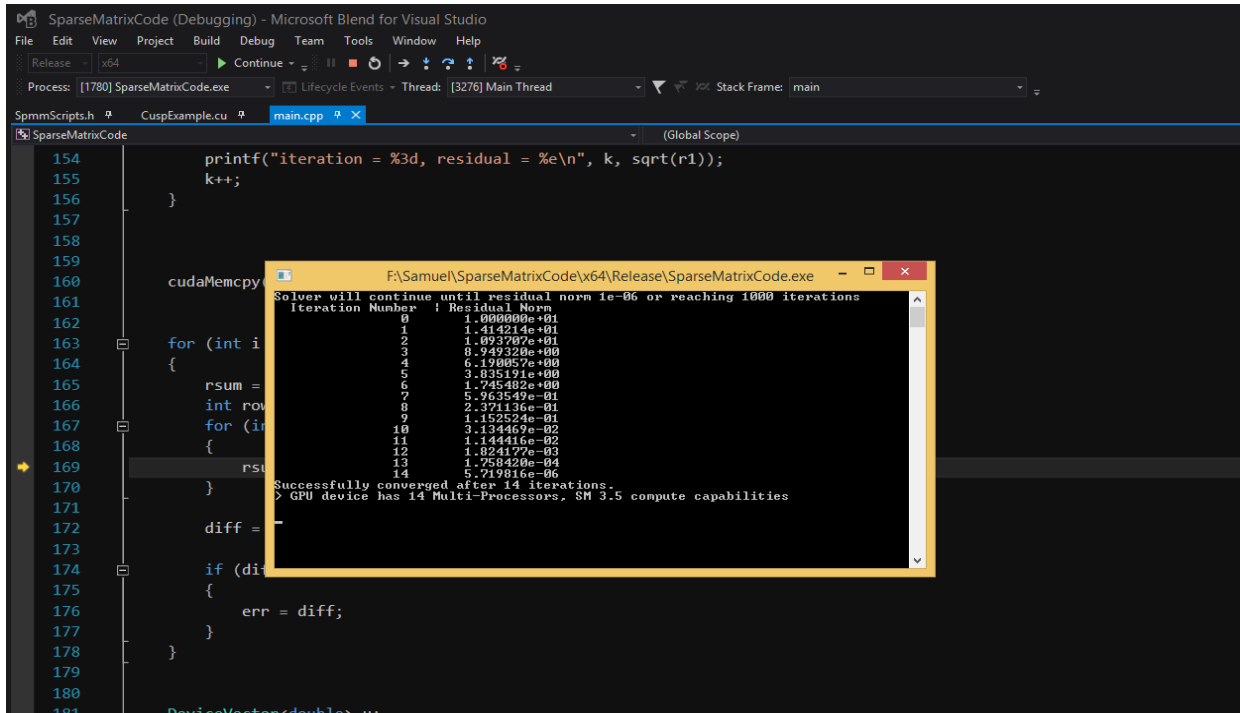


Fig 4.1 Poisson Matrix used for testing

In mathematics, the discrete Poisson equation is the finite difference analog of the Poisson equation. In it, the discrete Laplace operator takes the place of the Laplace operator. The discrete Poisson equation is frequently used in numerical analysis as a stand-in for the continuous Poisson equation, although it is also studied in its own right as a topic in discrete mathematics. It makes a standard Matrix for testing in almost all cusp documentations.

After the successful testing using the Poisson Matrix now the actual lab generated matrix is brought to study the performance with respect to each iterative algorithm. The routine with the solver is changed after each execution and the call to different Krylov methods is executed. Tabulation is performed after checking if many times the algorithm solves the system in the same number of iterations only. Initially Conjugate gradient method was executed and 1964 iterations were required for solving. With Preconditioning the numbers improved to 1633 iterations. GMRES failed to converge as it is a method for non-Symmetric matrix. Conjugate Residual has performed well with 1555 iterations. Bi-Conjugate Stabilized Method has taken the least 1254 iterations. Bi-Conjugate Gradient Method has taken up to 1632 iterations.

| Sl No. | Name of the iterative solver method | Number of iterations |
|--------|---|----------------------|
| 1 | Conjugate Gradient Method | 1964 |
| 2 | Conjugate Gradient with Preconditioning | 1633 |
| 3 | GMRES | Failed to converge |
| 4 | Conjugate Residual | 1555 |
| 5 | Bi-Conjugate Stabilized Method | 1254 |
| 6 | Bi-Conjugate Gradient Method | 1632 |

Table 4.1 Number of Iterations in each method

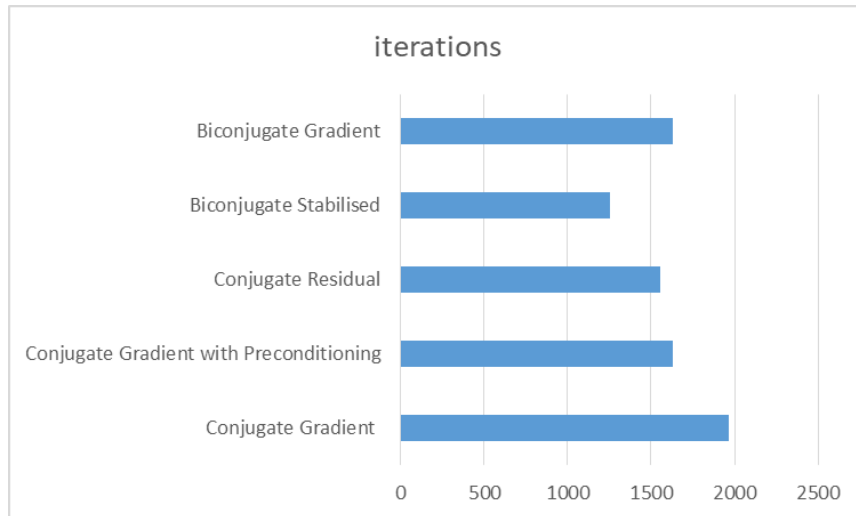


Fig 4.1 Graph plotting the different methods and number of iterations

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

The GPU-based iterative algorithm of image reconstruction shows that the algebraic iterative methods are capable to effectively Solve the Sparse Linear System at high computational cost leveraging the computational resources of the NVIDIA graphics processor (GPU). CUDA parallel programming model with CUSP, CUBLAS and CUSPARSE libraries allow overcoming the challenge to solve complex computational problems.

5.2 FUTURE SCOPE

To be able to reconstruct the image with the iterative method using the system matrix. To analyze the capacity of iterative algorithms in the parallel reconstruction of images from less number of projections.

APPENDIX A

SOURCE CODE

Program:

```
// main.cpp
// includes, system
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>

/* Using updated (v2) interfaces to cublas */
#include <cuda_runtime.h>
#include <cusparse.h>
#include <cublas_v2.h>

/*Used for operations on CSR file*/
#include "SpmmScripts.h"
#include "DeviceMatrix/SparseDeviceMatrixCSR.h"
#include "DeviceMatrix/SparseDeviceMatrixCSR0perations.h"

/*Declare the prototype cusp-based functions*/
int DoSomethingWithCusp(SparseDeviceMatrixCSR<double> A);

int main(int argc, char **argv)
{
    /*Loading the Matrix file in CSR format*/
    std::string file = "F:/Samuel/TestMatrices/mouse280.csr";
    SparseHostMatrixCSR<double> A=LoadSparseHostMatrixCSR<double>(file);
    SparseDeviceMatrixCSR<double> B = ToDevice(A);

    /*Host initializations*/
    cudaError_t cudaStat1, cudaStat2, cudaStat3, cudaStat4, cudaStat5,
    cudaStat6,cudaStat7;
    uint *csrRowIndexHostPtr = A.RowStarts().Pointer();
    uint *csrColIndexHostPtr = A.ColIndices().Pointer();
    double *csrValHostPtr = A.Values().Pointer();

    /*device initializations*/
    uint *csrRowIndexDevicePtr = B.RowStarts().Pointer();
    uint *csrColIndexDevicePtr = B.ColIndices().Pointer();
    double *csrValDevicePtr = B.Values().Pointer();

    float *x = NULL;
    float *rhs = NULL;
    int *d_RowIndex = NULL; /*d_col
    int *d_ColIndex = NULL; /*d_row
    float *d_Val = NULL; /*d_val
    float *d_x = NULL;
    float *d_r = NULL;
    float *d_p = NULL;
```

```

float *d_Ax = NULL;

/* Using values from Matrix A for Computation*/
int width = A.Width();//N
int height = A.Height();
int nz = (int)A.NonZeroCount();//nnz

/*Other Initializations*/
float alpha, beta, alpham1;
float a, b, na, r0, r1;
int k;
float dot;
const float tol = 1e-5f;
const int max_iter = 10000;
double rsum, diff, err = 0.0;

/* This will pick the best possible CUDA device*/
int device;
cudaGetDevice(&device);
cudaDeviceProp deviceProp;
cudaGetDeviceProperties(&deviceProp, device);

/* Statistics about the CUDA Device*/
printf("> GPU device has %d Multi-Processors, SM %d.%d compute
capabilities\n\n",deviceProp.multiProcessorCount, deviceProp.major, deviceProp.minor);

/* Get handle to the CUBLAS context */
cublasHandle_t cublasHandle = 0;
cublasStatus_t cublasStatus;
cublasStatus = cublasCreate(&cublasHandle);

/* Get handle to the CUSPARSE context */
cusparseHandle_t cusparseHandle = 0;
cusparseStatus_t cusparseStatus;
cusparseStatus = cusparseCreate(&cusparseHandle);

/* Creating Matrix Descriptor */
cusparseMatDescr_t descr = 0;
cusparseStatus = cusparseCreateMatDescr(&descr);

/* Define the properties of the matrix */
cusparseSetMatType(descr, CUSPARSE_MATRIX_TYPE_SYMMETRIC);
cusparseSetMatIndexBase(descr, CUSPARSE_INDEX_BASE_ZERO);

/* Allocating Memory in the device*/
cudaStat1 = cudaMalloc((void **)&d_ColIndex,nz*sizeof(int));
cudaStat2 = cudaMalloc((void **)&d_RowIndex, (width + 1) * sizeof(int));
cudaStat3 = cudaMalloc((void **)&d_Val, nz * sizeof(float));
cudaStat4 = cudaMalloc((void **)&d_x, width * sizeof(float));
cudaStat5 = cudaMalloc((void **)&d_r, width * sizeof(float));
cudaStat6 = cudaMalloc((void **)&d_p, width * sizeof(float));
cudaStat7 = cudaMalloc((void **)&d_Ax, width * sizeof(float));

/* Copying the elements in the Host to the memory allocated in the device*/

```

```

    cudaMemcpy(d_ColIndex, csrColIndexHostPtr, nz * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_RowIndex, csrRowIndexHostPtr, (width + 1) * sizeof(int),
cudaMemcpyHostToDevice);
    cudaMemcpy(d_Val, csrValHostPtr, nz * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(d_x, x, width * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(d_r, rhs, width * sizeof(double), cudaMemcpyHostToDevice);

    alpha = 1.0;
    alphas1 = -1.0;
    beta = 0.0;
    r0 = 0.;

    /* Preconditioning steps*/
    cusparseScsrmv(cusparseHandle, CUSPARSE_OPERATION_NON_TRANSPOSE, width, width, nz,
&alpha, descr, d_Val, d_RowIndex, d_ColIndex, d_x, &beta, d_Ax);
    cublasSaxpy(cublasHandle, width, &alphas1, d_Ax, 1, d_r, 1);

    cublasStatus = cublasSdot(cublasHandle, width, d_r, 1, d_r, 1, &r1);

    k = 1;

    while (r1 > tol*tol && k <= max_iter)
    {
        if (k > 1)
        {
            b = r1 / r0;
            cublasStatus = cublasSscal(cublasHandle, width, &b, d_p, 1);
            cublasStatus = cublasSaxpy(cublasHandle, width, &alpha, d_r, 1, d_p, 1);
        }
        else
        {
            cublasStatus = cublasScopy(cublasHandle, width, d_r, 1, d_p, 1);
        }

        cusparseScsrmv(cusparseHandle, CUSPARSE_OPERATION_NON_TRANSPOSE, width, width, nz,
&alpha, descr, d_Val, d_RowIndex, d_ColIndex, d_p, &beta, d_Ax);
        cublasStatus = cublasSdot(cublasHandle, width, d_p, 1, d_Ax, 1, &dot);
        a = r1 / dot;

        cublasStatus = cublasSaxpy(cublasHandle, width, &a, d_p, 1, d_x, 1);
        na = -a;
        cublasStatus = cublasSaxpy(cublasHandle, width, &na, d_Ax, 1, d_r, 1);

        r0 = r1;
        cublasStatus = cublasSdot(cublasHandle, width, d_r, 1, d_r, 1, &r1);
        cudaThreadSynchronize();
        printf("iteration = %3d, residual = %e\n", k, sqrt(r1));
        k++;
    }

    cudaMemcpy(x, d_x, width * sizeof(float), cudaMemcpyDeviceToHost);

```

```

for (int i = 0; i < height; i++)
{
    rsum = 0.0;
    int rowLength = A.RowLength(i);
    for (int j = csrRowIndexHostPtr[i]; j < rowLength; j++)
    {
        rsum += csrValHostPtr[j] * x[csrColIndexHostPtr[j]];
    }

    diff = fabs(rsum - rhs[i]);

    if (diff > err)
    {
        err = diff;
    }
}

int result = DoSomethingWithCusp(B);

/* Destroy contexts */
cuspDestroyMatDescr(descr);
cuspDestroy(cuspHandle);
cublasDestroy(cublasHandle);

/* Free device memory and Host memory */
free(csrRowIndexHostPtr);
free(csrColIndexHostPtr);
free(csrValHostPtr);
free(x);
free(rhs);
cudaFree(d_RowIndex);
cudaFree(d_ColIndex);
cudaFree(d_Val);
cudaFree(d_x);
cudaFree(d_r);
cudaFree(d_p);
cudaFree(d_Ax);
return 0;

```

```

// Cuspexample.cpp

#include <cuda_runtime.h>
#include <cusparse/choo_matrix.h>
#include <cusparse/csr_matrix.h>
#include <cusparse/monitor.h>
#include <cusparse/krylov/cg.h>
#include <cusparse/gallery/poisson.h>
#include <cusparse/io/matrix_market.h>
#include <cusparse/multiply.h>
#include "DeviceMatrix/SparseDeviceMatrixCSR.h"
#include "DeviceMatrix/SparseDeviceMatrixCSROperations.h"

void ToCusp(cusparse::csr_matrix<unsigned int, double, cusparse::device_memory>& result,
SparseDeviceMatrixCSR<double> B) {
    cusparse::csr_matrix<int, double, cusparse::host_memory>
A(B.Width(),B.Height(),B.NonZeroCount());

    HostVector<unsigned int> rowStarts = ToHost(B.RowStarts());
    for (int i = 0; i < B.Height(); i++)
        A.row_offsets[i] = rowStarts[i];

    HostVector<unsigned int> colindices = ToHost(B.ColIndices());
    for (int i = 0; i < B.Width(); i++)
        A.column_indices[i] = colindices[i];

    HostVector<double> values= ToHost(B.Values());
    for (int i = 0; i < B.Width(); i++)
        A.values[i] = values[i];

    result=cusparse::csr_matrix<unsigned int, double, cusparse::device_memory>(A);
}

int DoSomethingWithCusp(SparseDeviceMatrixCSR<double> B)
{
    cusparse::csr_matrix<unsigned int, double, cusparse::device_memory> A;
    ToCusp(A, B);
    int width = A.num_rows;
    cusparse::array1d<double, cusparse::device_memory> x(width, 0);
    cusparse::array1d<double, cusparse::device_memory> b(width, 1);
    cusparse::monitor<double> monitor(b, 10000, 1e-6, 0, true);
    cusparse::identity_operator<double, cusparse::device_memory> M(A.num_rows, A.num_rows);
    cusparse::krylov::cg(A, x, b, monitor,M);
    return 0;
}

// SparseHostMatrixCSR.h

class SparseHostMatrixCSR{

```

```

    int width;
    int height;
    HostVector<T> values;//length is number of nonzeros
    HostVector<uint> colIndices;//length is number of nonzeros
    HostVector<uint> rowStarts;//length is height+1(terminated CSR)
public:
    SparseHostMatrixCSR():width(0),height(0){}
    SparseHostMatrixCSR(int width, int height,    HostVector<T> values, HostVector<uint>
colIndices,HostVector<uint> rowStarts)

        :width(width),height(height),values(values),colIndices(colIndices),rowStarts(rowStarts
){
        Verify(values.Length()==colIndices.Length(),"v87r8498n8749");
        Verify(rowStarts.Length()==height+1,"4535c635hj5");
    }

    int Width()const{return width;}
    int Height()const{return height;}
    int DimX()const{return width;}
    int DimY()const{return height;}

    int64 NonZeroCount()const{return values.Length();}

    int RowLength(int r)const{
        unsigned int rowStart=rowStarts[r];
        int rowLength=rowStarts[r+1]-rowStart;
        return rowLength;
    }
void GetRowPointer(int r, T*& rowValues, unsigned int*& rowIndices, int& rowLength){
    unsigned int rowStart=rowStarts[r];
    rowLength=rowStarts[r+1]-rowStart;
    rowValues=values.Data()+rowStart;
    rowIndices=colIndices.Data()+rowStart;
}

void GetRow(int r, T*& rowValues, unsigned int*& rowIndices, int& rowLength){
    unsigned int rowStart=rowStarts[r];
    rowLength=rowStarts[r+1]-rowStart;
    rowValues=values.Data()+rowStart;
    rowIndices=colIndices.Data()+rowStart;
}

    CSparseVector<T> GetRow(int r){
        unsigned int rowStart=rowStarts[r];
        int nonZeros=rowStarts[r+1]-rowStart;
        return
CSparseVector<T>(values.Data()+rowStart,colIndices.Data()+rowStart,width,nonZeros);
    }

    SparseHostVector<T> Row(int r){
        unsigned int rowStart=rowStarts[r];
        int length=rowStarts[r+1]-rowStart;
        return SparseHostVector<T>(values.SubVector(rowStart,length),
colIndices.SubVector(rowStart,length),width);
    }

    HostVector<T> Values(){return values;}
    HostVector<uint> ColIndices(){return colIndices;}

```



```

        HostVector<uint> RowStarts(){return rowStarts;}
};

// SpmmScripts.h

template<typename T>
static SparseHostMatrixCSR<T> LoadSparseHostMatrixCSR(std::string fileName) {
    Verify(fileName.size()>4 && fileName.substr(fileName.size() - 4, 4) == ".csr", "File
must end with .csr");
    ClosingFile file(fileName, "rb");
    Verify(ExMI::Deserialize<int>(file) == 55673, "Control number missing");//Control
number
    Verify(ExMI::Deserialize<int>(file) == sizeof(T), "Wrong data type");
    int width = ExMI::Deserialize<int>(file);
    int height = ExMI::Deserialize<int>(file);
    int64 nnz = ExMI::Deserialize<int64>(file);
    HostVector<T> values(nnz);
    ExMI::Deserialize(file, values.Pointer(), values.Length());
    HostVector<uint> colIndices(nnz);
    ExMI::Deserialize(file, colIndices.Pointer(), colIndices.Length());
    HostVector<uint> rowStarts(height + 1);
    ExMI::Deserialize(file, rowStarts.Pointer(), rowStarts.Length());
    return SparseHostMatrixCSR<T>(width, height, values, colIndices, rowStarts);
}

template<typename T>
static void Serialize(ClosingFile& file, const T& t) {
    fwrite(&t, sizeof(t), 1, file.GetFile());
}

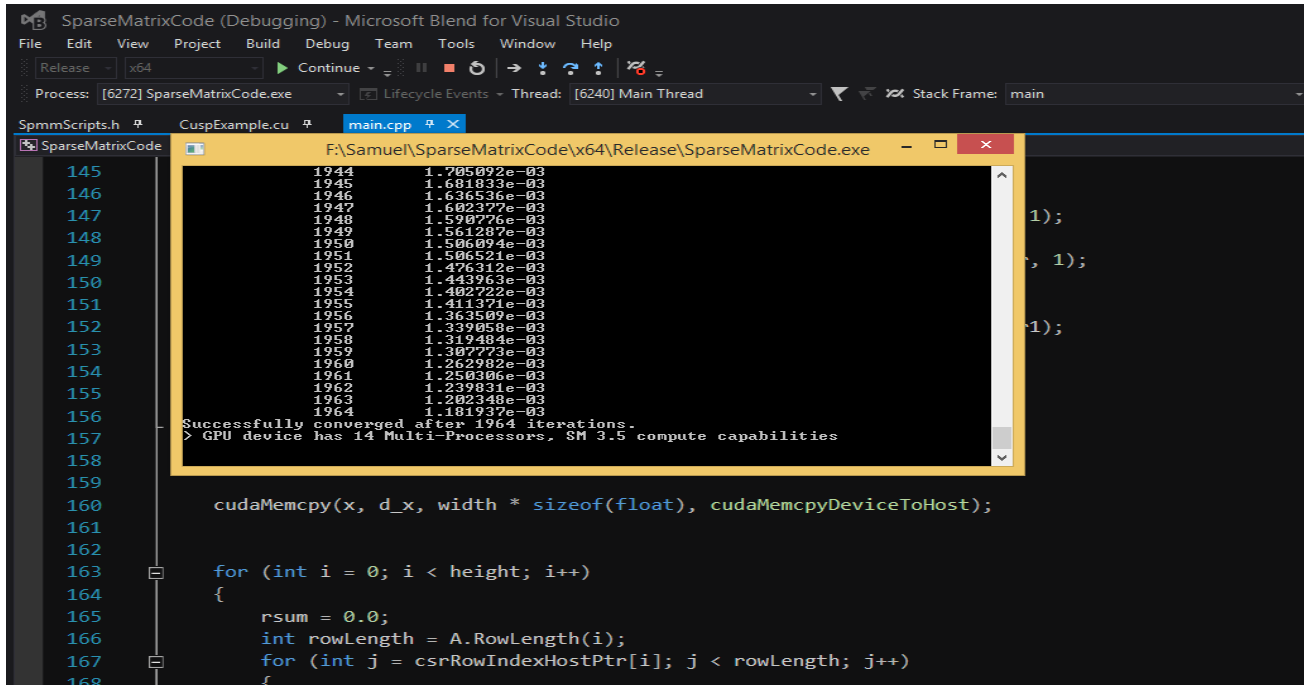
template<typename T>
static void Deserialize(ClosingFile& file, T* p, int64 n) {
    fread(p, sizeof(T), n, file.GetFile());
}

```

APPENDIX B

OUTPUT SCREENSHOTS

Conjugate Gradient Method:



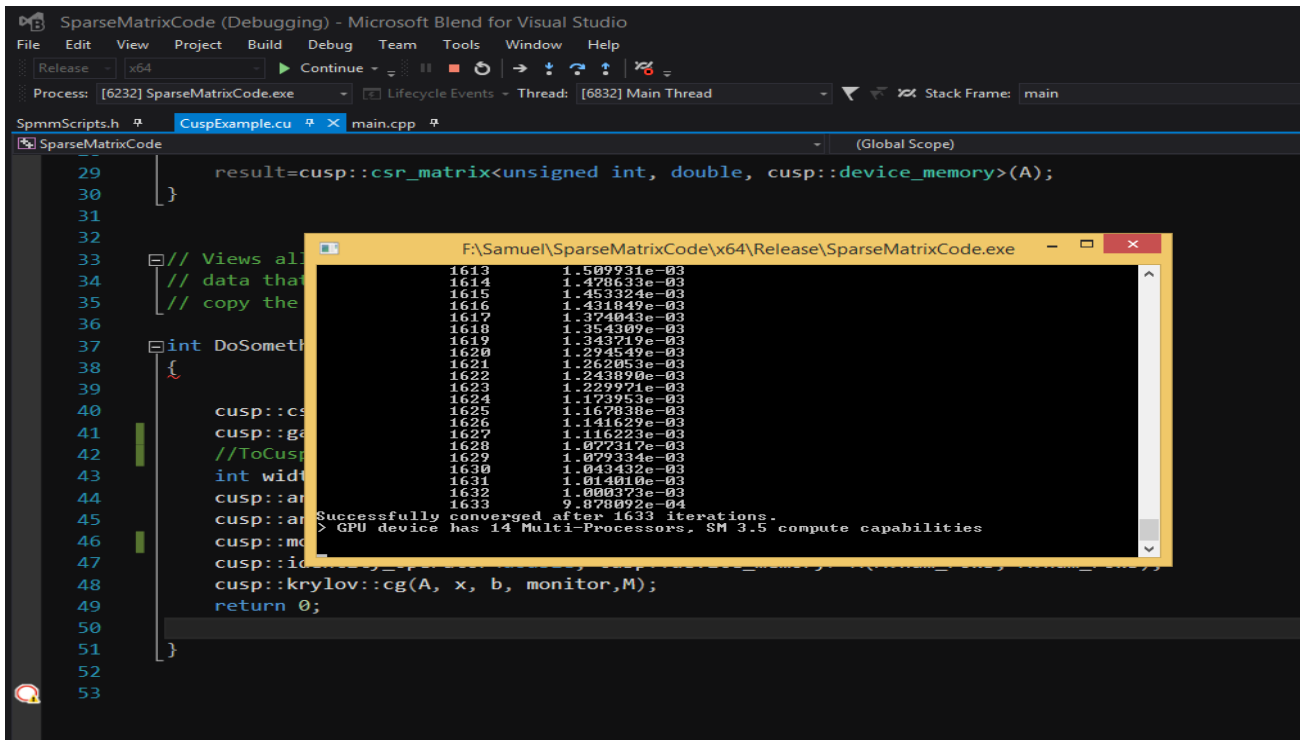
```
1944 1.705092e-03
1945 1.681833e-03
1946 1.636536e-03
1947 1.602377e-03
1948 1.590776e-03
1949 1.561287e-03
1950 1.506094e-03
1951 1.506521e-03
1952 1.476312e-03
1953 1.443963e-03
1954 1.402722e-03
1955 1.411321e-03
1956 1.363509e-03
1957 1.339058e-03
1958 1.319484e-03
1959 1.307723e-03
1960 1.262982e-03
1961 1.250306e-03
1962 1.239831e-03
1963 1.202348e-03
1964 1.181937e-03

Successfully converged after 1964 iterations.
> GPU device has 14 Multi-Processors, SM 3.5 compute capabilities

cudaMemcpy(x, d_x, width * sizeof(float), cudaMemcpyDeviceToHost);

for (int i = 0; i < height; i++)
{
    rsum = 0.0;
    int rowLength = A.RowLength(i);
    for (int j = csrRowIndexHostPtr[i]; j < rowLength; j++)
    {
```

Conjugate Gradient preconditioning Method:



```
29 result=cusp::csr_matrix<unsigned int, double, cusp::device_memory>(A);
30 }
31
32 // Views all
33 // data that
34 // copy the
35
36 int DoSomething
37 {
38
39     cusp::cs
40     cusp::ga
41     //ToCusp
42     int width
43     cusp::ar
44     cusp::ar
45     cusp::m
46     cusp::id
47     cusp::id
48     cusp::krylov::cg(A, x, b, monitor,M);
49     return 0;
50 }
51
52
53
```

```
1613 1.509931e-03
1614 1.478633e-03
1615 1.453324e-03
1616 1.431849e-03
1617 1.374043e-03
1618 1.354309e-03
1619 1.343719e-03
1620 1.294549e-03
1621 1.262053e-03
1622 1.243890e-03
1623 1.229271e-03
1624 1.173953e-03
1625 1.167838e-03
1626 1.141629e-03
1627 1.116223e-03
1628 1.077317e-03
1629 1.079334e-03
1630 1.043432e-03
1631 1.014010e-03
1632 1.000373e-03
1633 0.878092e-04

Successfully converged after 1633 iterations.
> GPU device has 14 Multi-Processors, SM 3.5 compute capabilities
```

GRMES Method:

The screenshot shows the Visual Studio IDE with the 'SparseMatrixCode' project open. The 'main.cpp' file is displayed, showing a loop that iterates over rows and columns, calculating a residual sum (rsum) and a difference (diff). The code is in C++ and uses CUDA for GPU acceleration. The output window shows the results of the GRMES method, displaying a list of values and a message indicating that the method failed to converge after 2000 iterations.

```
160 cudaMemcpy(x, d_x, width * sizeof(float), cudaMemcpyDeviceToHost);
161
162
163 for (int i = 0; i < height; i++)
164 {
165     rsum = 0.0;
166     int rowLen = ...;
167     for (int j = 0; j < rowLen; j++)
168     {
169         rsum += ...;
170     }
171     diff = fabs(rsum);
172
173     if (diff > ... || i % 100 == 0)
174     {
175         err = ...;
176     }
177 }
178
179
180 int result = DoSomethingWithCusp(B);
```

Output window content:

```
1982 5.002460e+02
1983 5.001191e+02
1984 4.999921e+02
1985 4.998670e+02
1986 4.997399e+02
1987 4.996137e+02
1988 4.994845e+02
1989 4.993597e+02
1990 4.992323e+02
1991 4.991074e+02
1992 4.989773e+02
1993 4.988498e+02
1994 4.987235e+02
1995 4.985984e+02
1996 4.984707e+02
1997 4.983340e+02
1998 4.982110e+02
1999 4.981015e+02
2000 4.979908e+02
Failed to converge after 2000 iterations.
Failed to converge after 2000 iterations.
> GPU device has 14 Multi-Processors, SM 3.5 compute capabilities
```

Conjugate Residual:

The screenshot shows the Visual Studio IDE with the 'SparseMatrixCode' project open. The 'main.cpp' file is displayed, showing a loop that iterates over rows and columns, calculating a residual sum (rsum) and a difference (diff). The code is in C++ and uses CUDA for GPU acceleration. The output window shows the results of the Conjugate Residual method, displaying a list of values and a message indicating that the method successfully converged after 1555 iterations.

```
160 cudaMemcpy(x, d_x, width * sizeof(float), cudaMemcpyDeviceToHost);
161
162
163 for (int i = 0; i < height; i++)
164 {
165     rsum = 0.0;
166     int rowLen = ...;
167     for (int j = 0; j < rowLen; j++)
168     {
169         rsum += ...;
170     }
171     diff = fabs(rsum);
172
173     if (diff > ... || i % 100 == 0)
174     {
175         err = ...;
176     }
177 }
178
179
180 int result = DoSomethingWithCusp(B);
```

Output window content:

```
1535 1.437922e-03
1536 1.410140e-03
1537 1.382540e-03
1538 1.354313e-03
1539 1.328178e-03
1540 1.302796e-03
1541 1.278024e-03
1542 1.252454e-03
1543 1.228543e-03
1544 1.205624e-03
1545 1.183152e-03
1546 1.160264e-03
1547 1.138292e-03
1548 1.117237e-03
1549 1.097211e-03
1550 1.076735e-03
1551 1.056576e-03
1552 1.038094e-03
1553 1.019268e-03
1554 1.000717e-03
1555 9.822953e-04
Successfully converged after 1555 iterations.
> GPU device has 14 Multi-Processors, SM 3.5 compute capabilities
```

Bi-conjugate Stabilized Method:

The screenshot shows the Visual Studio IDE with the 'SparseMatrixCode' project open. The 'main.cpp' file is selected, and the 'Release' configuration is active. The 'Process' is '[2304] SparseMatrixCode.exe'. The 'Thread' is '[5716] Main Thread'. The 'Stack Frame' is 'main'. The code is running, and a console window is open, displaying the following output:

```
1244 8.704680e-03
1245 2.836612e-03
1245 1.931393e-03
1246 1.931130e-03
1246 1.627247e-03
1247 1.626961e-03
1247 1.199271e-03
1248 1.199141e-03
1248 1.145914e-03
1249 1.145804e-03
1249 1.109172e-03
1250 1.109086e-03
1250 1.110923e-03
1251 1.110370e-03
1251 1.106913e-03
1252 1.105296e-03
1252 1.112385e-03
1253 1.014019e-03
1253 8.053920e-03
1254 1.038436e-03
1254 9.756394e-04
Successfully converged after 1254 iterations.
> GPU device has 14 Multi-Processors, SM 3.5 compute capabilities
```

The code in the background shows the following lines:

```
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160 cudaMemcpy(x, d_x, width * sizeof(float), cudaMemcpyDeviceToHost);
161
162
163 for (int i = 0; i < height; i++)
164
```

Bi-conjugate Gradient Method:

The screenshot shows the Visual Studio IDE with the 'SparseMatrixCode' project open. The 'main.cpp' file is selected, and the 'Release' configuration is active. The 'Process' is '[5424] SparseMatrixCode.exe'. The 'Thread' is '[3796] Main Thread'. The 'Stack Frame' is 'main'. The code is running, and a console window is open, displaying the following output:

```
1612 1.509931e-03
1613 1.478633e-03
1614 1.453324e-03
1615 1.431849e-03
1616 1.374043e-03
1617 1.354309e-03
1618 1.343719e-03
1619 1.294549e-03
1620 1.262053e-03
1621 1.243890e-03
1622 1.229971e-03
1623 1.173953e-03
1624 1.167838e-03
1625 1.141622e-03
1626 1.116223e-03
1627 1.077317e-03
1628 1.079334e-03
1629 1.043432e-03
1630 1.014019e-03
1631 1.000373e-03
1632 9.878092e-04
Successfully converged after 1632 iterations.
> GPU device has 14 Multi-Processors, SM 3.5 compute capabilities
```

The code in the background shows the following lines:

```
145 a = r1 / dot;
146
147 cublasStatus = cublasSaxpy(cublasHandle, width, &a, d_p, 1, d_x, 1);
148 na = a;
149 cublasStatus = cublasSaxpy(cublasHandle, width, &a, d_p, 1, d_x, 1);
150
151 r0 = r1;
152 cublasStatus = cublasSaxpy(cublasHandle, width, &a, d_p, 1, d_x, 1);
153 cudaMemset(d_r, 0, width * sizeof(float));
154 printf("Iteration %d\n", k);
155 k++;
156 }
157
158
159
160 cudaMemset(d_r, 0, width * sizeof(float));
161
162
163 for (int i = 0; i < height; i++)
164 {
165     rsum = 0.0;
166     int rowLength = A.RowLength(i);
167     for (int j = csrRowIndexHostPtr[i]; j < rowLength; j++)
168     {
169         rsum += csrValHostPtr[j] * x[csrColIndexHostPtr[j]];
170     }
171 }
```

REFERENCES

- [1] Gremse, F., Doleschel, D., Zafarnia, S., Babler, A., Jahnen-Dechent, W., Lammers, T., Lederle, W., Kiessling, F. Hybrid μ CT-FMT imaging and image analysis. *J. Vis. Exp.* (100), e52770, doi:10.3791/52770 (2015).
- [2] Felix Gremse, Benjamin Theek, Sijumon Kunjachan, Wiltrud Lederle, Alessa Pardo, Stefan Barth, Twan Lammers, Uwe Naumann, Fabian Kiessling Absorption Reconstruction Improves Biodistribution Assessment of Fluorescent Nanoprobes Using Hybrid Fluorescence-mediated Tomography.
- [3] Felix Gremse, Andreas Höfter, Lars Ole Schwen, Fabian Kiessling, and Uwe Naumann GPU-Accelerated Sparse Matrix-Matrix Multiplication by Iterative Row Merging.
- [4] Liubov Flores , Vicent Vidal, Gumersindo Verdú System Matrix Analysis for Computed Tomography Imaging
- [5] Somesh Srivastava, Accelerated Statistical image reconstruction algorithms and simplified cost functions for X-ray Computed Tomography
- [6] Liubov A. Flores*, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú CT Image Reconstruction Based on GPUs
- [7] Wikipedia- Medical imaging CT