To the Graduate Council:

I'm submitting herewith a thesis written by Dilip Reddy Patlolla entitled "A GPU-based Implementation for Improved Online Rebinning Performance in Clinical 3-D PET". I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical and Computer Engineering.

Dr. Gregory D. Peterson, Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Hairong Qi

Dr. Itamar Arel

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

A GPU-based Implementation for Improved Online Rebinning Performance in Clinical 3-D PET

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Dilip Reddy Patlolla

December 2009

Copyright © 2009 by Dilip Reddy Patlolla

All rights reserved.

ACKNOWLEDGEMENTS

First of all, I would like to thank all my professors at UT for instructing me in different fields of Electronics at the graduate level. The knowledge that they have imparted to me is immense and invaluable. A special thanks to my supervisor Dr. Gregory D. Peterson for guiding me to the successful completion of my Masters degree. I would also like to express my sincere gratitude to Dr. Itamar Arel and Dr. Hairong Qi for reviewing my work and serving on my committee. I would also like to thank Dr. Donald W. Bouldin for the entire series of Digital Design courses, which gave me a complete overview of the fundamentals underlying Digital VLSI design.

I would like to take this opportunity to thank the entire Electronics Research & Development team of Siemens Medical Solutions, Knoxville for giving me an opportunity to work for them and supporting me to complete my Master's thesis. Special regards to my manager Mr. Chad Seaver who recruited me and guided me to successfully complete the implementation.

Special thanks to Mr. Eric Breeding, who proposed this idea and guided me through the technicalities of the algorithm.I would like to thank him for all the great discussions we had on varied topics which gave me the much needed break from work and also for being ever ready to help me in other aspects as well. I would like to thank Mr.Bill Jones for helping me in understanding the principles of PET and also for providing the Time of Flight algorithm.

A true friend is someone who thinks that you are a good egg even though he knows that you are slightly cracked. This is really true in the case of Siva, Praveen, Lavanya and Varun, who are my best friends and great gifts and made my life a wonderful and fun filled experience. I would like to thank Murali, Kanth, Narsi, Sarat and Phani who have made my stay in Knoxville a memorable one. I would also like to thank Mr.Selva Kumar who is my friend , guide and philosopher.

I would like to thank my grandparents Sri. Ranga Reddy, Smt. Lakshmi and Late. Sri Ananth Reddy and Smt. Shashi Rekha for their love and support. I am most indebted to my parents, Sri. Bhoopal Reddy and Smt. Uma Reddy and my sister Ms.Anoosha Reddy. I should have been truly blessed to have such amazing parents and take this opportunity to express my thanks for the unconditional love and unrelenting support, and encouragement they have given me. I also wish to express my thanks to all my uncles, aunts and cousins for the love and affection they have always showered onto me.

Finally, I wish to thank the Almighty and my ancestors for their love and blessings, without which none of this would have been possible.

ABSTRACT

Online rebinning is an important and well-established technique for reducing the time required to process Positron Emission Tomography data. However, the need for efficient data processing in a clinical setting is growing rapidly and is beginning to exceed the capability of traditional online processing methods. High-count rate applications such as Rubidium 3-D PET studies can easily saturate current online rebinning technology. Real-time processing at these high-count rates is essential to avoid significant data loss. In addition, the emergence of time-of-flight (TOF) scanners is producing very large data sets for processing. TOF applications require efficient online Rebinning methods so as to maintain high patient throughput. Currently, new hardware architectures such as Graphics Processing Units (GPUs) are available to speedup data parallel and number crunching algorithms. In comparison to the usual parallel systems, such as multiprocessor or clustered machines, GPU hardware can be much faster and above all, it is significantly cheaper. The GPUs have been primarily delivered for graphics for video games but are now being used for High Performance computing across many domains. The goal of this thesis is to investigate the suitability of the GPU for PET rebinning algorithms.

TABLE OF CONTENTS

CHAPTER PAGE

1 INTRODUCTION 1

1.1 MOTIVATION 1

1.2 ORGANIZATION 2

2 POSITRON EMISSION TOMOGRAPHY 3

2.1 INTRODUCTION 3

2.2 HISTORY OF PET 5

2.3 APPLICATION AREAS 6

2.4 PET FUNCTIONALITY 8

3 APPROACH 11

3.1 GRAPHICS PROCESSING UNITS 11

3.1.1 DIFFERENCE BETWEEN CPU AND GPU 12

3.1.2 GPU ARCHITECTURE 15

3.2 COMPUTE UNIFIED DEVICE ARCHITECTURE 21

3.2.1 INTRODUCTION 21

3.2.2 CUDA PROGRAM STRUCTURE 21

3.2.3 ADVANTAGES 22

3.2.4 HARDWARE MODEL 23

3.2.5 PROGRAMMING MODEL 25

3.2.6 CUDA MEMORY MODEL 26

4 RELATED WORK 30
4.1 PRESENT SYSTEM 30
4.2 PDR HARDWARE DESCRIPTION 31
4.2.1 PROCESSING 32
4.2.2 DATAFLOW 33
5 PET PHYSICS AND REBINNING 36
5.1 COINCIDENCE DETECTION 36
5.2 TIME OF FLIGHT (TOF) 38
5.3 TYPES OF COINCIDENT EVENTS 39
5.3.1 SCATTERED COINCIDENCE 39
5.3.2 TRUE COINCIDENCE 40
5.3.3 RANDOM COINCIDENCE 41
5.4 DATA ACQUISITION FOR PET – 2 D MODE AND 3 D MODE 42
5.5 SINOGRAM GENERATION AND REBINNING 44
5.5.1 COORDINATES 44
5.6 IMAGE RECONSTRUCTION 47
5.7 3D PET DATA 49
5.8 REBINNING 51
6 IMPLEMENTATION 53
6.1 INTRODUCTION 53
6.2 REBINNING ALGORITHM 53
6.2.1 PARTITIONING OF THE PROBLEM 55

6.3 SYSTEM 57
6.4 64 BIT DATA FORMAT 58
6.5 WORKING MODEL 59
6.5.1 PRE COMPUTATION AND CONSTANT MEMORY 60
6.6 OPTIMIZATIONS 61
6.6.1 CONSTANT MEMORY AND PRECOMPUTATION 62
6.6.2 MEMORY ACCESS PATTERNS 62
6.6.3 MEMORY COALESCING 63
6.6.4 ARITHMETIC INSTRUCTIONS 65
6.7 ASYNCHRONOUS APPLICATION PROGRAMMING INTERFACE 66
6.8 SYSTEM SPECIFICATIONS 68
7 RESULTS 69
7.1 COMPARISON OF RESULTS 69
7.2 CUDA VISUAL PROFILER 75
7.3 LIMITATIONS 79
8 CONCLUSION AND FUTURE WORK 80
8.1 CONCLUSION 80
8.2 FUTURE WORK 81
BIBLIOGRAPHY 82
VITA 88

LIST OF FIGURES

Figure 2.1 PET Scanner 4

Figure 2.2 Annihilation and emission (Adapted from [1] with permission) 9

Figure 2.3 Event Detection (Adapted from [1] with permission) 10

Figure 3.1 CPU and GPU Block Representation (reproduced from [20]) 12

Figure 3.2 Performance between GPUs and CPUs for Single precision Floating Point operations (Based on slide 7 from GPU Physics SIGGRAPH 2007) 15

Figure 3.3 Nvidia Tesla Architecture (reproduced from [19]) 16

Figure 3.4 SM Multithreaded Processor (reproduced from [19]) 18

Figure 3.5 Decomposition structure of a parallel program for GPU 20

Figure 3.6 CUDA Hardware Model (adapted from [20]) 24

Figure 3.7 CUDA Programming Model 25

Figure 3.8 CUDA Memory Model (reproduced from [20]) 27

Figure 4.1 Present system with PET Gantry and Acquisition system with PDR card (Reprinted from [25] with permission) 31

Figure 4.2 Snapshot of a PDR (reprinted from [23] with permission) 32

Figure 4.3 PDR Block Diagram (reproduced from [25]) 33

Figure 4.4 Stage Digital PDR pipeline (reprinted from [25] with permission) 34

Figure 5.1 Coincidence Detection 37

Figure 5.2 Scattered Coincident Event (adapted from [1]) 39

Figure 5.3 True Coincident Event (adapted from [1]) 40

Figure 5.4 Random Coincident Event (adapted from [1]) 41

Figure 5.5 2D – coincidences between detectors in the same ring or neighboring rings. .42

Figure 5.6 3D – coincidences between any pair of rings permitted 43

Figure 5.7 LOR and sinogram coordinates.....45

Figure 5.8 LORs plotted in sinogram format.....46

Figure 5.9 projections generated from a single point source.....47

Figure 5.10 Back-projections of a point source (reproduced from [1]).....48

Figure 5.11 Δr and z coordinates of an LOR in a multi ring scanner.....50

Figure 5.12 Transaxial view (left) and longitudinal section (right) of scanner showing the projection of an LOR onto the transaxial plane located at point z 51

Figure 5.13 Principal of rebinning algorithm 52

Figure 6.1 3-stage PDR Digital Pipeline as applied to Coincident event system.....54

Figure 6.2 7-stage digital pipeline PDR Digital Pipeline as applied to the TOF PET System. (reproduced from [25] with permission).....55

Figure 6.3 Partitioning the rebinning problem 56

Figure 6.4 New system with PDR card along with GPU incorporated on the Data Acquisition System.....57

Figure 6.5 Working Model of the GPU based rebinning system 59

Figure 6.6 Memory Coalescing 64

Figure 6.7 Asynchronous implementation of kernel in CUDA.....67

Figure 7.1 Rebinning time performance.....70

Figure 7.2 Throughput performance comparision.....71

Figure 7.3 Comparision of Execution Time 72

Figure 7.4 Comparision of Transfer Time.....72

Figure 7.5 TOF Rebinning Performance 74

Figure 7.6 Height Plot using CUDA visual profiler 75

Figure 7.7 Percentage plot using CUDA visual profiler.....76

Figure 7.8 Kernel runtime stats from CUDA visual profiler.....77

Figure 7.9 Instruction Output from CUDA visual profiler.....78

LIST OF TABLES

Table 2.1 Radiotracers and their clinical applications.....7

Table 6.1 64 Bit raw data packet 58

Table 6.2 GPU Specifications.....68

Table 7.1 Performance of Coincident event rebinning.....73

Table 7.2 TOF rebinning performance values.....74

1 INTRODUCTION

Online rebinning algorithms are very high count-rate applications that require real time data processing to avoid data loss during the process. The implementation of PET rebinning algorithms on a parallel computing platform provides a possible solution to meet the demands of efficient data processing in a clinical setting. Using a Graphics Processing Unit (GPU) to implement the rebinning algorithms improves the computational performance and allows for a high input data rate.

In this thesis the focus will be towards the analysis of GPU architecture, the analysis of Rebinning algorithm and the implementation. The limited focus in going into the details of the rebinning algorithms is the restricted permissions from the original developers of the algorithm at CTI Siemens.

1.1 MOTIVATION

The motivation of this work comes from the idea of implementing the rebinning algorithm on a GPU instead of the existing dedicated PDR rebinning hardware. The rebinning algorithms are highly parallelizable and thus are highly suited for the GPU architecture.

1.2 ORGANIZATION

Chapter 2 presents an introduction to Positron Emission Tomography (PET) and also the working principles. The functionality, history and working of PET have been discussed. Chapter 3 presents the details about the related work. A brief description about the existing dedicated rebinning hardware is presented. Chapter 4 presents the design approach for the implementation. GPU and Compute Unified Device Architecture are elaborately discussed. Chapter 5 discusses about the PET physics and rebinning. An outline of the rebinning algorithm is discussed.

2 POSITRON EMISSION TOMOGRAPHY

2.1 INTRODUCTION

Positron emission tomography (PET) is a major diagnostic imaging modality that helps the doctors view the patient's internal organs. It is a radiotracer imaging technique in which small amounts of radioisotopes are injected into a patient to visualize and track biochemical and physiological processes *in vivo*. It produces two-dimensional or three-dimensional images that reflect tracer concentration within the patient. This concentration is proportional to volumetric metabolic activity and often correlates with physical abnormalities. The main reason for the importance of PET in medical research is the existence of positron emitting isotopes such as carbon, nitrogen, and oxygen, which may be processed to create a range of tracer compounds which are similar to naturally occurring substances in the body. For example, one of the most widely used isotopes is the positron emitting isotope of fluorine, ^{18}F [1]. ^{18}F can be substituted, through a chemical synthesis process, for hydrogen in complex compounds such as glucose forming Fluro-deoxy-glucose (FDG). When FDG is injected into the patient, the body will attempt to use it in the same fashion as it would normal glucose. A modern PET scanner is shown in Figure 2.1.

Figure 2.1 PET Scanner

2.2 HISTORY OF PET

Researchers interest in positron emitting isotopes arose from the fact that three of the most basic elements in the human body (carbon, nitrogen, and oxygen) occur as positron emitters. Tatsuo Ido *et al* at the Brookhaven National Laboratory was the first to describe the synthesis of ^{18}F-FDG, the most commonly used PET scanning isotope carrier [14]. Abass Alavi *et al* first administered the compound to two normal human volunteers in August 1976 at the University of Pennsylvania [15]. FDG was later used in dedicated positron tomographic scanners, to yield the modern procedure [16]. The development of radiopharmaceuticals like FDG made it easier to study living beings, and set the groundwork for more in-depth research into using PET to diagnose and evaluate the effect of treatment on human disease [16].

The first PET camera development is credited to Rankowitz in 1962 [12]. The scanner was developed at Brookhaven National Laboratory [16]. The design consisted of a single ring of 32 sodium iodide crystals that allowed each crystal to be in coincidence with number of other crystals. The poor stopping ability of the sodium iodide detectors led to the loss of sensitivity gains due to the scanners electronic collimation.

David Kuhl and Roy Edwards made another attempt in 1963 [13]. Michel Ter-Pogossian, Michael E. Phelps and others further developed tomographic imaging techniques at the Washington University School of Medicine [1].

Through the 1970s, many single PET camera designs were attempted. One of the first successful PET cameras was developed in 1975[16]. It consisted of 48 detectors in six banks. The detectors were made of a long cylinder of sodium iodide with one photomultiplier tube mounted at each end of the cylinder. These cylindrical detectors were arranged with their length axis parallel with patient opening axis [11]. Axial positioning was performed by the ratio of light seen by the two photo multiplier tubes. This ratio space was equally divided into four axial planes. By accepting adjacent ring coincidences, three cross planes of data were also created, producing a total of seven planes of data.

2.3 APPLICATION AREAS

The largest area of clinical use of PET is in oncology. The most widely used tracer in oncology is ^{18}F-Fluoro-deoxy-glucose (^{18}F-FDG). ^{18}F-FDG is relatively easy to synthesize with a high radiochemical yield [3] .It is used to detect and determine whether a cancer has spread in a body and assess the effectiveness of the treatment plan.

PET also has applications in cardiology. ^{13}N-NH_3 is used as a tracer for myocardial perfusion. When ^{13}N-NH_3 and ^{18}F-FDG scans of the same patient are interpreted together, PET can be used to distinguish between viable and non-viable tissue in poorly perfused areas of the heart [4]. Such information is extremely valuable in determining the effects of a heart attack, or myocardial infarction, on areas of the heart and also to identify areas of the heart muscle that would benefit from a procedure such as angioplasty or coronary artery bypass surgery (in combination with a myocardial perfusion scan) [7].

In neurology, PET has been used in evaluating brain abnormalities, such as tumors, memory disorders, seizures, and other central nervous system disorders [8]. It offers the possibility of quantitative measurements of biochemical and physiological processes *in vivo*. This is important in both research and in clinical applications. For example, it has been shown that semi-quantitative measurements of FDG uptake in tumors can be useful in the grading of disease [5]. Some examples of these radiotracers are shown in Table 2.1, together with some typical clinical and research applications.

Table 2.1 Radiotracers and their clinical applications

Isotope	**Tracer compound**	**Physiological function**	**Typical application**	**Reference**
^{11}C	methionine	protein synthesis	oncology	Hellman *et al* [6]
^{13}N	ammonia	blood perfusion	myocardial perfusion	Kuhle *et al* [7]
^{15}O	carbon dioxide	blood perfusion	brain activation studies	Kanno *et al* [8]
^{18}F	Fluoro-deoxy-glucose	glucose metabolism	oncology, neurology, cardiology	Brock *et al* [9]
^{8}F	Fluoride ion	bone metabolism	oncology	Hawkins *et al* [10]

2.4 PET FUNCTIONALITY

A very small amount of a radio labeled compound is injected into the subject. The injected or inhaled compound accumulates in the tissue to be studied. This compound, which closely resembles a natural substance used by the body, is meticulously selected to have a specified short life and is generally in the form of glucose. The radioactive substance used during a PET scan depends on the organ under investigation as different tissues in the body take up different radionuclides.

The subject is placed within the Field of View (FOV) of a number of detectors capable of registering incident gamma rays. The injected or inhaled compound accumulates in the tissue to be studied. As the radioactive atoms in the compound decay, they release smaller particles called positrons that are positively charged. When the positrons reach thermal energies, they start to interact with electrons by annihilation and their rest masses are converted into a pair of annihilation photons. Each annihilation produces two photons (light particles) having identical energies (511 KeV) and are emitted simultaneously in ~180 degree opposing directions. These photons may be detected by the detectors, which are linked so that two detection events unambiguously occurring within a certain time window may be called coincident and thus be determined to have come from the same annihilation. Figure 2.2 illustrates the process of annihilation and emission.

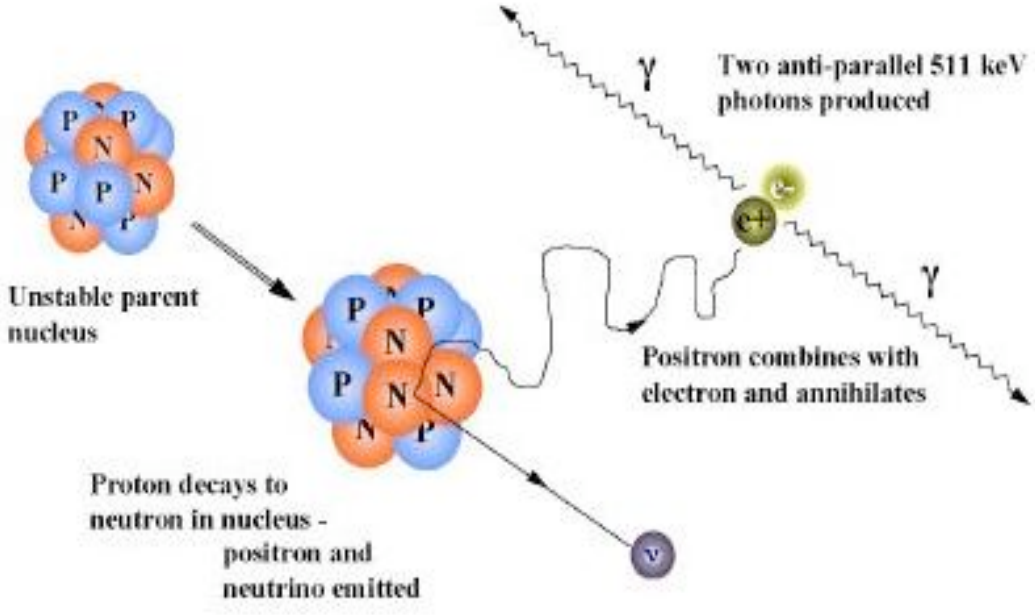


Figure 2.2 Annihilation and emission (Adapted from [1] with permission)

The near simultaneous detection of the two photons enables PET to localize their origin along a line between the two detectors known as Line of Response (LOR). In the PET camera, each detector generates a timed pulse when it registers an incident photon. These pulses are then combined in coincidence circuitry, and if the pulses fall within a short time-window, they are deemed to be coincident. These coincident events can be stored in arrays corresponding to projections through the patient and reconstructed using tomographic techniques. Figure 2.3 shows an overview this process.

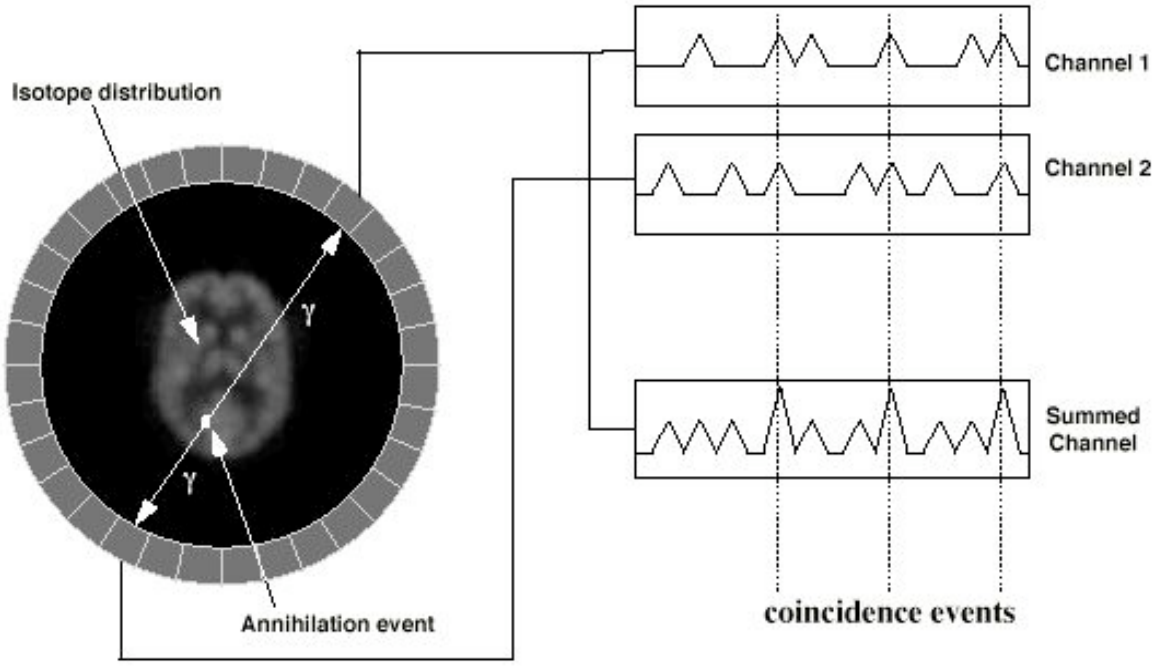


Figure 2.3 Event Detection (Adapted from [1] with permission)

The number of positrons emitted is proportional to the chemical activeness of the organ or tissue as it indicates the amount of radioactive substance the organ has taken up. Hence, areas that take up more compounds are brighter on a PET scan and areas that don't intake much compound are damaged and are therefore not as bright on a PET scan. The information from PET is further processed and converted into images.

The resulting pictures do not show as much detail as Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) because the pictures show only the location of the tracer. The PET picture may be matched with those from a CT scan to get more detailed information about where the tracer is located.

3 APPROACH

3.1 GRAPHICS PROCESSING UNITS

Traditionally, many applications are written as sequential programs, whose execution can be understood by sequentially stepping through the code. A single processor core is used to run the sequential program. These single processor cores are not expected to become much faster than those of today as at a frequency of about 4 GHz the heat losses are too big, hence it is not practical to increase the clock speeds of single core.

Without performance improvement, application developers will no longer be able to introduce new features and capabilities into their software, reducing the growth opportunities of the entire computer industry. Rather, the applications software that will continue to enjoy performance improvement with each new generation of microprocessors will be a parallel program, in which multiple threads of execution cooperate to achieve faster functionality.

A Graphic Processing Unit (GPU) is a dedicated graphics-rendering device. GPUs are very efficient at manipulating and displaying computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for a range of complex algorithms.

3.1.1 DIFFERENCE BETWEEN CPU AND GPU

Physical matters and high power consumption now limit CPU frequency growth. Increasing the number of cores often raises their performance. GPUs are based on a multiprocessor with many cores and hundreds of Arithmetic logic units (ALU), several thousand registers, and some shared memory. Besides, a graphics card contains fast global memory, which can be accessed by all multiprocessors, local memory in each multiprocessor, and special memory for constants. The multiprocessor cores in a GPU are Single Instruction Multiple Data (SIMD) cores. These cores execute the same instructions simultaneously. This programming style is essential for graphics algorithms and many scientific tasks. Figure 3.1 illustrates differences in the design of a CPU and a GPU where GPU devotes more transistors for data processing than data caching and flow control [19].

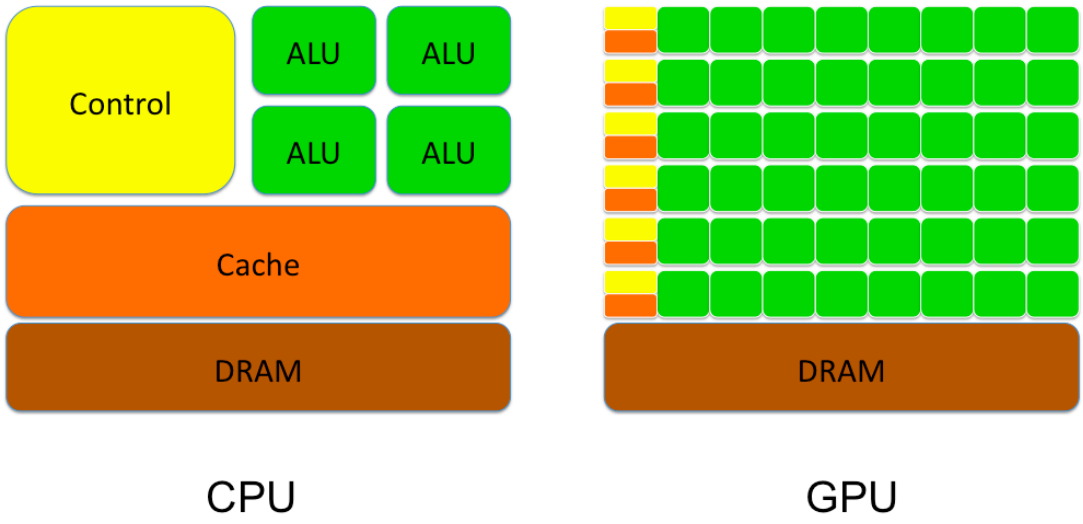


Figure 3.1 CPU and GPU Block Representation (reproduced from [20])

CPU cores are basically designed to execute a small number thread of sequential instructions with maximum speed, though some parallel applications can be programmed through MPI [31]. GPUs are designed for fast execution of many parallel instruction threads. The GPUs use a lot of execution units that can be easily loaded, unlike sequential instruction threads for CPU [17].

Memory operations are different in GPUs and CPUs. For example, not all CPUs have built-in memory controllers, and GPUs usually have several controllers. Besides, graphics cards use faster memory, so GPUs enjoy much higher memory bandwidth, which is relevant for parallel computations with huge data streams.

CPUs use caches to increase their performance owing to reduced memory access latencies, and GPUs use caches or shared memory to increase memory bandwidth. CPUs reduce memory access latencies using large caches as well as branching prediction. These hardware parts take up most of the die surface and consume much power. GPUs solve the problem of memory access latencies using simultaneous execution of thousands threads, when one thread is waiting for data from memory, a GPU can execute another thread without latencies.

There exist a lot of differences in multi-threaded operations as well. CPUs can execute 1-2 threads per core, while GPUs can maintain up to 1024 threads per multiprocessor (there are several of them in a GPU). Switching from one thread to another costs hundreds of cycles to CPUs, but GPUs switch several threads per cycle.

Using GPUs for computing demonstrates excellent results in algorithms that use parallel data processing. That is, when the same sequence of mathematical operations is applied to a large volume of data. The best results are achieved if the number of arithmetic instructions significantly exceeds the number of memory access calls.

The GPU is designed as a numeric computing engine and it will not perform well on some tasks that CPUs are designed to perform well. GPU algorithms do not perform as well on small sizes due to the overhead of the graphics Asynchronous Programming interface (API) [19]. Therefore, one should expect that most applications would use both CPUs and GPUs, executing the sequential parts on the CPU and numeric intensive parts on the GPUs. This is why the Compute Unified Device Architecture (CUDA) programming model is designed to support joint CPU-GPU execution of an application.

Another important consideration for using GPUs is the support for the IEEE 754 Floating-Point standard by the GPUs [19]. GPU support for the IEEE Floating-Point standard has become comparable with that of the CPUs. As a result, one can expect that more numerical applications will be ported to GPUs and yield comparable values as the CPUs. A major remaining issue is that the GPUs floating-point arithmetic units are primarily single precision today.

As a result of all differences described above, theoretical performance of graphics processors is much higher than CPU performance

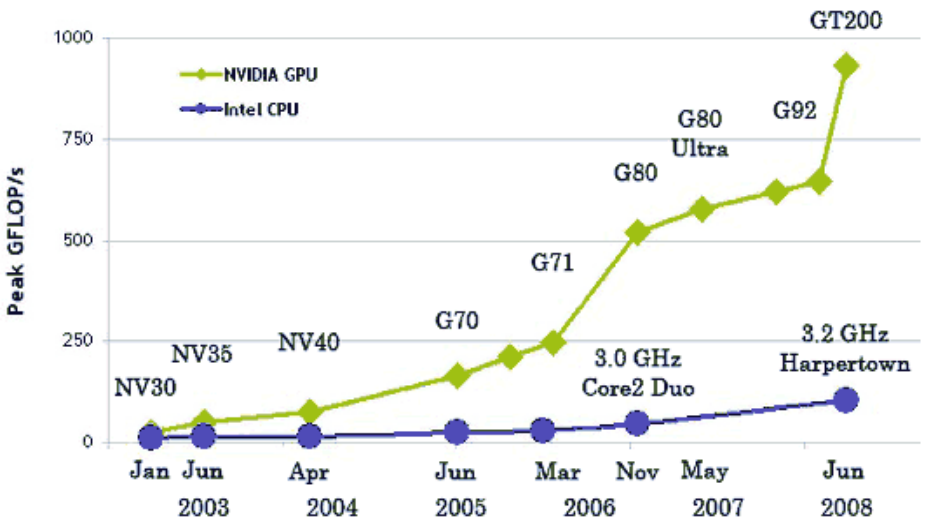


Figure 3.2 Performance between GPUs and CPUs for Single precision Floating Point operations (Based on slide 7 from GPU Physics SIGGRAPH 2007)

3.1.2 GPU ARCHITECTURE

The GPU used for this research is the NVIDIA 280 GTX based on the Tesla Architecture. The Tesla computing architecture [17] was designed by NVIDIA to speedup programs written in the Compute Unified Device Architecture (CUDA) programming language. This architecture is built around a processor array, divided into a number of Simultaneous Multithreading (SM) multiprocessors, each containing eight scalar Scalable Parallel processor cores [18]. SM is a technique permitting several independent threads to issue instructions to a superscalar processor's multiple functional units in a single cycle. The objective of SM is to substantially increase processor utilization in the face of both long memory latencies and limited available parallelism per thread. The SM multiprocessor provides a very easy low latency thread synchronization facility, which enhances efficient communication between threads in a block.

3.1.2.1 TESLA ARCHITECTURE

Figure 3.3 shows the Tesla architecture of the GeForce GTX 280. It has 240 Scalable Parallel streaming processor cores, divided into 30 Simultaneous Multithreading multiprocessors [17]. Each multithreaded SP core shares a register file of 2,048 entries and executes 128 concurrent threads; in total, the GPU executes up to 30,720 concurrent threads.

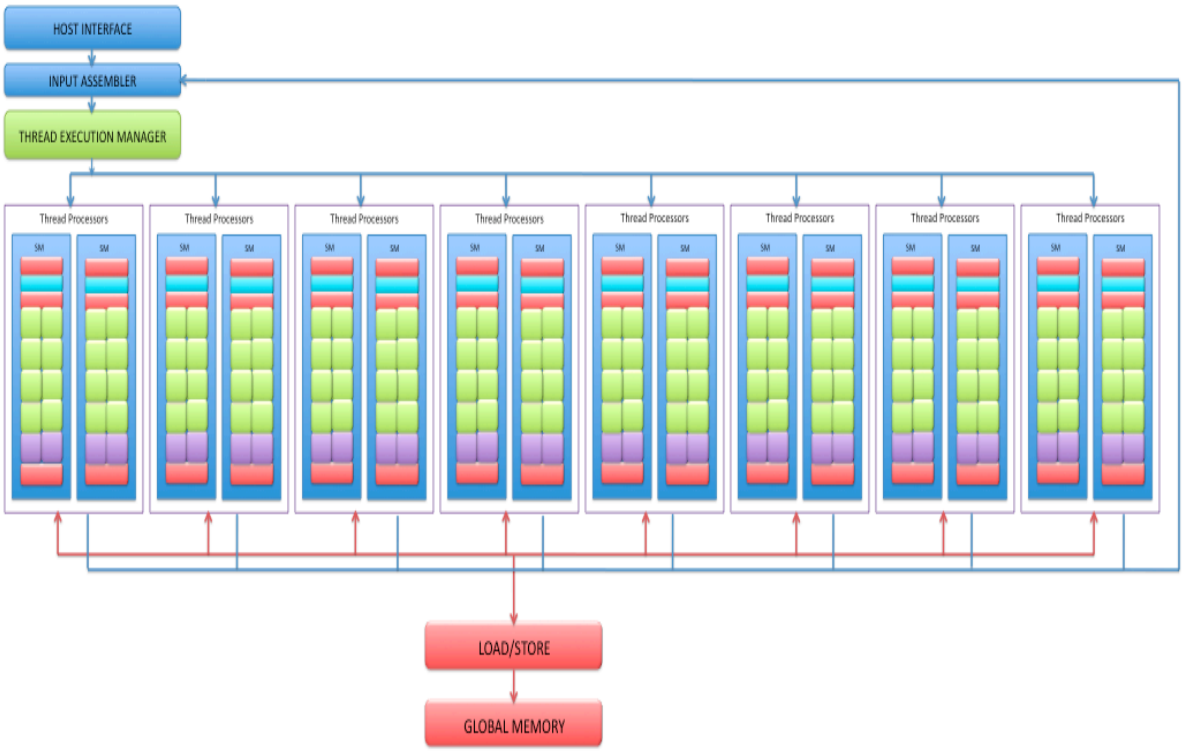


Figure 3.3 Nvidia Tesla Architecture (reproduced from [19])

The deep multithreading Tesla architecture provides latency tolerance, allowing resources to be dedicated towards throughput rather than large caches. It is preferable to operate on massively parallel number crunching problems. Creation of a thread and its scheduling and its destruction are done at negligible cost in the hardware with virtually no overhead involved in thread scheduling [18]. Threads described in the CUDA program are mapped to physical threads resident in the GPU.

3.1.2.2 SINGLE INSTRUCTION MULTIPLE THREAD

The Tesla architecture employs the Single Instruction Multiple Thread (SIMT) architecture [17]. SIMT has been developed to manage and execute hundreds of threads running different programs efficiently. In SIMT the blocks threads are divided into *warps* containing 32 threads, which at a particular time can execute only a single instruction. The threads in the warp can execute an instruction by following different code paths; this is known as divergence. The threads must wait for the other threads to finish their instructions; this is known as convergence. Each thread has its own instruction address and register state. In most typical data parallel programs, all the threads in a warp execute the same instruction, which leads to a notable gain in execution speed when implemented on hardware that employs SIMT.

The SIMT architecture is similar to a single instruction, multiple-data (SIMD) design, which applies one instruction to multiple data lanes. The difference is that SIMT applies one instruction to multiple independent threads in parallel, not just multiple data lanes [1]. In contrast to SIMD vector architectures, SIMT enables programmers to write thread

level parallel code for independent threads as well as data-parallel code for coordinated threads [17].

3.1.2.3 STREAMING MULTIPROCESSORS

Figure 3.4 shows the block diagram of a Streaming Multithreaded (SM) Processor. It is a computing and graphics processor that executes pixel shader and parallel computing programs [17]. Eight streaming processors (SP), shared memory, instruction cache, constant cache (C-Cache), two special function units (SFU) and a multithreaded instruction fetch and issue unit (MT Issue) make up the SM.

Figure 3.4 SM Multithreaded Processor (reproduced from [19])

Shared memory is a high-speed memory cache used to store the shared data for parallel computation and can be accessed by the threads. The SFUs are used for floating point operations and other atomic operations. The SPs are connected to the shared memory through a low latency network.

3.1.2.4 PARALLEL COMPUTING MODEL

The scalable Tesla architecture leads to faster execution for high throughput performance computing or data intensive applications. Thus the Tesla architecture favors the applications that are based on data parallelism, or require intensive floating-point operations or high memory bandwidth.

Figure 3.5 depicts the model for programming a data parallel application for a GPU. The program is written by partitioning the problem into grids with each grid sub divided into blocks, which are independently parallel [17]. So the SM computes the blocks while the individual threads operate on the individual elements.

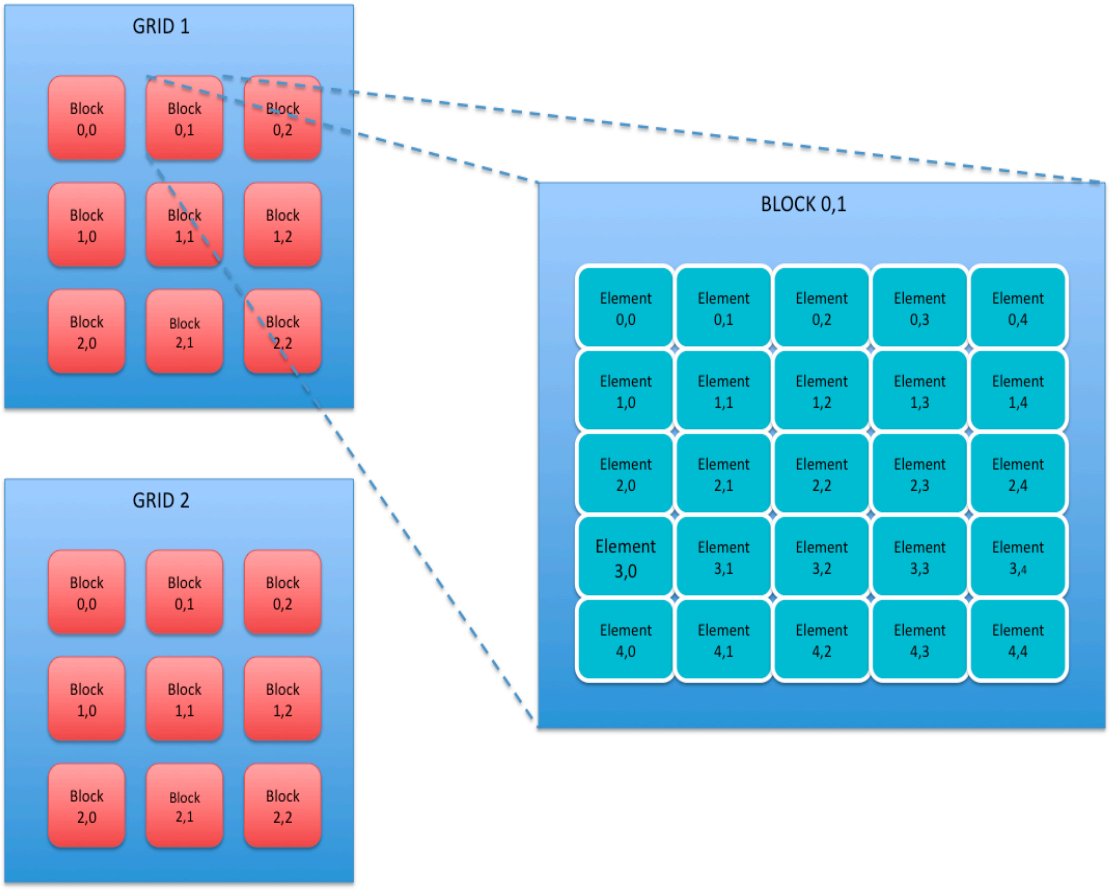


Figure 3.5 Decomposition structure of a parallel program for GPU

3.2 COMPUTE UNIFIED DEVICE ARCHITECTURE

3.2.1 INTRODUCTION

Compute Unified Device Architecture (CUDA) is a parallel computing architecture developed by NVIDIA Corporation. It is the computing engine in NVIDIA graphics processing units or GPUs. CUDA is a co-evolved hardware-software architecture that enables high performance computing developers to harness the tremendous computational power and memory bandwidth of the GPU in a familiar programming environment – the C programming language [19]. CUDA enables programmers to access this computing power without the need to be familiar with computer graphics programming like OpenGL, DirectX etc.

3.2.2 CUDA PROGRAM STRUCTURE

A basic CUDA program can be considered to be of two parts based on the platform they are executed. The parts that exhibit little or no data parallelism are implemented on the CPU and is called host code. The parts that exhibit data parallelism are implemented on the GPU and are known as device code. The NVIDIA C Compiler (NVCC) is designed to recognize and separate the two [19]. The host code is C code only. It is compiled with the host's standard C - compiler and runs as an ordinary process. The device code is written in ANSI C with extended library functions. These extended library functions are called kernels. The device code is compiled by the NVCC, and it executes on a GPU device. Kernels typically generate a large number of threads to exploit data parallelism. The

threads take very few cycles to generate and schedule due to efficient hardware support. This is in contrast with a CPU thread that typically takes thousands of clock cycles to generate and schedule.

3.2.3 ADVANTAGES

CUDA differs from older GPU programming methods in that the architecture is designed for efficient usage of non-graphics computations and uses the C programming language. It offers a new way of GPU computing that does not use a graphics Application Programming Interface (API).

CUDA has special hardware features that were non-existent in prior graphics APIs. On important feature is Shared Memory. Shared Memory a thread accessible small volume of memory (16 KB per multiprocessor). The most frequently used data can be cached using this memory [21]. Thus, it reduces memory access latencies in implementing parallel algorithms. For example, it's useful for linear algebra, fast fourier transformation, and image processing filters.

CUDA also offers more convenient access to memory. It supports the concept of scatter. Scatter is the ability to write an unlimited number of values to any addresses. Such advantages allow GPUs to execute algorithms that cannot be effectively implemented with GPGPU methods based on graphics APIs [22].

3.2.4 HARDWARE MODEL

The Tesla architecture is built around a scalable array of multithreaded Streaming Multiprocessors. When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity. The threads of a thread block execute concurrently on one multiprocessor. As thread blocks terminate, new blocks are launched on the vacated multiprocessors.

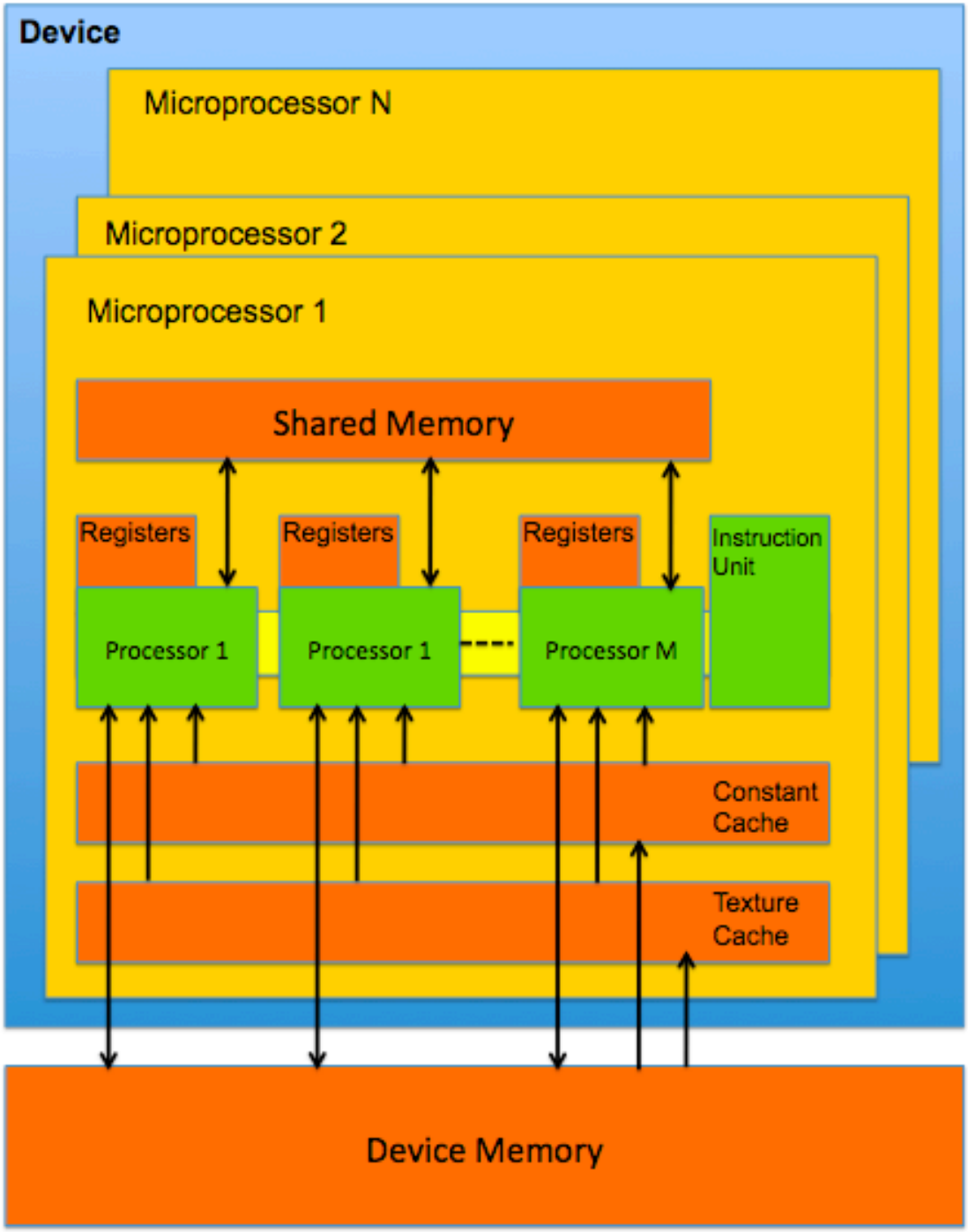


Figure 3.6 CUDA Hardware Model (adapted from [20])

3.2.5 PROGRAMMING MODEL

The CUDA programming model is very well suited to exploit the parallel capabilities of GPUs. The latest generation of NVIDIA GPUs, based on the Tesla Architecture. It supports the CUDA programming model and tremendously accelerates CUDA applications [20].

CUDA uses a parallel computing model, in which each of the SIMD processors executes the same instruction over different data elements simultaneously. A GPU is a computing device acting as a co-processor (device) for a CPU (host), possessing its own memory, and processing multiple threads in parallel.

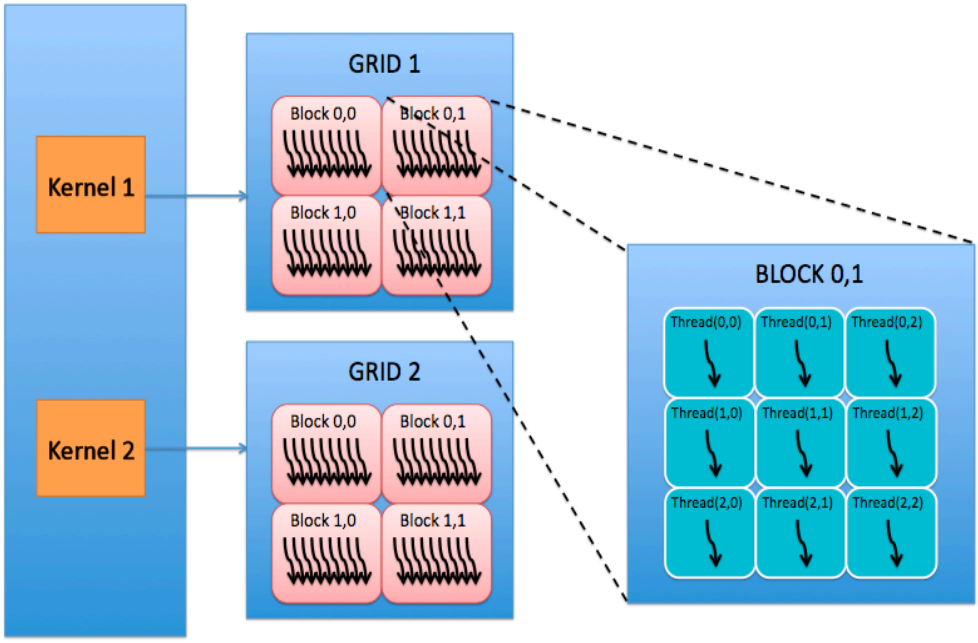


Figure 3.7 CUDA Programming Model

The CUDA programming model enables the grouping of threads. Threads unite into thread blocks, one- or two-dimensional grids of threads that interact with each other via shared memory. A program (kernel) is executed over a grid of thread blocks as illustrated in Figure 3.7, one grid is executed at a time. Each block can be one-, two-, or three-dimensional in form, and it may consist of 512 threads. Thread blocks are executed in the form of small groups called warps (32 threads each).

Grouping blocks into grids helps avoid the limitations and applies the kernel to more threads per call. It also helps in scaling. If a GPU does not have enough resources, it will execute blocks one by one. Otherwise, blocks can be executed in parallel, which is important for optimal distribution of the load on GPUs of different levels.

3.2.6 CUDA MEMORY MODEL

The CUDA memory model allows byte wise addressing and support for scatter and gather [20]. There are up to 1024 registers per streaming processor. Access to these registers is very fast. These registers can store 32-bit integer or single precision floating-point numbers. The CUDA memory model is shown in Figure 3.8 and is summarized in the following sections.

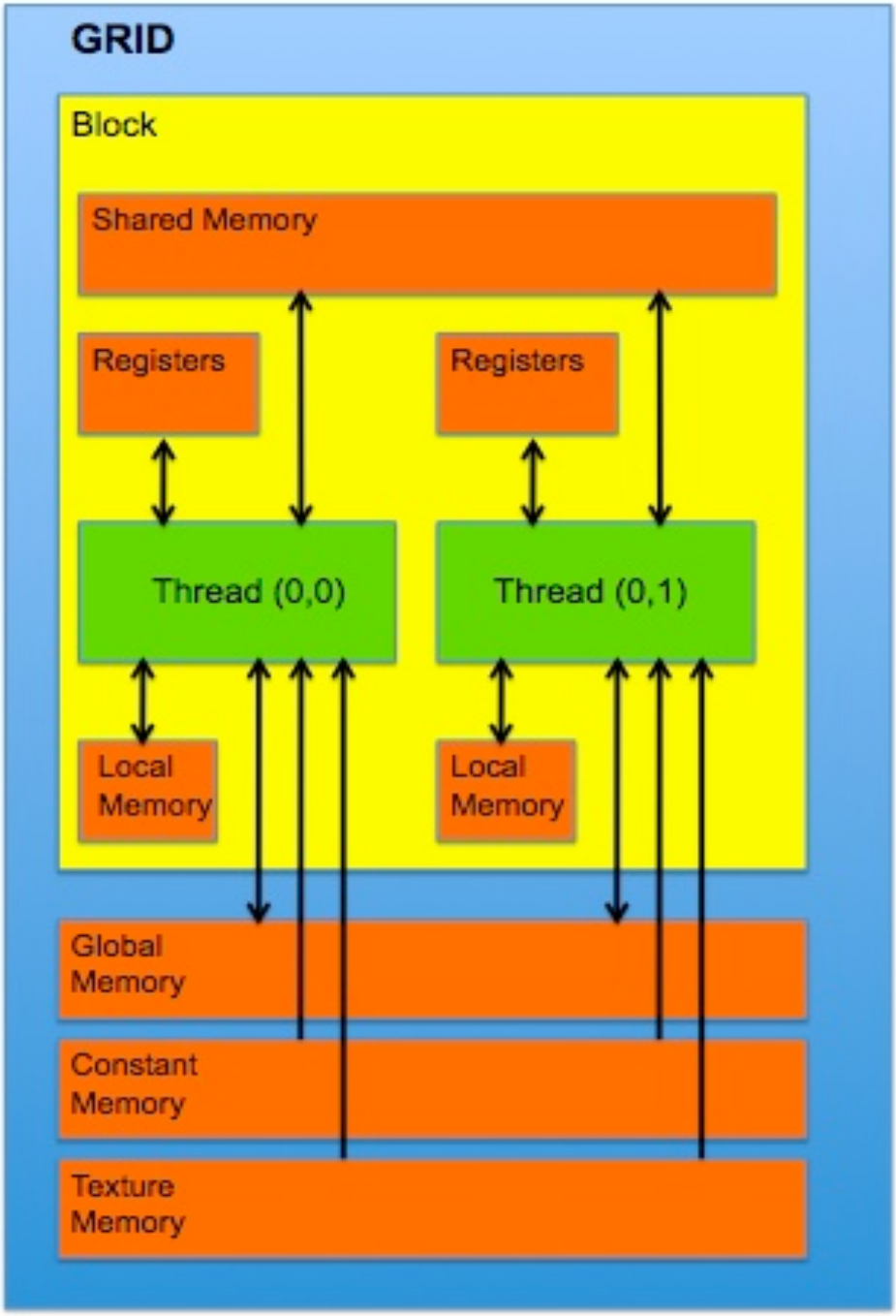


Figure 3.8 CUDA Memory Model (reproduced from [20])

Global Memory is the largest volume of memory available to all multiprocessors in a GPU. It's size ranges from 256 MB to 4 GB (up to 4 GB in Tesla) [19]. It offers high bandwidth. High-end solutions from Nvidia have bandwidths exceeding 100 GB/s to access the Global Memory. Global Memory suffers from very high latencies (several hundred cycles) [19] and is not cached. It is read/write accessible to grids.

Local Memory is a small volume of memory, which can be accessed by a single streaming processor. Its bandwidth is low and is limited in size. Like Global Memory, Local Memory is not cached and thus the accesses incur large penalties [20]. Local Memory is read/write accessible to threads.

Shared Memory is a 16-KB memory accessible to all streaming processors. Like registers, shared memory is fast access (low - latency). This memory enables interaction between threads [20]. Advantages of shared memory include its usage as a controllable L1 Cache, reduced latencies for ALUs accesses, and fewer calls to global memory [20]. Shared memory is read/write accessible to blocks.

Constant Memory is read only. It has a memory area of 64 KB [2]. It's cached by 8 KB for each multiprocessor. Reading from the constant cache is as fast as reading from a register as long as all threads read the same address [20]. On a cache miss it is as slow as global memory [20].

Texture Memory is also a read only memory and is available for reading to all multiprocessors. The texture units in a GPU fetch data. Texture Memory is cached by 8

KB for each multiprocessor. Texture Memory costs one read from texture cache but on a cache miss it is as slow as global memory [20].

Global, local, texture, and constant memory are physically the same memories as local video memory of a graphics card. They differ only in caching algorithms and access models. CPUs can refresh and access only the external memory, i.e. the global, constant, and texture memories.

4 RELATED WORK

Online rebinning is an important and well-established technique for reducing the time required to process PET data. It is a common part of PET data acquisition and is necessary for downstream support of on-line histogramming. On-line rebinning and histogramming serve to minimize post processing and speed the delivery of medical images [23].

4.1 PRESENT SYSTEM

Most Siemens PET tomographs shipped in the past 5 years use the Petlink DMA Rebinner (PDR) card for online rebinning. The PDR was designed by CPS Innovations, Inc. prior to the company being acquired Siemens Medical Solutions USA, Inc in 2005. Figure 4.1 is the present system diagram showing the PET Gantry and the PET Data Acquisition System containing the Petlink DMA Rebinner (PDR) card. The PDR is a PCI card designed for general purpose, rapid-yet-accurate, on-line LOR-to-bin mapping of PET coincidence data. On the PDR, LUTs service the computations required for mapping from detector-pair space into projection data space. The output of this pipeline is returned to the router FPGA and is then typically output by direct memory access in 32-bit bin address packet form. Besides rebinning, the PDR card is also used for data acquisition of detector pair event words and tag packets received from the gantry.

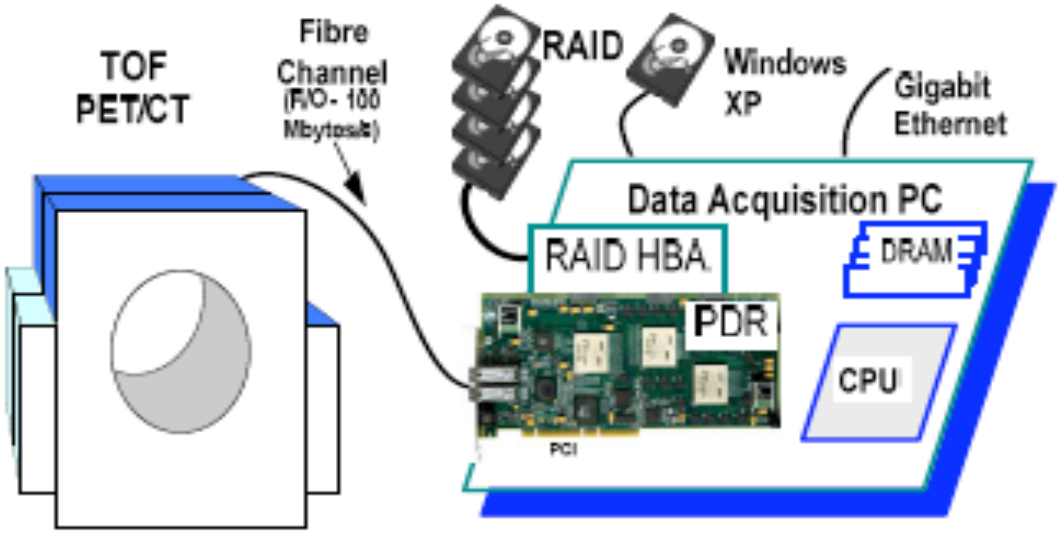


Figure 4.1 Present system with PET Gantry and Acquisition system with PDR card

(Reprinted from [25] with permission)

4.2 PDR HARDWARE DESCRIPTION

Principal interfaces for the PDR include: dual Fibre Channel transceivers for streaming of PETLINK™ packets, 64/66-MHz PCI bus for DMA transfers of PETLINK™ detector-pair and bin address packets, and an RS-232 serial port for "real-time" updates of gating and rotational position information [23]. Figure 4.2 shows a snapshot of the PDR. It is a PCI bus interface board [23]. There are two Xilinx Virtex II Pro FPGAs and twenty 4M x 16 bit flash memory devices on the board that are used to carry out main computation and processing tasks. A third Xilinx Virtex II Pro is used to interface the interface devices with Fibre Channel, PCI, and RS 232 [23], provide register interface, and other logic functions.

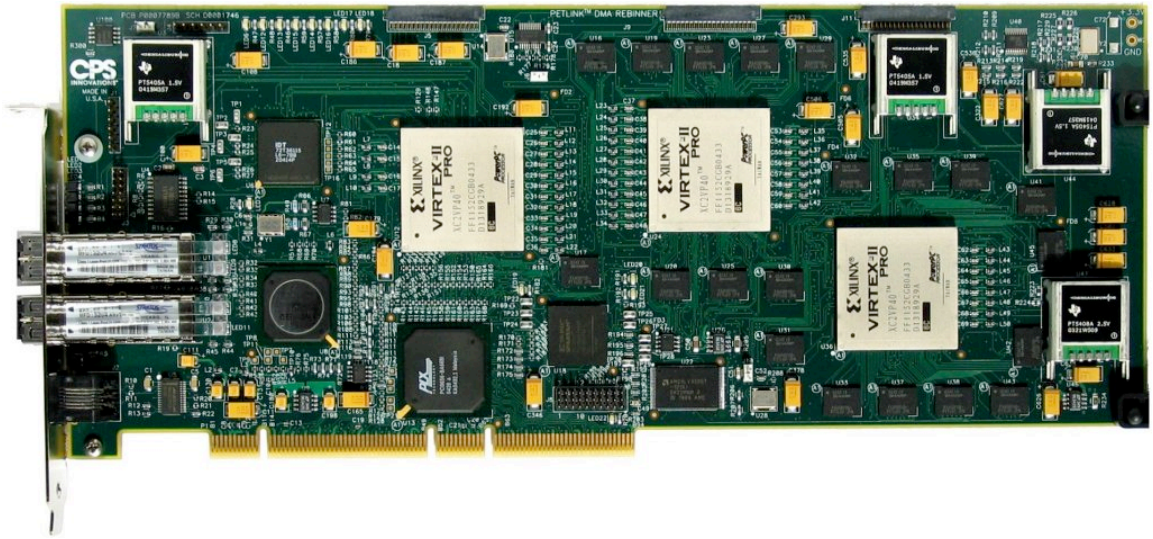


Figure 4.2 Snapshot of a PDR (reprinted from [23] with permission)

4.2.1 PROCESSING

A sequence of transformations is required to convert a detector-pair packet to a bin address packet. The on-line rebinning function requires a sequence of transformations to convert a detector-pair packet to a bin address packet. Typical transformations include: the calculation of the physical location and angular aspect of each line of response (LOR) defined by each incoming detector-pair packet, dynamic determination of "emission" and "transmission" LORs using a process known as "rod windowing," and a "nearest-neighbor" calculation to map each detector pair into the appropriate bin in the 3-D bin-address space [23]. In the PDR, a pipeline-architecture is employed to accomplish the sequence of transformations required for a given algorithm. Each stage of the pipeline is dedicated to a specific part of the sequence. A stage may employ one of multiple LUTs as processing engines (flash memory devices). The maximum clock rate for the pipeline is limited by the minimum access time of the flash memory [23].

4.2.2 DATAFLOW

Figure 4.3 represents the PDR block diagram showing the chip architecture. The PDR receives and/or transmits event packets on dual Fiber Channel fiber-optic links as well as a 64/66 PCI interface. The PDR contains flash memory and FPGA devices to form a pipeline [24]. Figure 4.4 is a sample rebinning pipeline diagram based on the model proposed by Jones et al. [25]. The algorithm is divided into 3 stages distributed across the two FPGAs, which are referred to as "group" FPGAs and are designated as Group A and Group B [23]. The thick vertical lines "1 2 3 4" represent the pipeline holding registers.

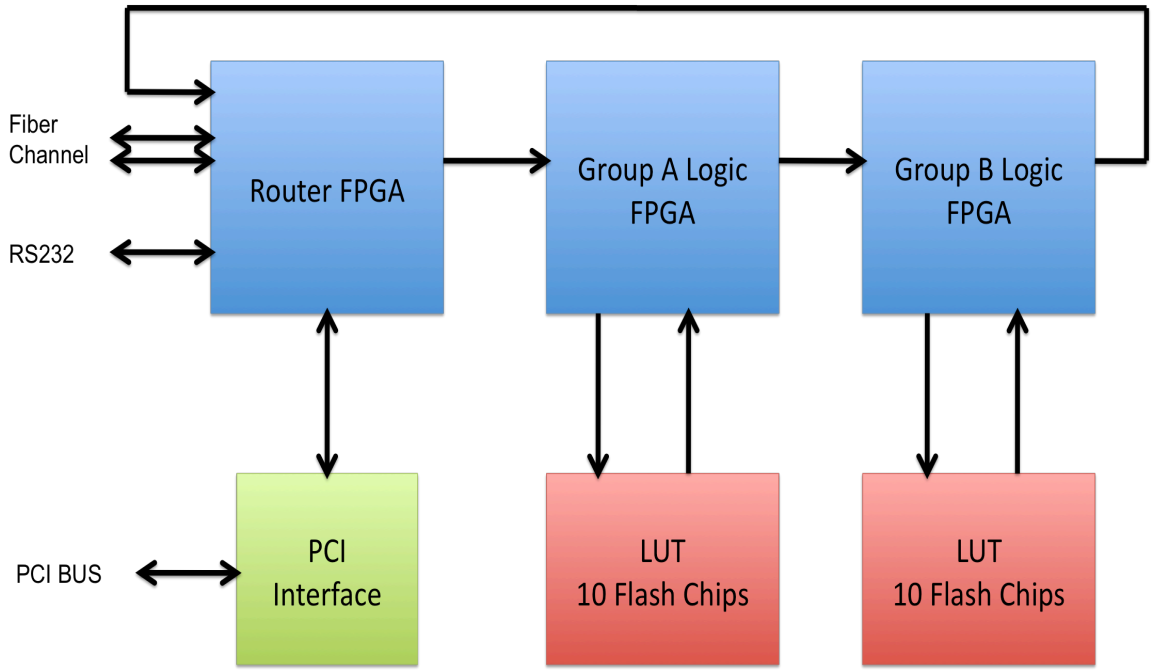


Figure 4.3 PDR Block Diagram (reproduced from [25])

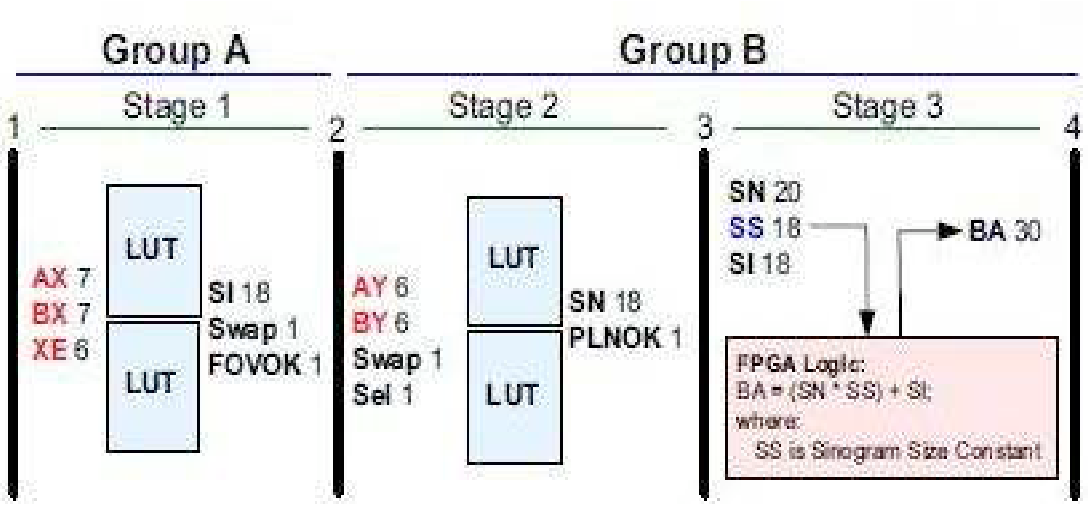


Figure 4.4 Stage Digital PDR pipeline (reprinted from [25] with permission)

Each LUT represents a separate Flash memory chip, which is programmed with the LUT content. Text at the left of each LUT box represent address input fields with bit size shown. At the right of each LUT box are data output fields with bit size shown. Stage 1 is part of the transaxial processing stage. The AX, BX and XE fields from each incoming detector-pair packet are applied to the LUT address lines to produce the sinogram index (SI), "swap" bit (SWAP), and "field-of-view OK" indicator (FOVOK).

Stage 2 is implemented in the Group B FPGA. Stage 2 is part of the axial processing stage. It uses intermediate results computed from Stage 1 as well as the AY and BY fields from the original input packet to compute the sinogram number (SN) and the "plane OK" indicator (PLNOK). The final stage of the pipeline does not use a LUT engine; rather, FPGA slices and hardware multipliers are used to compute the bin

address. Finally, only those packets having both FOVOK and PLNOK set are passed through for acquisition; all other packets (except for tag packets) are discarded [23].

The PDR card has proven to be effective for tasks in PET rebinning but can only achieve a pipeline throughput of 13 to 15 Million Events/sec [25]. While adequate for most clinical PET applications, it is not quite adequate for very high-count rate applications such as the latest PETs, which can approach 20 Million Events/sec. This leads us to explore new architectures that can achieve higher pipeline throughputs for a faster and effective rebinning.

5 PET PHYSICS AND REBINNING

When a positron undergoes annihilation with an electron, their rest masses are converted into a pair of annihilation photons. These photons have identical energies (511 KeV)[1] and are emitted simultaneously in ~180 degree opposing directions. These photons may be detected by the detectors, which are linked so that two detection events unambiguously occurring within a certain time window may be called a coincident and thus be determined to have come from same annihilation. This is known as Coincidence Detection. These coincident events can be stored in arrays corresponding to projections through the patient and reconstructed using tomographic techniques. The resulting pictures do not show as much detail as Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) because the pictures show only the location of the tracer. The PET picture may be matched with those from a CT scan to get more detailed information about where the tracer is located.

5.1 COINCIDENCE DETECTION

In a PET, each detector generates a timed pulse when it registers an incident photon. These pulses are then combined in coincidence circuitry, and if the pulses fall within a short time-window, they are deemed to be coincident. A coincidence event is assigned to a line of response (LOR) joining the two relevant detectors.

Event Detection in PET Detector is based on electronic collimation. An event is valid if it satisfies the following

I. The two photons are detected within a predefined electronic time window (coincidence window), which is typically 6 to 10 nanoseconds (1 ns = 10^{-9} sec).

II. The subsequent LOR formed between them is within a valid acceptance angle of the tomograph.

III. The energy deposited by both the photons in the crystal is within the selected energy window.

Figure 5.1 shows a pair of detectors in PET. The shaded area depicts the area from which a near simultaneous annihilation photons can be detected by the detectors. Not all annihilations are detected since it is necessary that both photons strike the detectors.

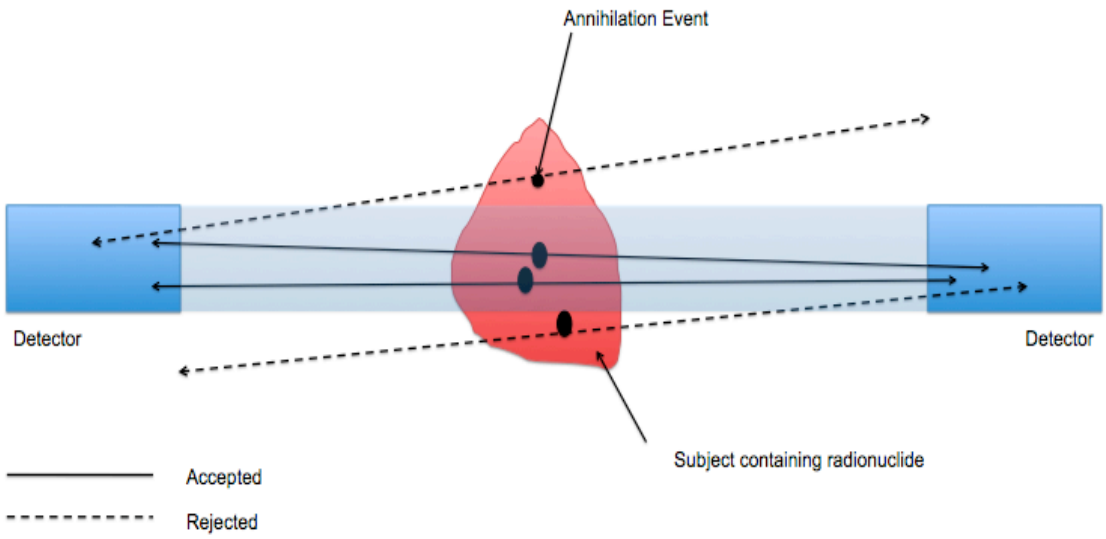


Figure 5.1 Coincidence Detection

5.2 TIME OF FLIGHT (TOF)

In a PET scan when a positron collides with an electron, two gamma rays are generated in the annihilation process. The detector ring is looking for two, almost "simultaneous" rays, which is then noted as an event and is stored as data. Therefore using the time difference (fraction of a second) between the gamma rays, the scanner can pinpoint the original location of the positron-electron annihilation.

As we know that when each nucleus decays, a positron is released that immediately collides with an electron, creating an annihilation that releases a pair of photons, or gamma rays. These two photons travel away from the collision point at 180° from each other. After detecting the photons, the PET scanner's computer uses that information to calculate where the radioactive agent is concentrated and produce an image localizing the affected area. TOF makes it possible for the point of origination of annihilation to be more accurately predicted, which leads to more accurate imaging. Improved event localization reduces noise in image data, resulting in higher image quality.

If the difference in the arrival times is Δt and c is the velocity of light then the location of annihilation event with respect to the midpoint between the two detectors s given by [1]

$$\Delta d = \frac{\Delta t * c}{2} \quad [1]$$

5.3 TYPES OF COINCIDENT EVENTS

Coincident events in PET can be categorized into four types: true, scattered, random and multiple.

5.3.1 SCATTERED COINCIDENCE

A scattered coincidence is one in which at least one of the detected photons has undergone at least one Compton scattering event prior to detection as shown in Figure 5.2 Since the direction of the photon is changed during the Compton scattering process, it is highly likely that the resulting coincidence event will be assigned to the wrong LOR [1]. Scattered coincidences add a background to the true coincidence distribution, which changes slowly with position, decreasing contrast and causing the isotope concentrations to be overestimated [1]. The number of scattered events detected depends on the volume and attenuation characteristics of the object being imaged and on the geometry of the camera [1]

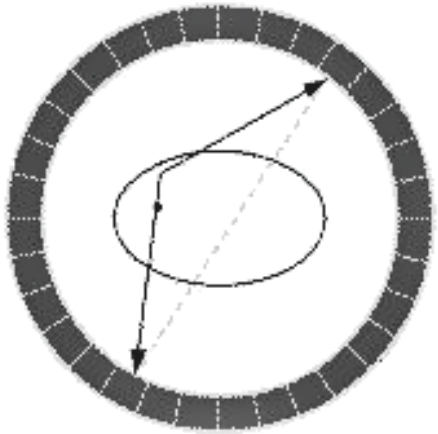


Figure 5.2 Scattered Coincident Event (adapted from [1])

5.3.2 TRUE COINCIDENCE

A coincident event is considered to be true when both photons from an annihilation event are detected by detectors in coincidence, neither photon undergoes any form of interaction prior to detection, and no other event is detected within the coincidence time-window. Figure 5.3 depicts the true event scenario.

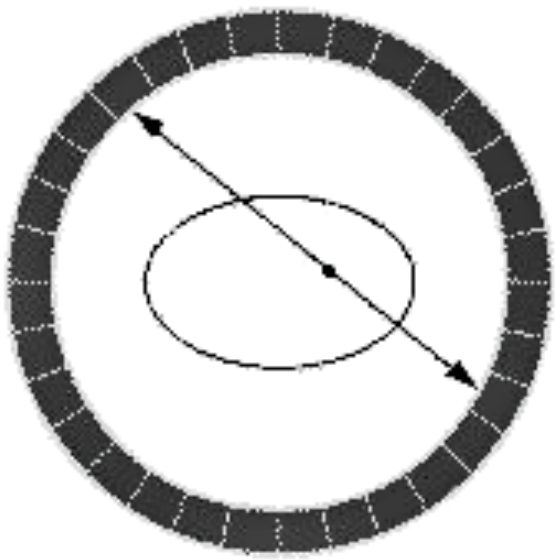


Figure 5.3 True Coincident Event (adapted from [1])

5.3.3 RANDOM COINCIDENCE

When two photons not arriving from the same annihilation are incident on the detectors, it is considered to be random event as shown in Figure 5.4. The number of random coincidences in a given LOR is closely linked to the rate of single events measured by the detectors joined by that LOR. The rate of random coincidences increases roughly with the square of the activity in the FOV [1]. The number of random events depends on the volume of the subject being imaged [1].

It is not possible to determine the LOR when more than two photons are detected in different detectors within the coincidence time window and thus the event is rejected.

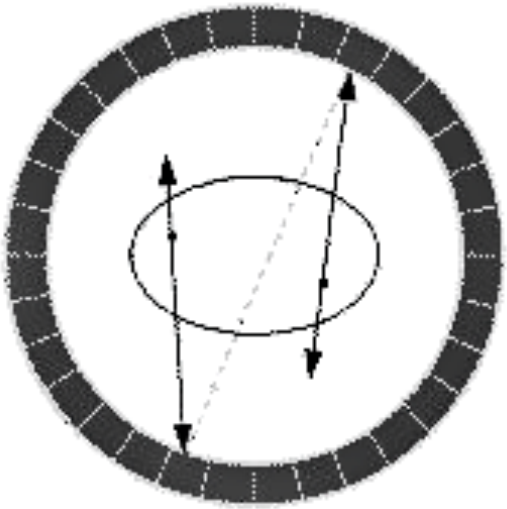


Figure 5.4 Random Coincident Event (adapted from [1])

5.4 DATA ACQUISITION FOR PET – 2 D MODE AND 3 D MODE

Most of the cameras in PET technology employ either two-dimensional or three-dimensional modes. In two-dimensional (2-D) mode a thin septa usually of lead separate is used. Events are only recorded between detectors within the same ring or lying in closely neighboring rings. Coincidences between detectors in closely neighboring rings are summed or rebinned to produce a dataset consisting of 2P + 1 coplanar sets of LORs normal to the axis of the camera, where P is the number of detector rings. This dataset may be reconstructed into images using standard tomographic techniques.

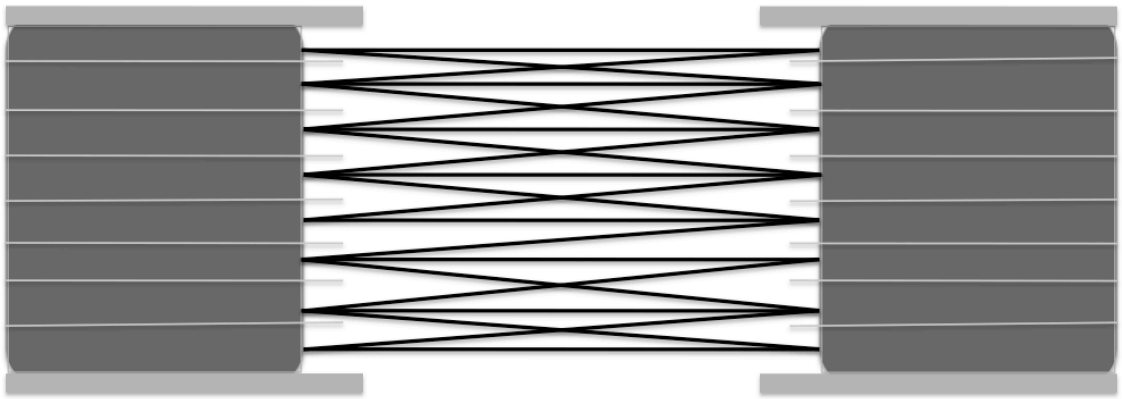


Figure 5.5 2D – coincidences between detectors in the same ring or neighboring rings.

In three-dimensional (3-D) mode the septa are absent and events are detected between detectors lying in any ring combinations and thus the Field of View (FOV) for events is increased [1]. This can result in significant increase in the number of random coincidences. The image reconstruction from the 3D mode dataset is computationally intensive. The computational intensiveness increases with the number of rings employed. Retraction of septa allows the tomograph to count a much larger number of LORs resulting in a significant increase in sensitivity that allows for less acquisition time. As seen in Figure 5.6 the 3D mode allows for counting LORs between any ring combinations.

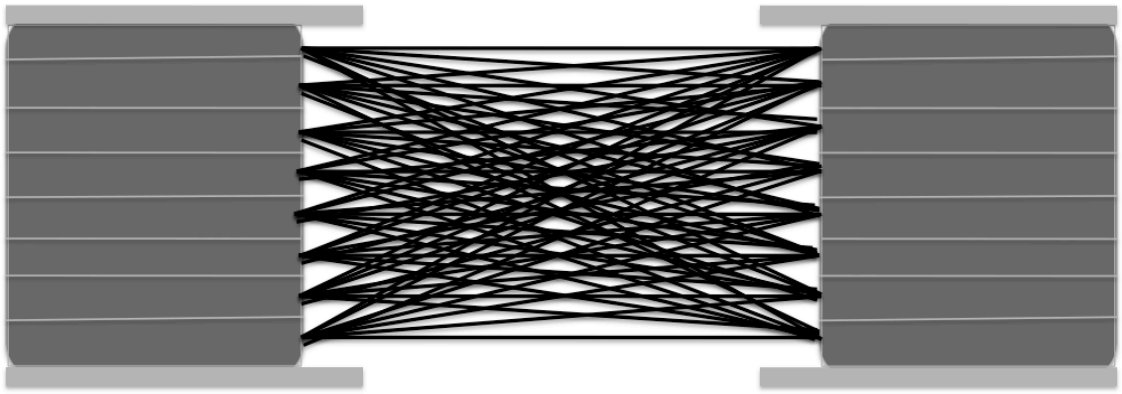


Figure 5.6 3D – coincidences between any pair of rings permitted

5.5 SINOGRAM GENERATION AND REBINNING

5.5.1 COORDINATES

An annihilation that occurs at a point in the FOV will result in two oppositely directed gamma rays being emitted along a line in any direction. When a pair of crystals detects the gamma emissions, the line or LOR between the involved crystals is identified by variables (X_r, ϕ). Since the gamma emission can be of any direction, the X_r coordinate is limited only by the extent of the object scanned and ϕ can be any angle. ϕ is commonly referred to as the projection angle.

X_r and ϕ coordinates are used in a very useful plot known as a sinogram. A sinogram is a 3-D plot of the total number of LORs for each (X_r, ϕ) of the tracer activity in an object. The number of LORs that occur for a particular (X_r, ϕ) is the total emission activity along that direction in the object for a particular transaxial plane.

By allowing for negative values of X_r, ϕ can be limited to 180°. As shown in Figure 5.7, LOR A would be identified by sinogram values (X_r, ϕ_1) whereas LOR C would have sinogram values ($-X_r$, ϕ_1). LORs B & D would have the same angle as A & C, but different values of X_r.

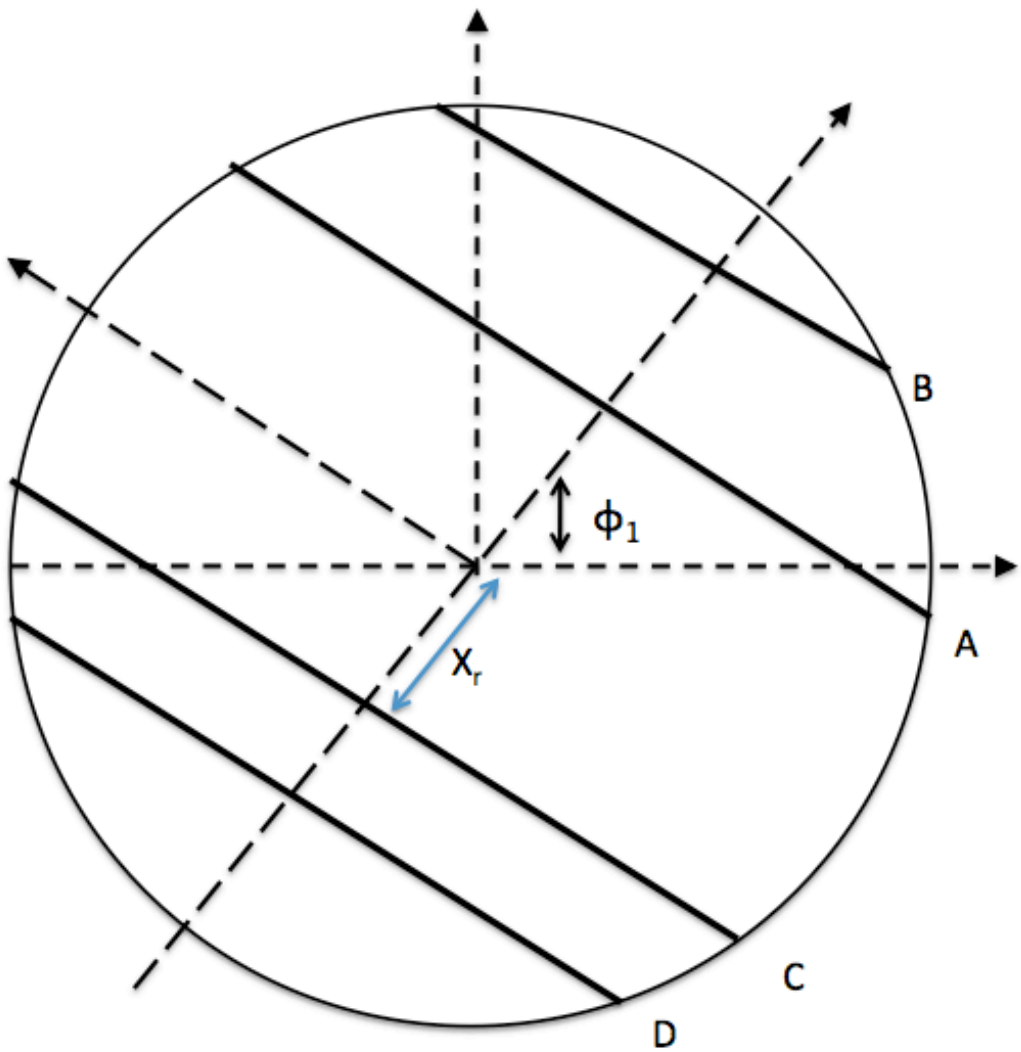


Figure 5.7 LOR and sinogram coordinates

An important point to notice is that a LOR (a line) in the gantry is a point in a sinogram. X_r and ϕ can be viewed as a way of describing the direction of an LOR. An important point to remember is that the LORs can be detected is limited by the number of crystals in the scanner and are pre-identified and stored in the scanner software. The hardware

only has to detect the gamma energy and determine which pairs of crystals are in coincidence. The software then identifies and counts the LORs defined by the coincident crystal pairs.

The LORs detected over a period of time can be plotted in sinogram format as shown in Figure 5.8. ϕ is limited to 0 to 180 degrees in the CCW direction. X_r can have positive and or negative values. X's indicate LORs A, B, C and D from Figure 5.7. Notice that each LOR in the sinogram plot is a point and they all lie on a horizontal line at the same angle ϕ_1. If all the LORs that the scanner is capable of identifying at a projection angle ϕ were plotted, they would all lie on a horizontal line at the position of the projection angle ϕ. This data is called a projection.

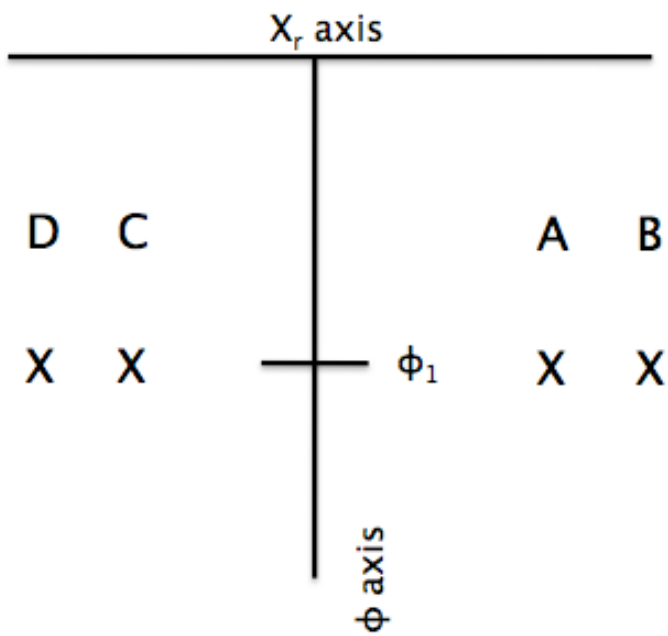


Figure 5.8 LORs plotted in sinogram format

If any LOR is detected multiple times, the "color" (either on a gray or color scale) is changed for that particular point. Hence the sinogram of an object is a 3D plot of the occurrence of LORs that the scanner is capable of detecting.

5.6 IMAGE RECONSTRUCTION

The number of counts for a particular LOR joining a pair of crystals is proportional to the summation of total activity in the object along the LOR. The set of LORs that are parallel to each other and their respective total counts is known as a projection. For a point source, this gives rise to a series of intensity profiles as shown in Figure 5.9.

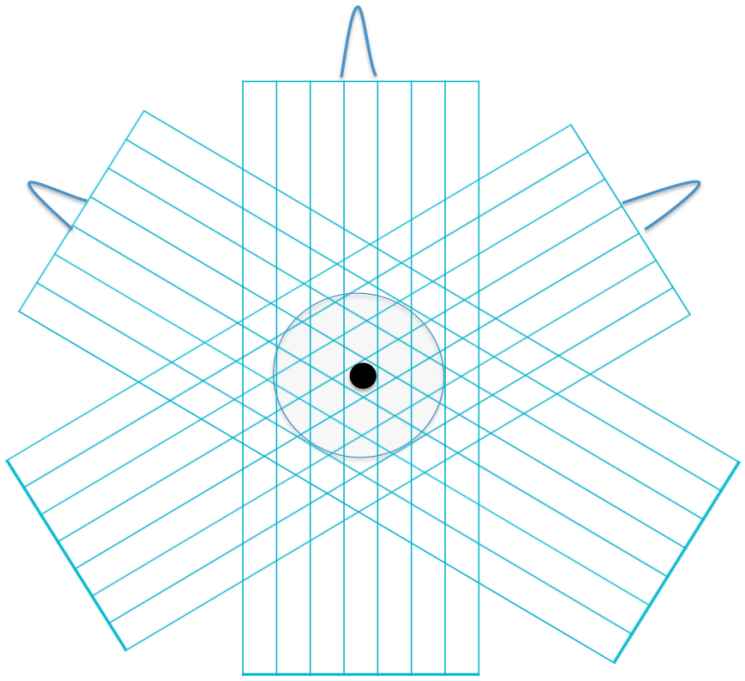


Figure 5.9 projections generated from a single point source

A process known as backprojection may obtain an estimate of the original source distribution. In the process the values of each projection are “painted” along the LORs to which they correspond. Backprojection for a single point source is shown in Figure 5.10.

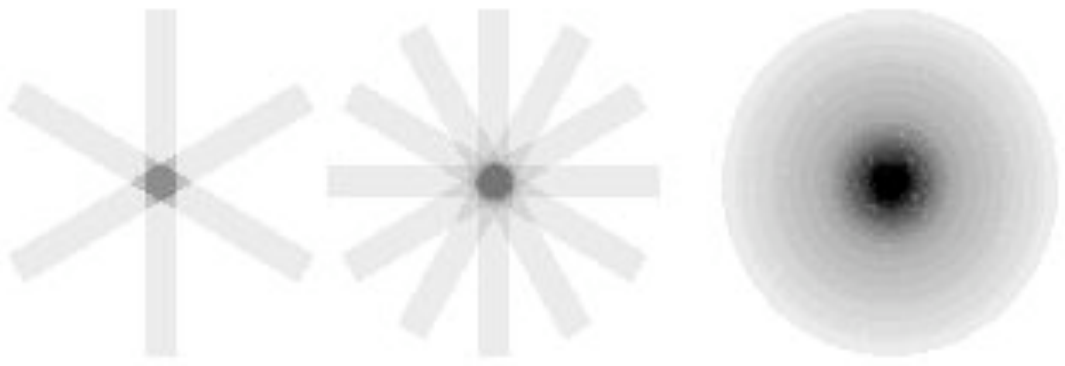


Figure 5.10 Back-projections of a point source (reproduced from [1])

5.7 3D PET DATA

One of the most common methods of event accumulation during a PET scan is to allocate a large memory array such that there is an array element for each LOR that can be measured in a scan. These array elements are initialized to zero. When a coincident event is detected, the array element corresponding to the LOR is incremented by one. Thus the array elements storing the counts are called bins and the array is called a histogram. Histogramming typically involves some loss of information as the bins only store the total number of events that have been grouped together based on some common properties. The collection of bin values for LORs that are parallel to each other for a given angle ϕ is called a projection. This data collected can be stored and displayed as an array indexed by vales x_r and ϕ. These are called Direct Sinograms. The data is stored for the range $0 \leq \phi < \pi$.

In a multi ring scanner a separate coordinate z is needed to identify the rings involved. It is the point midway between two detectors in coincidence [26]. Hence $z = (r1 + r2)/2$ where r1 and r2 are the axial coordinates of detectors. In 3D mode, another variable is needed to identify the ring difference between the coincident detectors because it is possible to have LORs with the same z coordinate, but that involve detectors of different rings. Let $\Delta r = r1 - r2$ be the ring difference since it represents the axial spacing between detectors in coincidence.

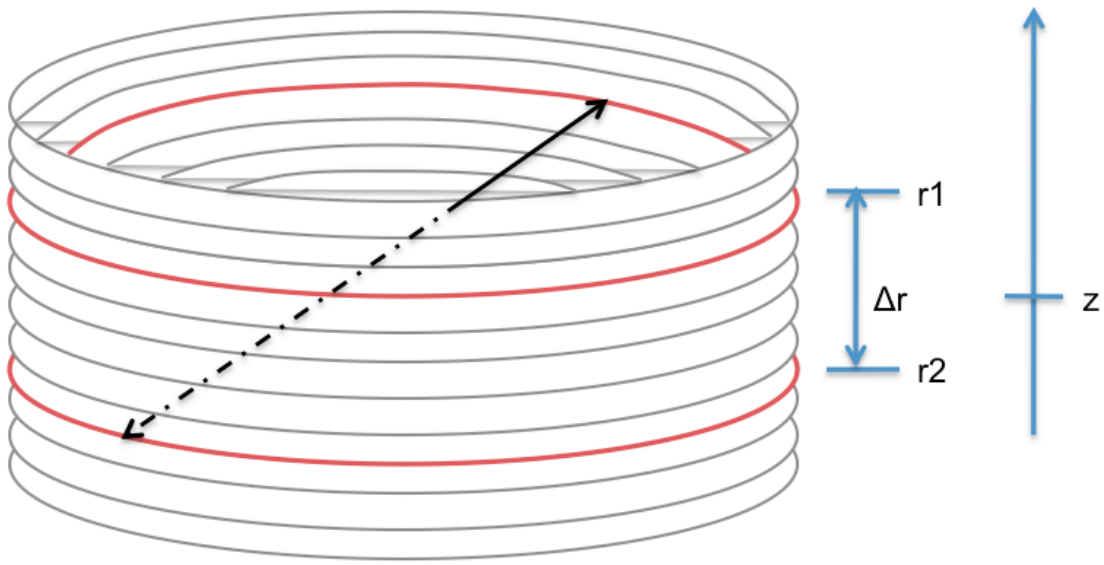


Figure 5.11 Δr and z coordinates of an LOR in a multi ring scanner

Since x_r and ϕ refer to coordinates in a transaxial plane, the four coordinates (x_r, ϕ, z, Δr) refer to the projection of a LOR onto a transaxial plane located at z as shown in Figure 5.12. The histogram of the number of LORs detected for a particular (z, Δr) is called an oblique sinogram. If Δr = 0, an oblique sinogram is direct sinogram [26].

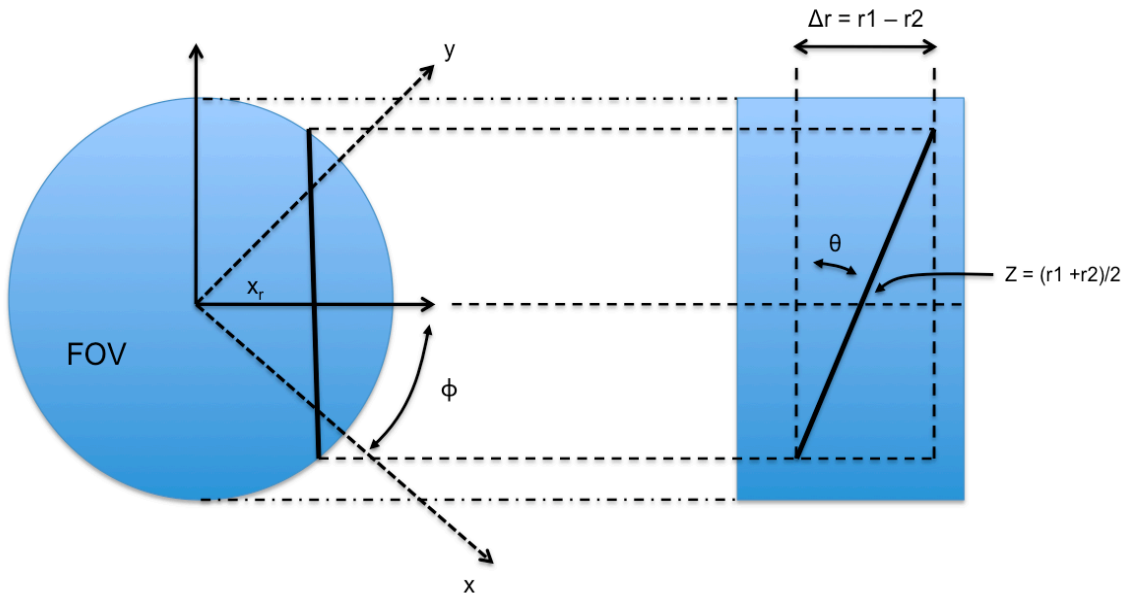


Figure 5.12 Transaxial view (left) and longitudinal section (right) of scanner showing the projection of an LOR onto the transaxial plane located at point z

5.8 REBINNING

A Rebinning algorithm is defined as an algorithm, which sorts the 3-D data into a stack of ordinary 2-D data sets, where for each transaxial slice the 2-D data are organized as a sinogram. The 3-D image is then recovered by applying to each 2-D slice a back projection method [26]. This approach allows for significant speedup of 3-D reconstruction, which is particularly useful for applications involving dynamic acquisitions.

Due to the large size of a full 3D PET data set, a single projection set can occupy 50 to 100 bytes. Thus the need to reduce data size becomes important. The time required to

reconstruct an image from 3D data is more than an order of magnitude longer than the time necessary to reconstruct 2D Data. This is due to a considerable increase in the number of LORs than need to be back projected. A process called rebinning can accomplish the data set reduction. Because the size of individual crystals is small, the angle of θ in Figure 5.12 as represented by Δr is small. The angular difference between adjacent values of Δr is small enough to allow the merging of oblique sinograms whose Δr differ by a small integer into a single sinogram. This process substantially reduces the number of sinograms. Thus rebinning can be described as the process of merging of sinograms with the same value of z and adjacent values of Δr. Figure 5.13 illustrates this concept where N is the number of axial crystal rings.

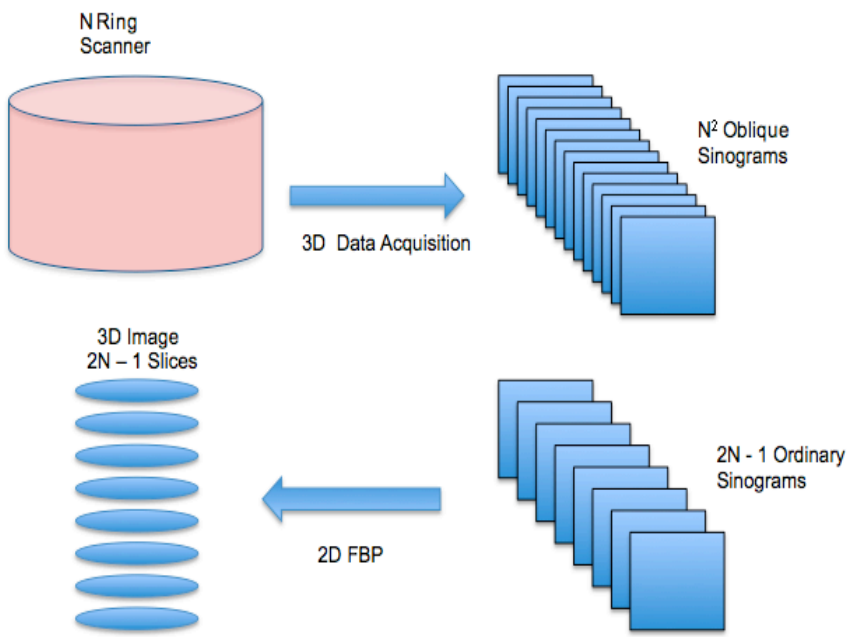


Figure 5.13 Principal of rebinning algorithm

6 IMPLEMENTATION

6.1 INTRODUCTION

The implementation of the rebinning algorithm on the GPUs was based on the GPU programming model, which is different from those used on a CPU. In contrast to the serial programming models that are usually employed on the CPU, GPU programming is based on the stream model [28]. In the stream model, data are represented as streams, and kernels perform computation. Each kernel operates on the whole stream of data, instead of individual elements. NVIDIA's Compute Unified Device Architecture (CUDA) and current GPUs offer massively parallel processing capability that can handle such computational complexity, as is characteristic of coincidence and TOF rebinning. CUDA is a co-evolved hardware-software architecture that enables high performance computing developers to harness the tremendous computational power and memory bandwidth of the GPU in a familiar programming environment – the C programming language [19].

6.2 REBINNING ALGORITHM

Rebinning algorithms involve the computation of several elementary functions such as sine, cosine, and arctangent and floating point operations. Such functions are easily computed on standard CPU-based architectures (PC and server platforms). While such architectures are ideal for post-processing, they are not well suited for real-time processing of PET data, especially high count-rate applications. More elaborate CPU-based architectures could be employed for such tasks, but they are presently not cost

effective for the clinical market. The present CUDA based GPUs support IEEE format single precision numbers and trigonometric functions [4] and are thus more favorable to implement these operations at low cost and a fast rate. Other optimizations like memory coalescing further increase the performance of the algorithm on the GPUs. Figure 6.1 and Figure 6.2 are pipelined rebinning algorithms for implementation on the PETLINK™ DMA Receiver (PDR) proposed by Jones et al. [27]. On the PDR, FPGAs make use of the LUTs stored in the 4Mx16-bit flash memory chips for high-speed computation.

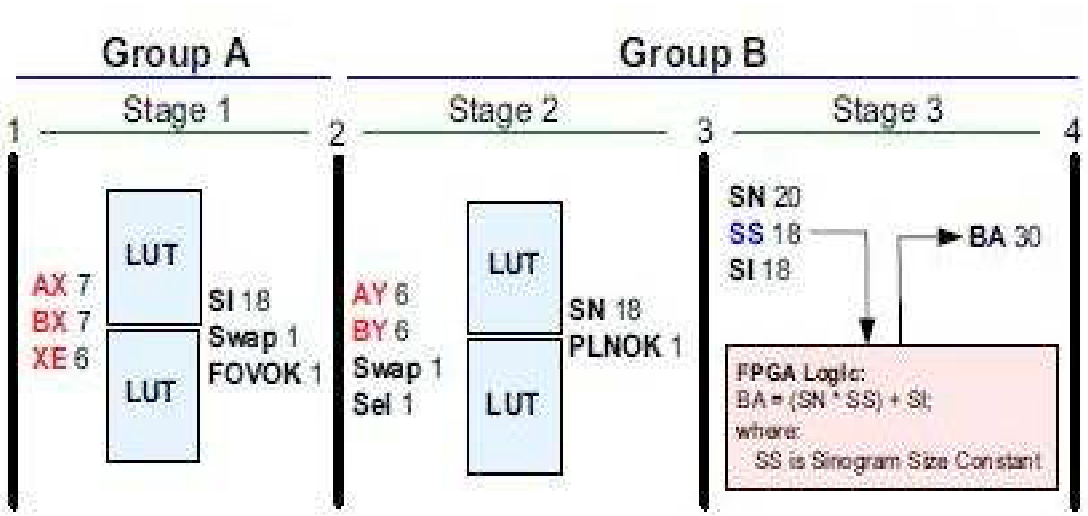


Figure 6.1 3-stage PDR Digital Pipeline as applied to Coincident event system. (reproduced from [25] with permission)

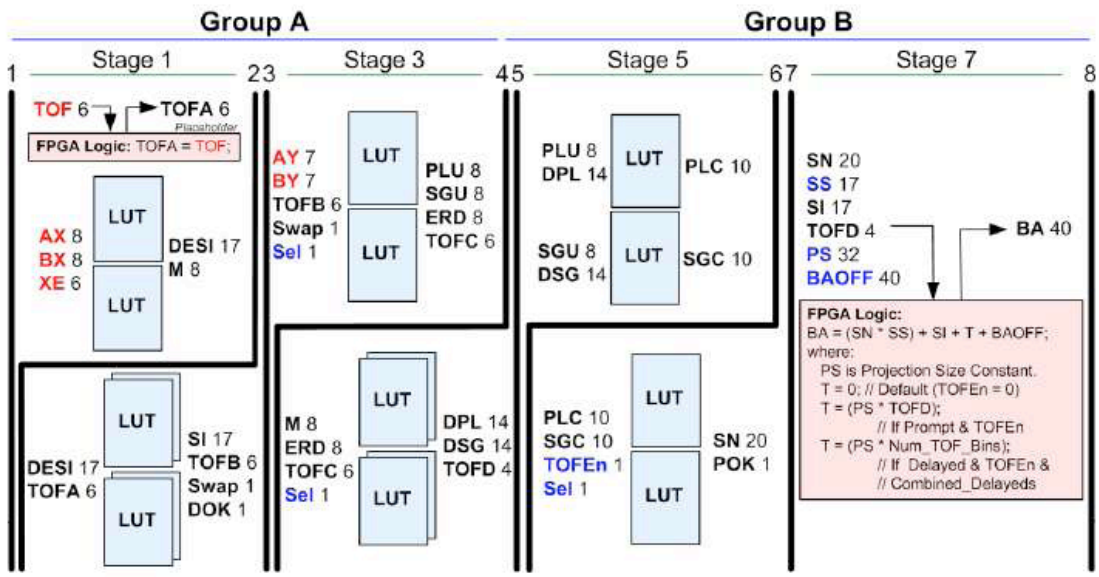


Figure 6.2 7-stage digital pipeline PDR Digital Pipeline as applied to the TOF PET System. (reproduced from [25] with permission)

6.2.1 PARTITIONING OF THE PROBLEM

When partitioning the rebinning problem the main factors to be considered results are the decomposition, assignments, and data acquisition. Preprocessing involves the reduction of load on the kernels by precomputing certain values for fast access. Decomposition involves exposing concurrency to exploit parallelism, but not so much that the cost of communication begins to outweigh the benefits of parallelism. Assignment considers the assignment of data to reduce communication between CUDA kernels, balance workload, and efficiently interface parallel threads [29]. This means reducing communication through effective task scheduling data locality, reducing synchronization costs. Efficient Data acquisition is required to process data with no losses to achieve the required

throughput. All the factors are interrelated as altering one-factor effects the others.

The rebinning problem has been decomposed into three major divisions Software, Hardware, and Output. The software has been built on C++. The hardware includes the PDR card and the GPU. The output involves the writing the output to storage media.

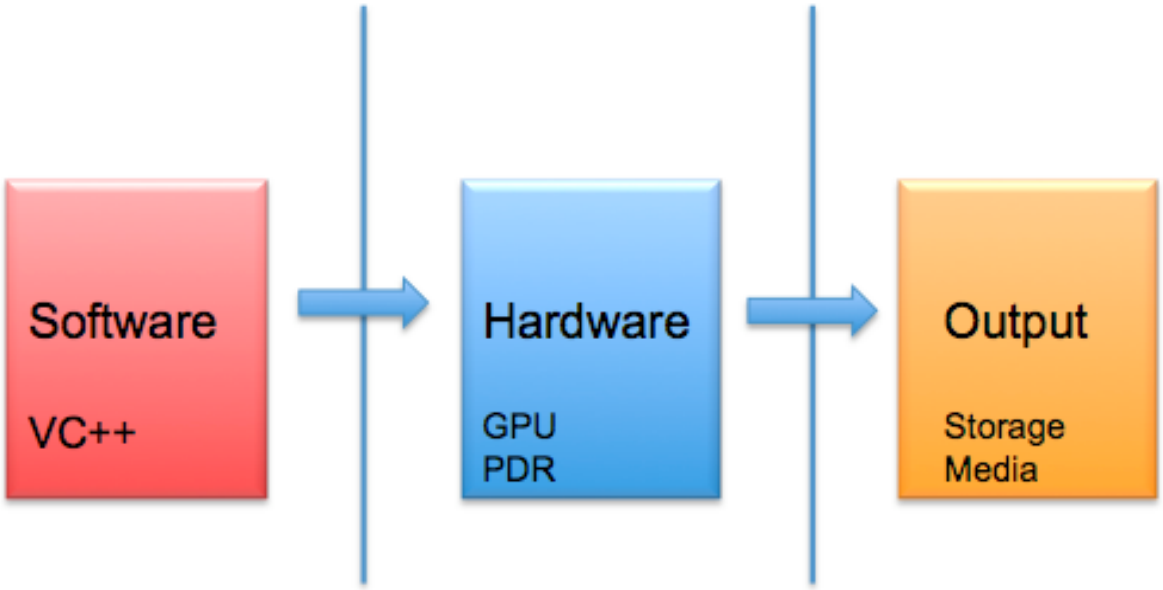


Figure 6.3 Partitioning the rebinning problem

6.3 SYSTEM

The newly developed system has a Graphics Processing Unit on a PCI express slot. This is very similar to the previous model except for the presence of the PCI based Nvidia GPU. The PDR is still used in this scenario but is limited to the collection of 64-bit PET coincidence or TOF data. The data from the PET gantry is acquired by PDR card through optical fiber cables. The output is written to a RAID 0 array of disk drives.

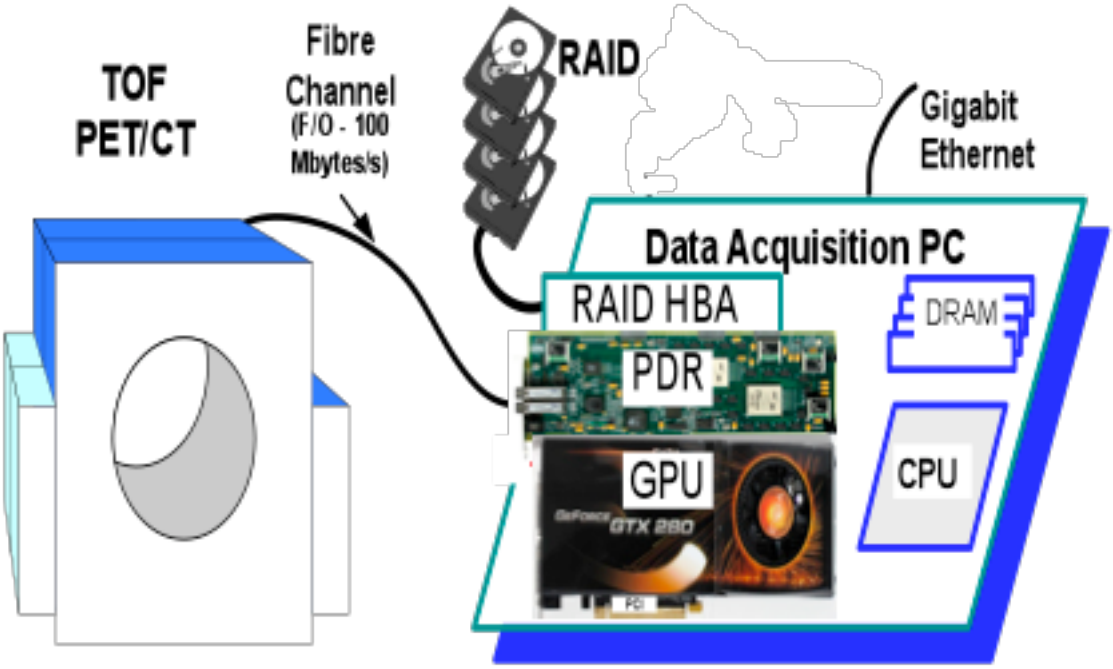


Figure 6.4 New system with PDR card along with GPU incorporated on the Data Acquisition System.

6.4 64 BIT DATA FORMAT

Based on the guideline to PETLINK [30] Table 6.1 is the 64-bit packet format for detector pair.

Table 6.1 64 Bit raw data packet

BIT #	First 32-bit word	BIT #	Second 32-bit word
0 – 7	AX0 – 7	0 – 7	BX0 – 7
8 – 15	AY0 – 7	8 – 15	BY0 – 7
16 – 18	XE0 – 2	16 – 18	XE3 – 5
19 – 21	AE0 – 2	19 – 21	BE0 – 2
22 – 24	AI0 – 2	22 – 24	BI0 – 2
25 – 27	TF0 – 2	25 – 27	TF3 – 5
28 – 29	Reserved	28 – 29	Reserved
30	Tag_64	30	Prompt
31	PS0 = 0	31	PS1 = 1

Where:

AX, BX Transaxial Head Detector Index

AY, BY Axial Head Detector Index

XE Transaxial Encoding

AE, BE Energy Window

AI, BI Depth of Interaction

TF Time of Flight

PS Packet Sync

TW Tag Word 0 - 31 (Limited to 32 bits for more effective Bit Packing.)

Tag_64 Indicates non-event (tag) 64-bit packet when set to 1;

event 64-bit packet when set to 0.

Prompt PET Prompt event word when set to 1; PET Delayed event word when 0.

6.5 WORKING MODEL

Figure 6.5 depicts a block based working model for the GPU based rebinning system.

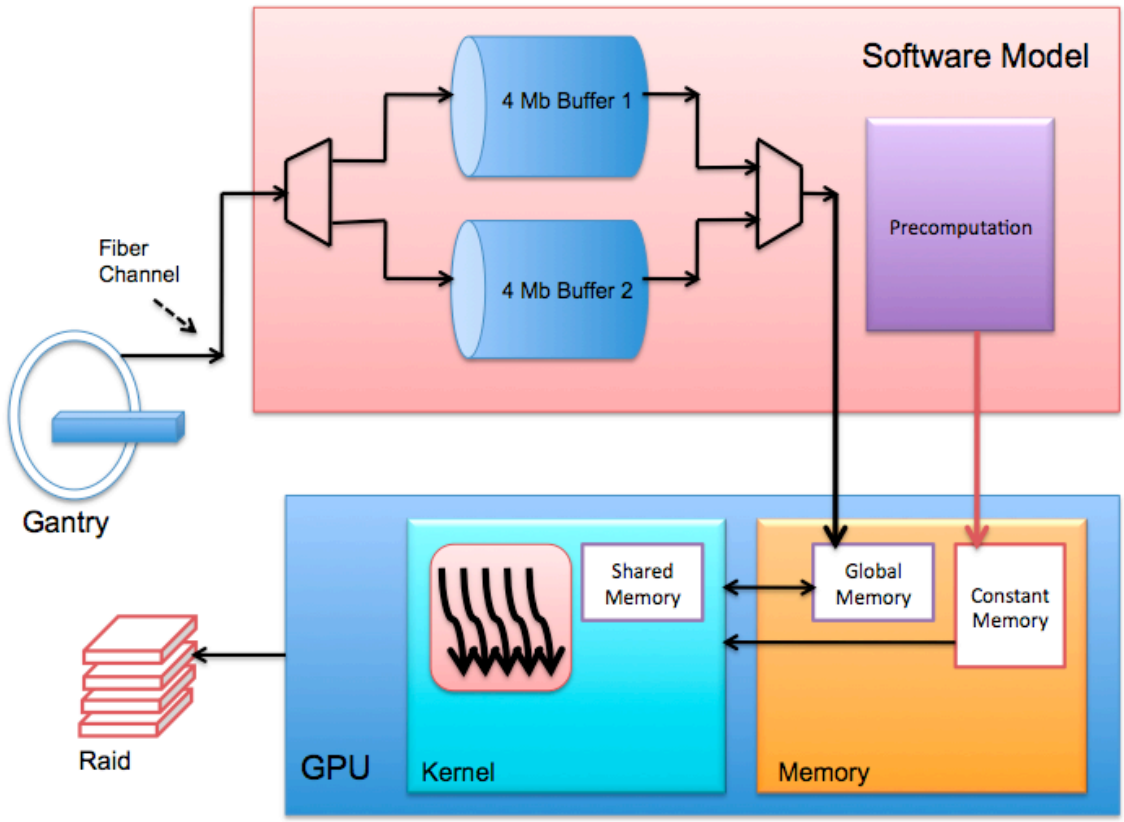


Figure 6.5 Working Model of the GPU based rebinning system

6.5.1 PRE COMPUTATION AND CONSTANT MEMORY

The first step in the process is the precomputation of certain values. The rebinning algorithms are complex and involve some time consuming for loops. Implementing these for loops proved to be too costly and therefore it was decided to precompute all the possible values for all combinations of the variables involved in the loops. Once calculated, these are loaded onto the constant memory of the GPU. Thus it is a LUT to which kernel threads have very fast access. It should be noted that the constant memory couldn't be modified once the CUDA kernel is being executed. The scope of a constant variable is all grids, meaning that all threads in all grids see the same version of a constant variable. The lifetime of a constant variable is the entire application execution. Constant variable are often used for variables that provide input values to kernel functions. Constant variables are stored in the global memory but are cached for efficient access. Accessing constant memory is extremely fast and parallel. Currently, the total size of constant variables in an application is limited at 65,536 bytes.

The 64 bit raw data from the gantry is sent through the fiber cable to the PDR card. This data is accumulated in two buffers: Buffer 1 and Buffer 2. They are designed such that once Buffer 1 is filled the data is begins to accumulate in Buffer 2. When the data is being filled in Buffer 2 the data from Buffer 1 is read and sent to the global memory of the GPU. This enables the data to be sent without losses and the input throughput to be maintained.

6.6 OPTIMIZATIONS

Performance optimization revolves around three basic strategies:

- Maximizing parallel execution.
- Optimizing memory usage to achieve maximum memory bandwidth.
- Optimizing instruction usage to achieve maximum instruction throughput.

The following are the optimization techniques employed:

Maximizing parallel execution starts with structuring the algorithm in a way that exposes as much data parallelism as possible. At points in the algorithm parallelism is lost because some threads need to synchronize in order to share data between each other, there are two cases: Either these threads belong to the same block, in which case they should use **__syncthreads()** and share data through shared memory within the same kernel call, or they belong to different blocks, in which case they must share data through global memory using two separate kernel invocations, one for writing to and one for reading from global memory. The rebinning algorithm is highly parallelizable in which the loops that required threads to share data have been removed and replaced with LUTs placed in the constant memory. Another aspect is the 64 bit raw data is packed with all the required variables and thus the thread in a block are dependent on the data from the same block and there is no data dependency on data generated from other thread blocks which makes a rebinning algorithm a favorable one to be implemented on the GPU.

Optimizing memory usage starts with minimizing data transfers with low bandwidth. That means minimizing data transfers between the host and the device, since these have

much lower bandwidth. That also means minimizing data transfers between the device and global memory by maximizing use of shared memory on the device. The bandwidth between the device and the global memory is much higher than the bandwidth between the device memory and the host memory [4]. Therefore, the algorithm was modified to send only the required variables to the kernels rather than the 64 bit raw data.

6.6.1 CONSTANT MEMORY AND PRECOMPUTATION

As discussed earlier the precomputed LUTs are stored in the fast Constant memory, which proved to be less time consuming than it would have been if the functions had been implemented on the GPU.

6.6.2 MEMORY ACCESS PATTERNS

The data from the buffers is initially stored in the global memory. The global memory is accessible to host code and can be read and write. The global memory space is not cached, so it is all the more important to follow the right access pattern to get maximum memory bandwidth, especially given how costly accesses to global memory are.

The effective bandwidth of each memory space depends significantly on the memory access pattern. Since global memory is of much higher latency and lower bandwidth than on-chip memory, global memory accesses should be minimized. A typical programming pattern is to stage data coming from device memory into shared memory.

Shared Memory is on-chip, and is much faster than the local and global memory spaces [20]. In fact, for all threads of a warp, accessing the shared memory is as fast as accessing

a register as long as there are no bank conflicts between the threads [20]. So, in order to achieve higher memory bandwidth we let each thread to :

Load data from global memory to shared memory, synchronize with all the other threads of the block so that each thread can safely read shared memory locations that were written by different threads, process the data in shared memory, synchronize again if necessary to make sure that shared memory has been updated with the results, and write the results back to global memory.

6.6.3 MEMORY COALESCING

The global memory access by all threads of a half-warp is coalesced into one or two memory transactions if it satisfies the following three conditions:

Threads must access:

Either a 32-bit word, resulting in one 64-byte memory transaction,

Or 64-bit words, resulting in one 128-byte memory transaction,

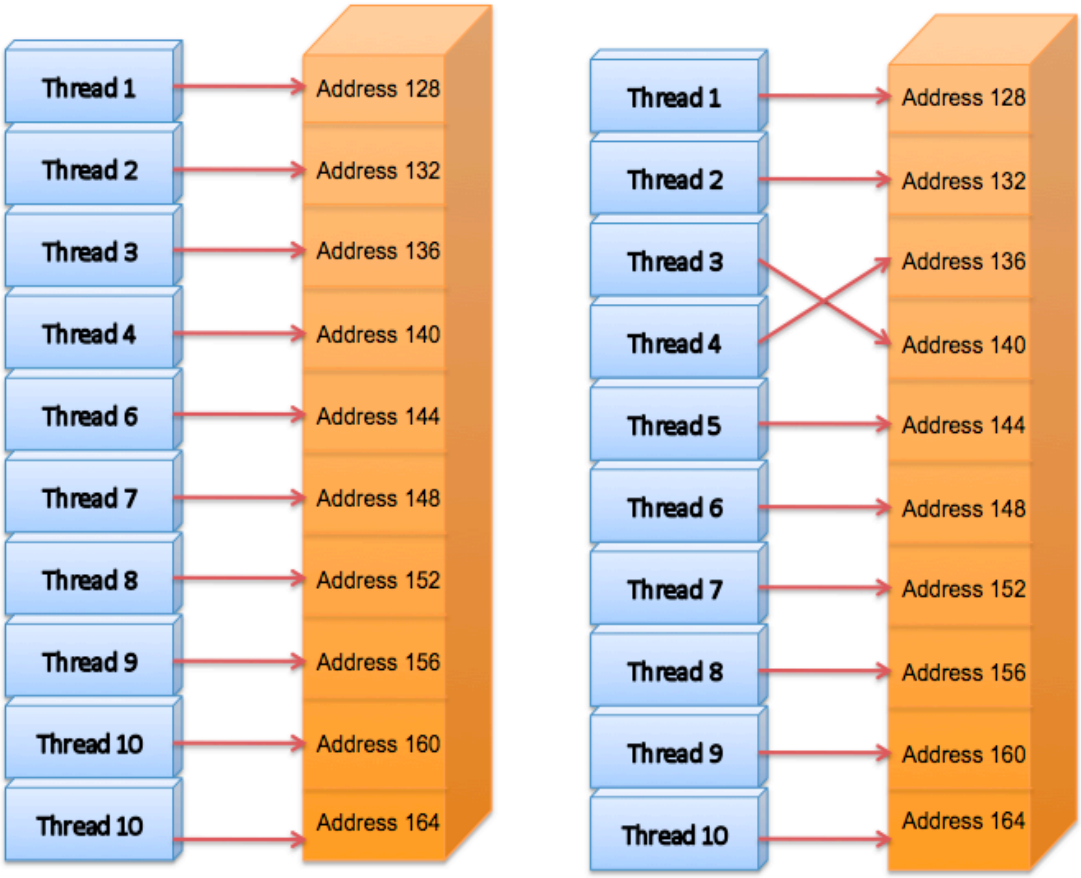
Or 128-bit words, resulting in two 128-byte memory transactions;

All 16 words must lie in the same segment of size equal to the memory transaction size (or twice the memory transaction size when accessing 128-bit words).

Threads must access the words in sequence: The *k*th thread in the half-warp must access the *k*th word.

If a half-warp does not fulfill all the requirements above, a separate memory transaction

is issued for each thread and throughput is significantly reduced [20]. Figure 6.6 is an example for coalesced and non-coalesced memory access patterns.



(a) (b)

Figure 6.6 Memory Coalescing

(a) Coalesced memory access resulting in Single memory transaction

(b) Non – Coalesced Memory access resulting 10 memory transactions

Coalesced memory access of the global memory results in a single memory transaction while an improper access results in as many memory transactions thus resulting an increase in computation time and thus affecting the performance of the algorithm on the whole.

6.6.4 ARITHMETIC INSTRUCTIONS

To process an instruction a processor has to read the instruction, execute it and write the result. An optimized instruction usage is achieved by minimizing instructions with low throughput. Also the algorithm must be such that it has a high number of arithmetic operations per memory operation. To issue one instruction, 4 clock cycles for a single-precision floating-point add, multiply, and multiply-add, integer add, bitwise operations, compare, min, max, type conversion instruction [20] and 16 clock cycles for reciprocal, reciprocal square root, logarithmic functions [20].

The rebinning algorithm involves computation of several elementary functions such as sine, cosine, and arctangent and single precision floating operations. Varied usages of instructions have been used in the rebinning to see that the final result computed is not affected. For example, __sinf(x), __cosf(x), __expf(x) take 32 clock cycles. sinf(x), cosf(x), tanf(x), sincosf(x) are much more accurate and expensive and even slower. Though the final result base address is a floating point number, only the exponent of this result is taken as the base address. This enables us to experiment with the floating-point operations and thus enables us to use high throughput instructions while compromising

on accuracy.

A few compiler flags have also been used. The programmer can control loop unrolling using the **#pragma unroll.** It must be placed immediately before the loop and only applies to that loop. A number that specifies how many times the loop must be unrolled optionally follows it. If no number is specified after **#pragma unroll**, the loop is completely unrolled.

6.7 ASYNCHRONOUS APPLICATION PROGRAMMING INTERFACE

CUDA has default Application Programming Interface (API) and an Asynchronous API. In the default API the memory transfer from the host to device (H2D) and from device to host (D2H) block the CPU thread. The host CPU thread is blocked when the kernel on the GPU is called and running. The implementation of the rebinning algorithm was done with the default API but it was also done with the Asynchronous API, which enables asynchronous H2D and from D2H data transfer. It also enables the kernel to be executed concurrently while the host code can be executed.

Events are inserted into a stream of CUDA calls. Since CUDA stream calls are asynchronous, the CPU can continue with computations while the GPU is executing (including DMA memcopies between the host and device). The CPU can query the CUDA event status to determine whether the GPU has completed its tasks. In order to

facilitate concurrent execution between host and device, some runtime functions are asynchronous control is returned to the application before the device has completed the requested task. **CudaMemcpyAsync ()** is used to do the memory transfer asynchronously. The runtime also provides a way to closely monitor the device's progress, as well as perform accurate timing, by letting the application asynchronously record *events* at any point in the program and query when these events are actually recorded. An event is recorded when all tasks – or optionally, all operations in a given stream – preceding the event have completed. Figure 6.7 is a sample code for Asynchronous implementation of a kernel

```
98
99      // asynchronously issue work to the GPU
100     CUT_SAFE_CALL( cutStartTimer(timer) );
101
102         cudaEventRecord(start, 0);
103         cudaMemcpyAsync(device_a, host_a, nbytes, cudaMemcpyHostToDevice, 0);
104
105         // Call TOF Kernel for BA Calculation
106         PET_TOF<<<blocks, threads, 0, 0>>>(device_a, result);
107
108         cudaMemcpyAsync(host_result, result, nbytes, cudaMemcpyDeviceToHost, 0);
109         cudaEventRecord(stop, 0);
110
111     CUT_SAFE_CALL( cutStopTimer(timer) );
112
113     // have CPU do some work while waiting for stage 1 to finish
114
115     while( cudaEventQuery(stop) == cudaErrorNotReady )
116     {
117
118     // code to be executed
119
120     }
```

Figure 6.7 Asynchronous implementation of kernel in CUDA

6.8 SYSTEM SPECIFICATIONS

CPU 1 is a lower end PC with Intel Dual Core Xeon 5140 2.33 GHz processor on the board with 3 GB of RAM. The processor has an L2 Cache memory of 4 MB. It does not support the Intel Hyper Threading technology

CPU 2 is a higher end system that has two physical Nehalem-based dual quad core Intel Xeon 5530 2.4 GHz processors each having 12 GB RAM each. Each processor has an 8 MB cache. The processor support Hyper Threading Technology.

The Graphic Processing Units used for benchmarking are 8800 GTX and 280 GT

Table 6.2 GPU Specifications

	GeForce 8800 GTX	**GeForce 280 GT**
Stream Processors	128	240
Core Clock (MHz)	575	600
Memory Clock (MHz)	900	1107
Memory Amount	768 MB	1024 MB
Memory Interface	384-bit	512-bit
Memory Bandwidth (GB/sec)	86.4	141
Texture Fill Rate (billion/sec)	36.8	48.2

7 RESULTS

The following chapters summarizes the results of the GPU-based implementation for improved online rebinning performance in clinical 3-D PET in comparison to the existing PDR rebinning hardware and stand-alone CPU. The following sections will discuss and interpret the results and difficulties encountered

7.1 COMPARISON OF RESULTS

Figure 7.1 plots the time-based comparison of the Coincident event rebinning for different event sizes with respect to time. We see that that the 280 GT on CPU 1 takes more time than the same GPU on CPU 2. This is because CPU 2 being an multi core machine with advanced Intel system with higher RAM and the board supports PCIE 2 6 GBps transfer rate.

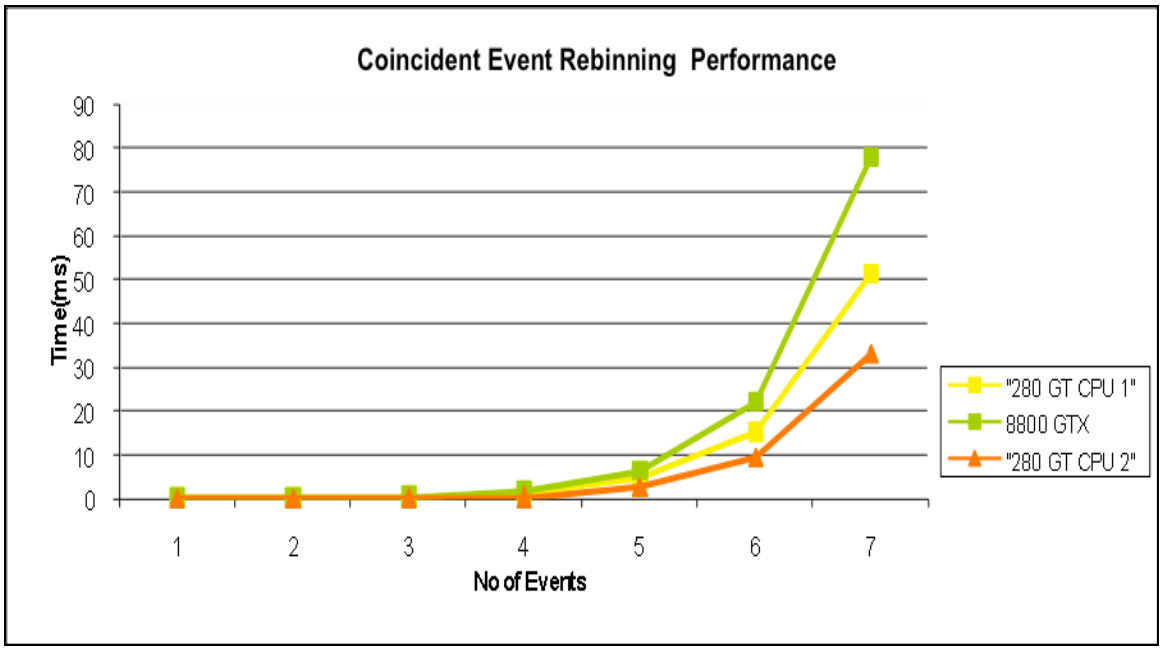


Figure 7.1 Rebinning time performance

Figure 7.2 represents the performance in terms of the throughput that can be supported. The existing dedicated rebinning hardware based on the PDR card can support an input throughput of only 8 million events per second. We also see that the GPU 8800 GTX that has low memory and lower bandwidth compared to GPU 280 GT can support event rate of 52 million events per second. The GPU 280 GT supports higher bandwidth and also has a higher memory interface fares much better with 82 million events per second. The same GPU on a CPU that supports PCIE 2 interface shows increase in performance and thus supports 136 million events per second.

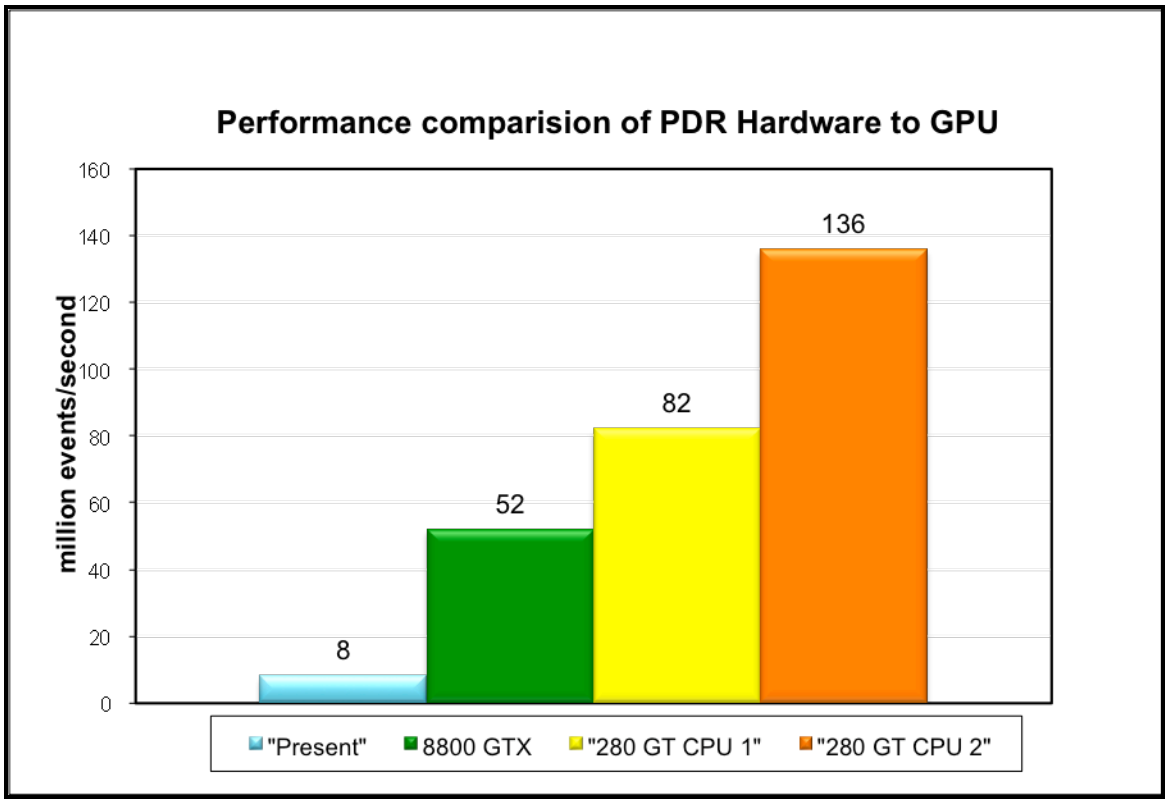


Figure 7.2 Throughput performance comparision

Figure 7.3 and Figure 7.4 plot the difference between the transfer and execution times for the GPU 280 GT on CPu 1 and CPU 2. Here we note an interesting aspect that the execution times for the kernel on both CPU 1 and CPU 2 are almost equal, but the transfer times differ by a considerable amount. This can be attribute to the fact that the CPU 1 supports only PCIe 1 which is 3 GBps while CPU 2 offers a higher bandwidth of 6 GBps.

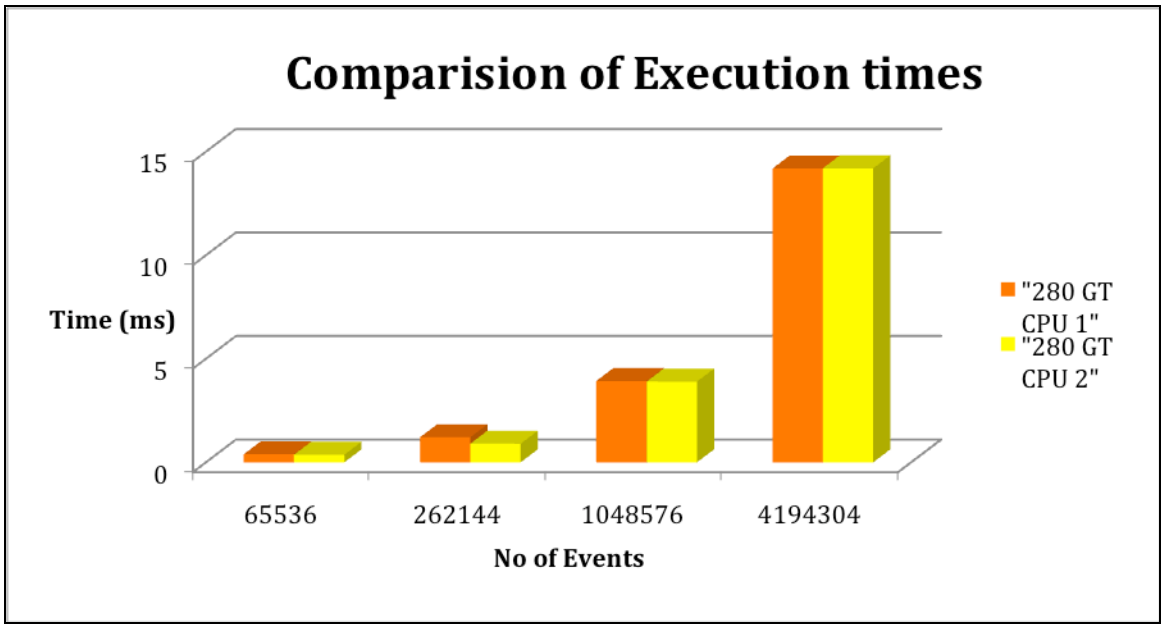


Figure 7.3 Comparision of Execution Time

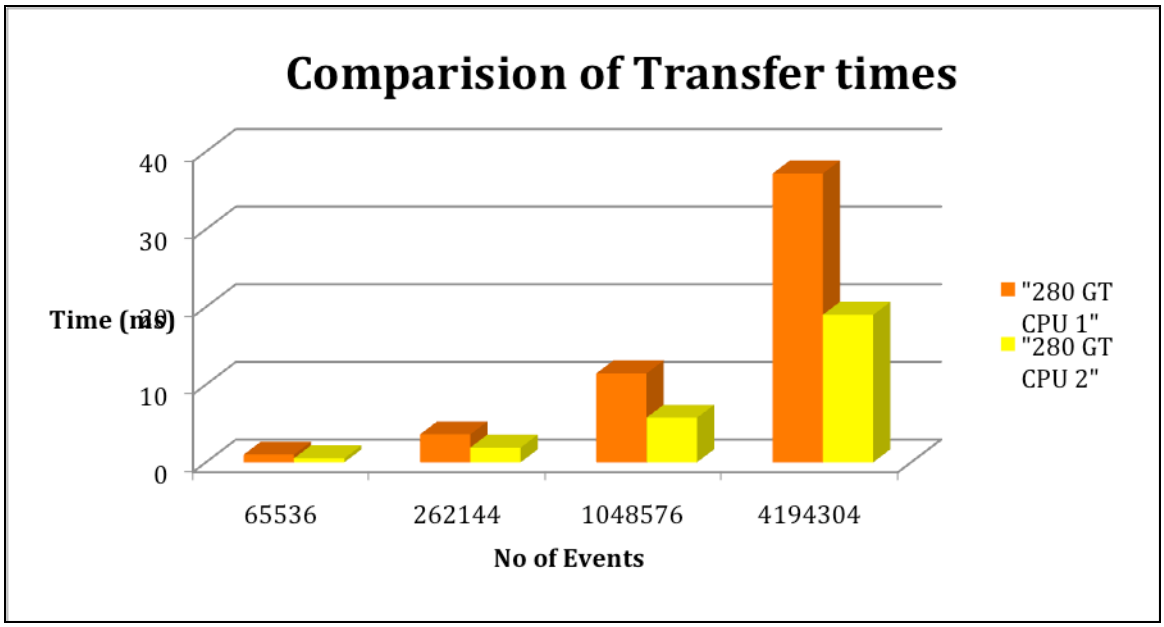


Figure 7.4 Comparision of Transfer Time

Figure 7.5 gives the comparision of the online time of flight (TOF) rebinning. The TOF has been implemented in the default API and also using the Asynchronous API. The performance of TOF rebinning on 280 GT on CPU 1 was low compared to coincident event rebinning because of the presence of the large number of floating point calculations in the TOF rebinning. This implementation could support 66 million events per second. The same algorithm implemented in the asynchronous API on CPU 1 resulted in a better performance because the data ttransfer to the H2D and D2H has been overlapped with the execution of the kernel. This led to increase in performance and could support the input rate of 90 milion events per second. Table 7.1 gives the numerical values for the Coincident event rebinning performance.

Table 7.1 Performance of Coincident event rebinning

No. Of events	CPU time (ms)	GPU 8800 GTX (ms)	GPU 280 GT	
			CPU 1	CPU 2
65536	191.01	1.78	1.39	0.89
262144	792.11	6.37	4.81	2.74
1048576	3116.27	22.12	15.38	9.6
4194304	12486.03	77.92	51.38	33.1

Performance based on Event Rate

System	Present	GPU 8800 GTX	CPU 1	CPU 2
Million events/sec	8	52	82	136

Comparison of Transfer and Execution times

No. of events	GPU 280 GT - CPU 1		GPU 280 GT -CPU 2	
	Execution Time (ms)	Transfer Time (ms)	Execution Time (ms)	Transfer Time (ms)
65536	0.39	1	0.37	0.52
262144	1.21	3.6	0.9	1.9
1048576	3.93	11.45	3.9	5.75
4194304	14.19	37.19	14.2	19

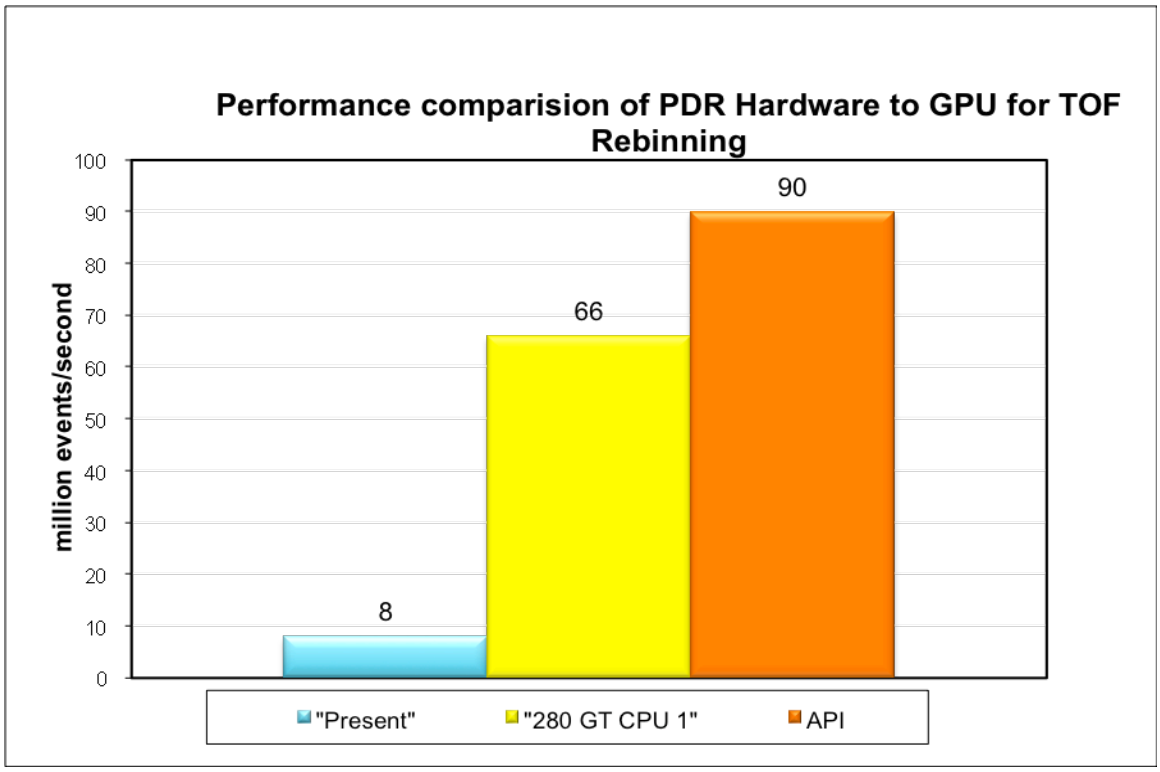


Figure 7.5 TOF Rebinning Performance

Table 7.2 TOF rebinning performance values.

No. Of events	CPU time (ms)	API	GPU 280 GT CPU 1	GPU 280 GT CPU 2
1048576	3116.27	12.1	15.8	10

Performance comparison in Events per second

System	Existing PDR Hardware	API	CPU 1	CPU 2
Million events/sec	8	90	66	104

Comparison of Transfer and Execution Times

No. Of events	GPU 280 GT - CPU 1		GPU 280 GT - CPU 2	
	Execution Time (ms)	Transfer Time (ms)	Execution Time (ms)	Transfer Time (ms)
1048576	1.8	14	1.8	8

7.2 CUDA VISUAL PROFILER

CUDA provides a visual profiler to get a statistics for an algorithm implemented on GPU. The following are the screenshots of the statistics obtained from the profiler for the rebinning algorithm per 1048576 coincident events on CPU 1 with 280 GT.

The GPU Time Height Plot is a bar diagram in which the height of each bar is proportional to the GPU time for a method. Figure 7.6 gives a height plot for the program with the time in microseconds on the Y-axis. For the rebinning algorithm we see that the transaxial rebinning occupies a major part of the runtime. We also see that the memcopy functions in blue, which includes data transfer from H2D and D2H, also are consuming the largest portion of time in the total runtime.

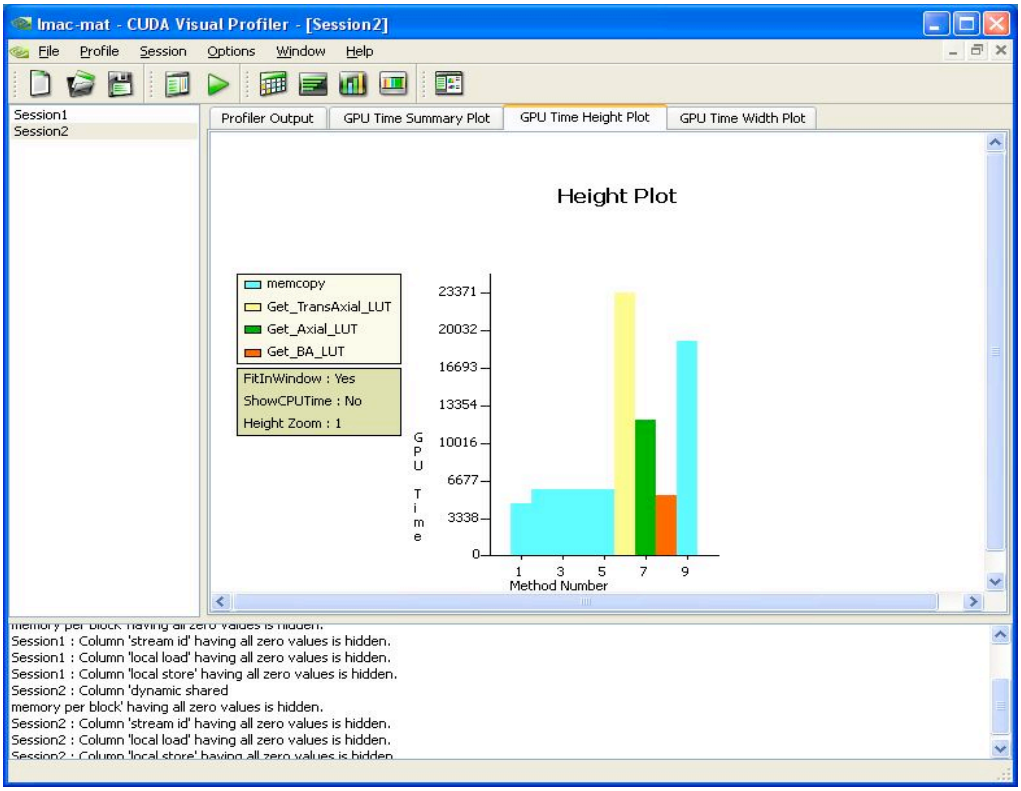


Figure 7.6 Height Plot using CUDA visual profiler

Figure 7.7 gives a percentage view of the runtime occupancy for the kernels and data transfer. We see that the 56 % of the time is being used for data transfer only. This gives the user an exact idea about the time occupancy of different kernels and the data transfers so that one can concentrate on specific kernels for optimizing them.

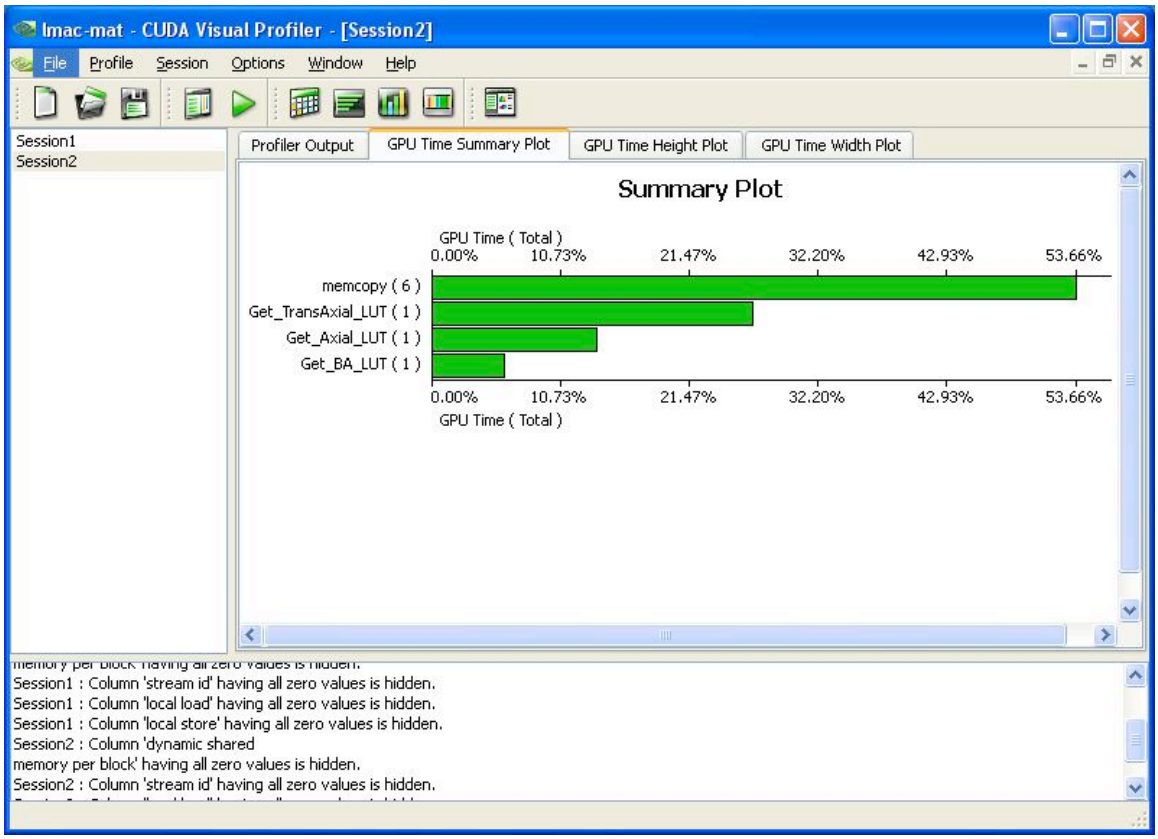


Figure 7.7 Percentage plot using CUDA visual profiler

The following Figures 7.8 and 7.9 give the transfer times, occupancy ratios, grid sizes etc and also some other important data. In Figure 7.8 we see the exact time stamps for the kernels executed and also the data transfers. The time taken for each data transfer and for the execution of each kernel is given. The 6th memcopy function takes the maximum time, as it is the transfer of resultant base address from D2H.

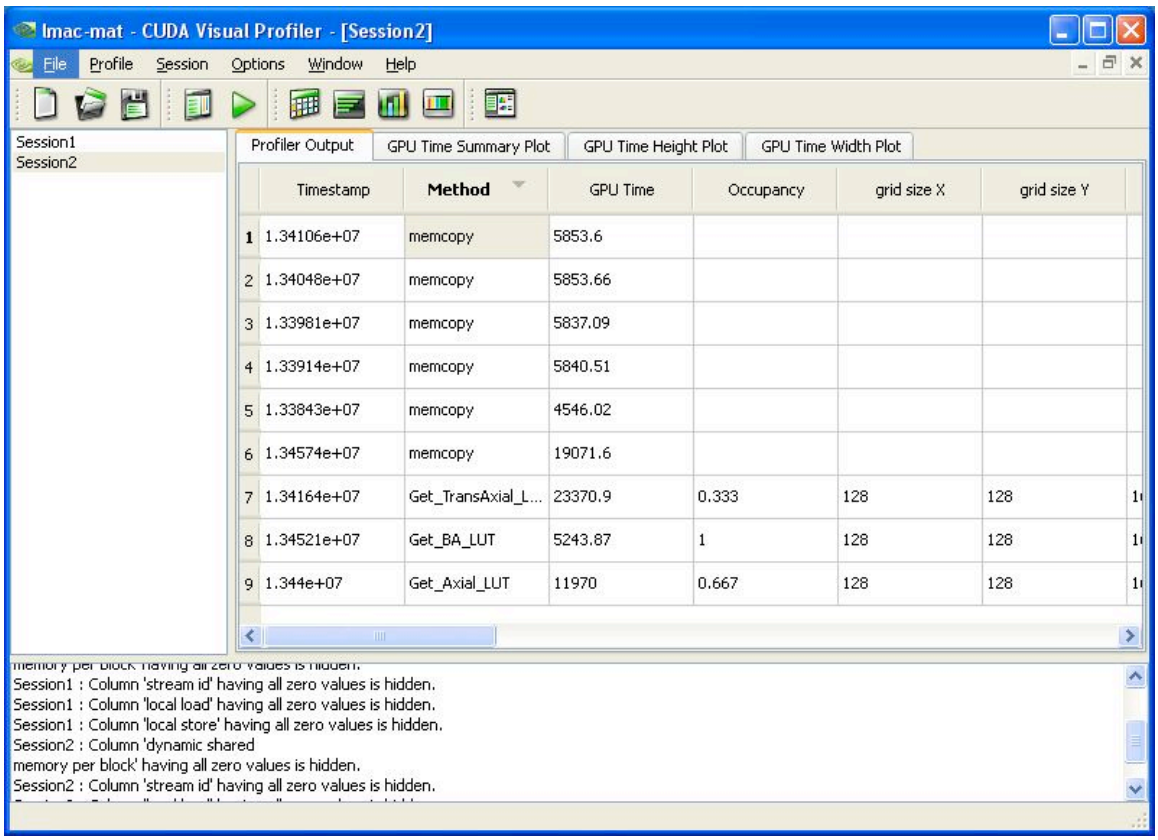
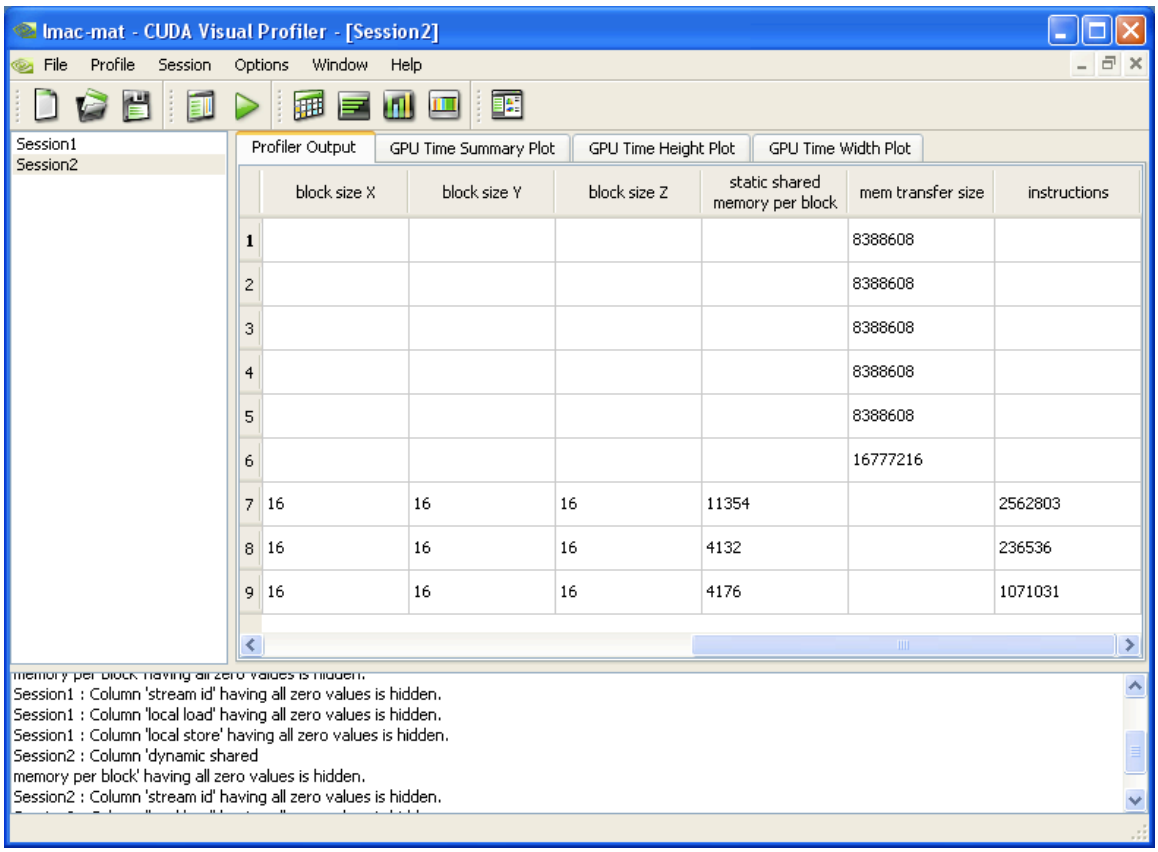


Figure 7.8 Kernel runtime stats from CUDA visual profiler

Figure 7.9 gives information about the number of instructions executed per kernel. It also gives the specific size of shared memory in KB used by each block during the kernel execution. We see that the 6th memory transfer is the largest which is the single precision base address calculated as output.



	block size X	block size Y	block size Z	static shared memory per block	mem transfer size	instructions
1					8388608	
2					8388608	
3					8388608	
4					8388608	
5					8388608	
6					16777216	
7	16	16	16	11354		2562803
8	16	16	16	4132		236536
9	16	16	16	4176		1071031

Figure 7.9 Instruction Output from CUDA visual profiler

7.3 LIMITATIONS

The limitations to the implementation have been the bandwidth support of the GPUs for the data transfer. This is very low and led to the memcopy () occupying a major part of the runtime than the actual time taken by the GPU to execute the kernel. Another bottleneck has been the data write speed to the storage media. Each Buffer written was taking ~70 ms. This is much higher than the 50 ms taken by the GPU.

8 CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

In evaluating the entire design and results a lot has been learned regarding the GPU architecture, CUDA programming model, and parallel computing. This thesis has successfully presented the GPU-based implementation of the rebinning algorithms in and has been tested on hardware in real time. The GPUs have proved to deliver very high performance when used for online rebinning. It is known that using an advanced multi core system which supports PCIe 2 leads to the reduction of data transfer times and leads to a much better performance. At their peak performance the GPUs are 17x faster than the dedicated rebinning hardware. This proves that it is worth continuing to explore research areas appropriate for GPUs where traditionally FPGA are being used. A faster rebinning leads to a faster reconstruction of images and leads to a high patient throughput in diagnostic fields. The NVIDIA CUDA architecture for GPU computing is a good solution for a wide circle of parallel tasks. CUDA works with many NVIDIA processors. It improves the GPU programming model, making it much simpler and adding a lot of features, such as shared memory and thread syncing. CPUs are developing rather slowly, they are not capable of such performance leaps. In fact, one can now get an inexpensive personal supercomputer, sometimes even without investing extra money, as graphics cards from NVIDIA are widely spread.

The existing PDR rebinning hardware is based on FPGAs. It can accept the imput rate of only 8 milion events per second. Developing a system with FPGAs is expensive, time consuming and requires skilled manpower. Whereas with the introduction of CUDA it has become easier to use the GPU as a parallel computing device.

8.2 FUTURE WORK

As the main bottleneck has proved to be the data transfer from H2D and D2H, it is worth exploring the possibility of accessing the GPU Global Memory and writing data to it directly. Right now the only possible way of writing data and reading from the GPUs global memory is through CUDA.

Secondly the results write speeds are very low when using traditional hard disk drives (HDD) is very low. This can easily mitigate the advantage of the speedups obtained by using the GPUs. An option would be to explore the new Solid State Drives (SSD) which use solid state memory to store data and offer much higher write and read speeds compared to HDDs.

It is also worthy investing time in trying to increase the data input rate. The present input rate to the system saturates at 13 million events/sec while the GPUs are ready to accept up to 100 million events/sec.

BIBLIOGRAPHY

1. Badawi R.D, *Aspects of optimization and quantification in three-dimensional Positron Emission Tomography*. Dissertation for the degree of Ph.D., University of London, UK, 1998
2. Ter-Pogossian, M.E. Phelps, E.J. Hoffman, *A positron-emission transaxial tomograph for nuclear imaging (PET)*. Radiology 1975, v. 114, no. 1, pp. 89-98
3. K Hamacher, HH Coenen and G Stocklin, Efficient *stereo specific synthesis of no-carrier-added 2-[^{18}F] fluoro-2-deoxy-D-glucose using amino polyether supported nucleophilic substitution*. J Nucl Med 27(2), 1986. 235-238
4. RC Marshall, JH Tillisch, ME Phelps, S-C Huang, R Carson, E Henze and HR Schelbert, *Identification and differentiation of resting myocardial ischemia and infarction in man with positron computed tomography, ^{18}F-labelled fluorodeoxyglucose and N-13 ammonia.* Circulation 67(4), 1983 766-778
5. LG Strauss and PS Conti, *The Application of PET in Clinical Oncology.* J Nucl Med 1991 32(4):623-648
6. P Hellman, H Ahlström, M Bergström, A Sundin, B Långström, Göran Weterberg and C Julin, *Positron emission tomography with ^{11}C-methionine in hyperparathyroidism Surgery.* 1998. 116(6), 974-981
7. WG Kuhle, G Porenta, S-C Huang, D Buxton, SS Gambhir, H Hansen, ME Phelps and HR Schelbert, *Quantification of regional myocardial blood flow using ^{13}N ammonia and reoriented dynamic positron emission tomographic imaging*. Circulation 1992 86(3), 1004-1017

8. I Kanno, AA Lammertsma, JD Heather, JM Gibbs, CG Rhodes, JC Clark and T Jones, *Measurements of cerebral blood flow using bolus inhalation of $C15O_2$ and positron emission tomography: description of the method and comparison with $C15O_2$ continuous inhalation method*. J. Cereb. Blood Flow Metabol. 1984 4(2), 224-234
9. CS Brock, SR Meikle, P Price, *Does fluorine-18 fluorodeoxyglucose metabolic imaging of tumors benefit oncology?*. Eur J Nucl Med 1997 24:691-705
10. RA Hawkins, Y Choi, S-C Huang, CK Hoh, M Dahlbom, C Schiepers, N Satyamurthy, JR Barrio and ME Phelps, *Evaluation of the Skeletal Kinetics of Fluorine-18-Fluoride Ion with PET*, J Nucl Med 33(5), 633-642 1992
11. Moyers J.C. Jr. "*High Performance Detector Electronics System for Positron Emission Tomography*". Dissertation for the degree of M. S., University of Tennessee, USA, 1990
12. Rankowitz, et al: *Circular Ring Transverse Axial Positron Camera for 3-dimensional Reconstruction of Radionuclides Distribution*, IEEE Trans. Nucl. Sci. NS-23(1):613-622, 1976.
13. Kuhl, D.E., Edwards, R. Q. *Image Separation Radioisotope Scanning, Radiology*, 80:653-661, 1963
14. Ido T, C-N. Wan, V. Casella, J.S. Fowler, A.P. Wolf, M. Reivich, and D.E. Kuhl, *Labeled 2- deoxy-D-glucose analogs. 18F-labeled 2-deoxy-2-fluoro-D- glucose, 2-deoxy-2-fluoro-D-mannose and C-14-2-deoxy- 2-fluoro-D-glucose*, The Journal of Labeled Compounds and Radiopharmaceuticals 1978; 14:175-182.
15. A. Alavi, Reivich, M., D. Kuhl, A. Wolf, J. Greenberg, M.Phelps, T. Ido, V.

Casella, J. Fowler, E. Hoff-man, P. Som, and L. Sokoloff, *The[18F f]ourodeoxyglucose method for the measurement of local cerebral glucose utilization in man*, Circular Research 1979; 44:127-137.

16. Gordon L. Brownell, *A HISTORY OF POSITRON IMAGING*, Physics Research Laboratory, Massachusetts General Hospital, Massachusetts Institute of Technology, Division of Radiological Sciences, Massachusetts Institute of Technology. 1999
17. E. Lindholm et al., *NVIDIA Tesla: A Unified Graphics and Computing Architecture,* IEEE Micro, vol. 28, no. 2, Mar./Apr. 2008, pp. 39-55.
18. Michael Garland, Scott Le Grand, John Nickolls, et al. "*Parallel computing experiences with CUDA" IEEE Micro 2008; Volume 28:13-27*
19. David Luebke, *CUDA: Scalable parallel programming for high-performance scientific computing.* NVIDIA Corporation. ISBI, 2008 pages 836–838.
20. NVIDIA. 2007. CUDA Programming Guide 2.0
21. M.Silberstein, A.Schuster, D.Geiger, A.Patney, J.D.Owens, *Efficient sum-product computations on GPUs using software-managed cache*. Proceedings of the 22nd ACM International Conference on Supercomputing. 2008. 309-318.
22. Svetlin A, Manavski, and Giorgio Valle1, *CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment.* BMC Bioinformatics 2008, 9(Suppl 2):S10

23. Eric Breeding et al. *"Final Specification, Hardware Design Description, PETLINK DMA REBINNER"*, (CPS INNOVATIONS), Siemens Medical solutions Internal Document. Used with permission

24. W. F. Jones, E. Breeding, B. Castleberry, J. Reed, *Nearest-Neighbor Rebinning in Clinical PET: Fast and Accurate on-Line 3-D LOR-to-Bin Mapping on the HRRT with the New PDR Card.* Nuclear Science Symposium Conference Record, 2004 IEEE

25. W. F. Jones, E. Breeding, M. Conti, F. Kehren, M. Casey, *"On- Line Time-of-Flight Mashing: the PDR Card Applied…,"* IEEE MIC Conference Record, San Diego, 2006.

26. Michel Defrise, P. E. Kinahan, D. W. Townsend, *Exact and Approximate Rebinning Algorithms for 3-D PET Data*, IEEE transactions on Medical Imaging, Vol 16, NO. 2, 1997

27. W. F. Jones, E. Breeding, M. Conti, F. Kehren, M. Casey, *On- Line Time-of-Flight Mashing: the PDR Card Applied…,,* IEEE MIC Conference Record, San Diego, 2006.

28. M. Pharr, *GPU Gems 2*. Addison-Wesley, 2005.

29. Kirk Andrew Baugher, *Sparse Matrix Sparse Vector Multiplication using Parallel and Reconfigurable Computing.* A Thesis Presented for the Master of Science Degree, The University of Tennessee 2004.

30. Guideline to the PETLINK Proposal, Siemens Medical Solutions Internal document. Used with permission.

31. Gropp, William; Lusk, Ewing; Skjellum, Anthony. (1994) Using MPI: portable parallel programming with the message-passing interface. MIT Press In Scientific And Engineering Computation Series, Cambridge, MA, USA. 307 pp. ISBN 0-262-57104-8

VITA

Dilip Reddy Patlolla was born on 26th October 1984 in Hyderabad, India. He spent his childhood and adolescent life in Hyderabad. He began attending college at DVR College of Engineering and Technology affiliated to Jawaharlal Nehru Technological University where he graduated with distinction in 2006. Immediately following the completion of his undergraduate degree, Dilip started his Masters degree at University of Tennessee, Knoxville. During his Masters he interned with Siemens Medical Solutions for one year. In 2009 he graduated with Master of Science in Computer engineering

Dilip will be starting his career as an engineer with Siemens Medical Solutions Inc, USA.