Accelerations of Structural and Functional Brain Connectivity Using Heterogeneous Computing

by

Shenjun Zhong, BMech, MInfoTech

Thesis

Submitted by Shenjun Zhong

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Dr. David Squire

Associate Supervisor: Dr. David G. Barnes, Dr. Parnesh Raniga, Dr. Govinda Poudel, Dr. Richard Beare

Clayton School of Information Technology

Monash University

November, 2016

Accelerations of Structural and Functional Brain Connectivity Using Heterogeneous Computing

Shenjun Zhong, BMech, MInfoTech
[redacted]
Monash University, 2016

Supervisor: Dr. David Squire
[redacted]
Associate Supervisor: Dr. David G. Barnes, Dr. Parnesh Raniga, Dr. Govinda Poudel, Dr. Richard Beare
[redacted]

Abstract

In this thesis, the main aims are to accelerate algorithms in diffusion tractography and functional MRI connectivity analysis, by mapping them on parallel architectures. Diffusion Tractography is an algorithm for studying micro-structure of brain white matter (WM), and functional MRI is a neuroimaging procedure to explore the time series of brain activities. Both of these algorithms are widely applied in neuroscience researches.

Diffusion-weighted (DW) magnetic resonance (MR) images can be processed to yield orientation information on the underlying anisotropic distribution of elongated fibers, such as in the white matter of the brain. As MR field strengths increase and scan protocols are improved, the spatial and angular resolution that can be achieved have reached the point where traditional diffusion tensor imaging (DTI) methods are being replaced in favour of non-tensor methods, so as to allow for multiple fiber directions per image voxel. Once the fiber distribution is known, the generation of whole streamlines - stochastic representations of white matter fiber bundles - while computationally dense, is intrinsically parallel and is eminently suited to acceleration with the contemporary graphical processing unit (GPU). Here, we report the design, implementation, validation and performance analysis of two different parallel mappings of the standard probabilistic tracking algorithm that is applied to DW MR images represented in spherical harmonics. We achieve a 10x speedup on a commodity GPU, compared to the standard multi-core CPU implementation, while recovering the expected distribution of the streamlines. Our parallel implementation scales well across different hardware and problem sizes. The best rate achieved is one million streamlines computed in less than 20 seconds.

The work on the accelerated sliding-window-based spatial ICA components tracking on functional MRI time-series data is also main part of our project, which is a more flexible approach of studying brain functional dynamics than traditional Region-of-Interests (ROI)-based analysis. In order to perform real-time ICA processes, we are investigating into the parallel mapping and implementations of the FastICA algorithm on GPU(s) to

achieve less than 2 seconds (i.e. the sampling speed of the current scanning for fMRI sequences). This may open new possibilities of performing real-time neurofeedback studies and intraoperative image-guided neurosurgery.

Accelerations of Structural and Functional Brain Connectivity Using Heterogeneous Computing

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

[redacted]

Shenjun Zhong
November 1, 2016

Acknowledgments

I would like to thank everyone who helped to make this possible. It has been an incredible journey of self-discovery, and I love every last one of you who are always believing in me.

I hardly had a good start of my PhD candidature, and unfortunately changed my entire supervision team and project at the end of the first year. I were very apprecited that Assoc. Prof. David Barnes, Dr. David Squire, Dr. Parnesh Raniga, Dr. Govinda Poudel and Dr. Richard Beare assisted me to overcome every bit of difficulty on the journey. Without them, there is no chance that I can reach this far. Finally, I am proud of being your student in my life time.

My sincere thanks also go to my parents, Faquan Zhong and Guixuan Wang who always support me, although we ran into a bit relationship crisis for a short period of time. Regardlessly, they are always beside me and love me. And, I want to say sorry to my ex-girl friend, Chenping Lv. She has never been abandomed our love and memory about all the happy time we spent together. If time goes back, I would like to walk through the journey of my PhD again, but with you aside me.

Speical thanks to all my colleagues, Phil Ward (and his lovely brother Roy), Dror Cohen, Qizhu Wu, Chao Suo, Nitin Mahadeo and the director of the MBI, Prof. Gary Egen for their supports and enlightenments; and all my sweet friends, Xiaoqyu, Difei, Ninghui, Xiaowei, Mu, Min, Gang, Hua, Shenglei and Limin.

Eventually, I am here and ready for the next journey,

Shenjun Zhong

Monash University
November 2016

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The brain is the most complicated and least understood organic system in the human body. Scientists have spent centuries studying the secret behind the brain, particularly the human brain, exploring the relationships between the brain properties (both structures and functions) and the (both conscious and subconscious) neuron activities that relate to memory, reasoning, judgement and other neuro-behaviours. Curiosity about the consciousness, memory mechanism and other aspects of the brain has driven many generations of scientists from different backgrounds (e.g. biology, neurology, physics, medicine and psychology) to work in this area, which is now known as the field of "neuroscience" in general. The big questions like: "what is consciousness", "how does memory work", and "how do we learn" are still largely unanswered.

Broadly, neuroscience is an interdisciplinary field that is linked to many areas. It is highly related to mental illness (in psychiatry and neurology), such as studies about Autism (Cook, 1990), Alzheimer's disease (McGeer et al., 1996), epilepsy (Mischel et al., 1995), and Parkinson's disease (Hughes et al., 1992). Mental illness and disease can affect the brain at any stage of life, and quality of life for sufferers can be substantially compromised. In psychology, it often associates with questions about cognitive functions, e.g. vision (Rolls and Deco, 2002), auditory (Schnupp et al., 2011), attention (Treisman and Gelade, 1980) and memory (LaBar and Cabeza, 2006). And, more neuroscience studies are conducted with genetic tests, in order to find the genetic origins of particular mental disorders and cognitive diversity (Hyman, 2002). Furthermore, in the bionic perspectives, investigating and understanding the (human) brain can also help development of artificial intelligence (AI) algorithms (Goldman, 2006) and robotics (Mataric, 2002), in which, deep learning networks (DL) are one of those obvious examples (Erhan et al., 2010; Bengio, 2009). A major technique in neuroscience is neuroimaging, which aims to develop imaging techniques (both hardware and software) to create more advanced way to studying both the structures and functions of the brain (Hashemi et al., 2012; Lyon and Rumsey, 1996). Last but not least, attempts of applying research findings into real-life clinical environments are also made by many scientists and clinicians, such as image-guided neurosurgery (Daga et al., 2011; Galloway et al., 1992; Lumenta et al., 2010).

Methods for studying the brain are evolving. Early simulation-based studies with electric currents since the early 19th century have established the general methodology of neuroscience. However, most progress in the study of human brain made since then rely on patients with neurological disorders, until in the last one to two decades, non-invasive brain imaging techniques, e.g. computed tomography (CT) (Brenner and Hall, 2007; Hsieh, 2009) and magnetic resonance imaging (MRI) (Basser and Jones, 2002; Hashemi et al., 2012), enable studies of healthy/treatment human subjects.

The advent of non-invasive medical imaging has provided scientists with greater possibilities to explore the human brain. Among different imaging techniques, MRI has undoubtedly been attractive for neuroscientists, since it can offer various image contrasts, high spatial and temporal resolution, and importantly, it does not utilize ionising radiation. MRI has been used in many aspects of brain imaging studies, particularly brain white matter research using diffusion MRI (dMRI) (Basser and Jones, 2002; Le Bihan, 2003) and functional studies using functional MRI (fMRI) (Biswal et al., 1995; Ogawa et al., 1990). The modality, diffusion MRI measures the random water molecule movements in biological tissue and the constraints on diffusion imposed by the physical membranes of axons, as a way to explore the white matter (WM) brain fibers. It is an important mechanism to study brain WM network properties and neurological/mental disorders. Functional MRI is an imaging technique that can detect the neuroactivity based on changes in the blood flow, i.e. the MR signals are believed to reflect the neuron activations in particular area that are coupled with the increasing blood supply (Ogawa et al., 1990). From that, spatial and temporal functional patterns can be constructed, which are known as functional connectivities that represent the relationships between different functional regions of the brain. Therefore, most of the projects that study brain (network) structures and cognitive functions are using dMRI and fMRI, and they become the standard in structural and functional connectivity neuroscience acquisitions.

In the last decade, many well-funded projects that are launched, aim to collect and analyse datasets that can address those big questions, such as the Human Connectome Project (HCP) (Elam and Essen, 2013), The Human Brain Project (HBP) (Markram, 2012) and BRAIN (Bargmann and Newsome, 2014) have invested a large number of human resources and fundings to study the human brain from the perspectives of both structure and function. In those projects, brain structural and functional data are acquired from a large number of populations, algorithms of pre-processing and analyzing data are developed, and corresponding toolkits and software packages are published (Fischl, 2012; Friston et al., 2003; Tournier et al., 2012; Jenkinson et al., 2012). One of the main motivations of those projects, particularly HCP project is to share high quality dataset with all the researchers all over the world, so that more findings can be published using those data.

With the availability of large amount of public datasets, researchers who use these datasets no longer need to worry about possible issues occur in the stage of data acquisition, such as artifacts caused by patient motion and eddy current effect. At the same time, it also opens more possibilities of brain studies that require large amount of data. However, it raises one of the challenges, which is the efficiency of processing those "big" data on conventional desktop-bound machines which are no longer meet of the requirement of frontier neuroscience research, especially in exploratory analysis environment.

And, (nearly) “real-time” process of both structural and functional brain analysis are highly demanded by many researchers in the field of neuroimaging and neuroscience. It introduces new algorithmic and computational challenges, due to the complexity of imaging algorithms and limitations of conventional computing architectures. Algorithms like diffusion tractography for analyzing brain white matter fiber structure (i.e. the structural connectivity), and (spatial and temporal) functional map extraction for studying neural activities (i.e. the functional connectivity), require high computing power (Farquharson et al., 2013; Zhang et al., 2005; Sulzer et al., 2013). Fast tractography implementation may bring the benefits that are not limited to obtaining results more quickly, but also making tractography-based diagnosis feasible in clinical environment (Farquharson et al., 2013; Goebel and Linden, 2014).

Also, techniques that enable “real-time” neural activity studies in fMRI become one of the most popular research areas, that can extensively enhance existing fMRI-based experimental paradigms and allow for more sophisticated designs, like neurofeedback (NF) (Gunkelman and Johnstone, 2005; Sherlin et al., 2011). NF is a very attractive experiment design, which enables subject self-regulation of his/her brain functions, according to the ’signal/information’ relayed back to the subject. Unlike conventional NF designs that are based on particular Region of Interest (ROI), more dynamic approaches like sliding-window-based independent component analysis (ICA) of fMRI data across the whole brain can provide more “exploratory” and possibly valuable information (Calhoun et al., 2003, 2009). Using multivariate analysis algorithm like ICA to extract spatial and temporal components (i.e. patterns of regional activations and time course) from the original time-series data, is still limited to off-line cases. Correspondingly, real-time whole-brain dynamic ICA-based approach has not been achieved, due to its high computation complexity and long processing time cost.

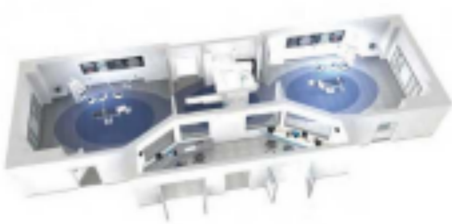
Besides, another obvious use case that essentially requires both the “real-time” tractography and data-driven fMRI methods is image-guided neurosurgery navigation, particularly the systems using intraoperative MRI (iMRI). It is widely accepted that the risks of disease recurrence (too little resection) and potential cognitive deficits (too much resection) are critical to consider, in order to maximize the recovery chance of a patient (Lacroix et al., 2001; Sanai et al., 2011; McGirt et al., 2009). Unfortunately, navigation systems that rely on preoperative neuroimaging data (e.g. ultrasound (Unsgaard et al., 2002; Lange et al., 2004), CT (Enislidis et al., 1997; Lustmann, 2005) and MRI (Maldjian et al., 1997; Mäkelä et al., 2001; Wilkinson et al., 2014)) still lack spatial accuracy, due to brain deformation (i.e. “brain shift”) in ongoing surgical procedures (Roberts et al., 1998; Fahlbusch et al., 2000). Brain shift is found to be a continuous dynamic process that occurs differently in distinct brain regions (Mislow et al., 2010), which can lead to unexpected errors (e.g. registration inaccuracy). Although the reliability of using preoperative image data can be improve by the brain shift compensation methods, they are not sufficient to accurately locate and identify brain tumors, especially deep brain tumors (Nimsky et al., 2001). Therefore, the intraoperative imaging techniques are believed to be able to quantitatively analysis and inspect the brain shift phenomenon, and evaluate the compensation models (Nimsky et al., 2000).

Intraoperative navigation is a technique that uses the images acquired after craniotomy is performed, and provides real-time, image-guided surgery. Intraoperatively acquired images and data can compensate the effects of brain shift, provide information with minimized uncertainties that can result in either inadequate and aggressive resections, which is the only way that can provide accurate image guidance (Gering et al., 2001; Nabavi et al., 2001), and also serve as an immediate mechanism of obtaining feedback regarding the surgical result. Intraoperative diffusion tensor imaging (DTI) and tractography algorithms can provide the information of underlying white matter fibers; together with an integration of intraoperative functional MRI data (Maldjian et al., 1997; Roux et al., 1999; Nimsky et al., 2004) and awake cortical mapping images (Bello et al., 2010; Gasser et al., 2005), it is achievable to maximize extent of resection, with a preservation of neurological functions.

In the meantime, the intraoperative hardware (i.e. scanner) and software solution are developing towards "real-time" data processing. The intraoperative MRI scanners are evolving from open-configuration solutions since 1980s (Grönemeyer et al., 1996), e.g. PoleStar N10 (0.12 Tesla) and more recently introduced N20 with a magnetic field of 0.15 Tesla (Hadani et al., 2001; Gerlach et al., 2008,?), to closed-configuration (i.e. closed-bore) system solutions. While the low-field open-configuration solutions can not yield good image resolution, the hybrid closed-bore system (that integrates the surgery platform and the MR imaging system) can provide much higher quality of intraoperative images. Figure 1.1a shows one of the system developed by IMRIS. Inc (http://imris.com), which mounts a 1.5T scanner using a ceiling-mounted rail system that enables moving the scanner from the room to the operating room, which are separated via sliding radiofrequency- and sound-shielded doors; also the scanner is able to be shared by two operating rooms, as shown in figure 1.1b (Hushek et al., 2008). Furthermore, the new generation of the IMRIS suites will feature 3-T magnetic field, which provides higher intraoperative imaging capabilities. At the same time, in the software packages used in intraoperative image-guided navigation, e.g. 3D slicer (Pieper et al., 2004), Medical Imaging Interaction Toolkit (MITK) (Wolf et al., 2005) and Image-Guided Surgery Toolkit (IGSTK) (Enquobahrie et al., 2007), the methods, e.g. high-order diffusion tractography, and ICA-based fMRI analysis, are not implemented efficiently to meet the "real-time" requirements, like the situation of intraoperative image-guided neurosurgery, due to high computing time.

While the challenges described in the above use cases still remain unsolved, it is noticed that accelerated computing techniques and hardware are opening up new possibility in those research and application areas. With contemporary parallel computing platforms, it is feasible to achieve fast tractography and real-time fMRI by implementing and/or improving existing algorithms on parallel architectures. Since (both structural and functional) MRI-based processing and analysis generally occur in (2D/3D/4D) image domain, processing MR data is inherently suitable for parallel computing techniques,

(a) An Intraoperative Scanner



(b) An iMRI Solution Design

Figure 1.1: Illustration of Intraoperative MRI Systems (http://imris.com)

i.e. exploration of data and algorithmic parallelism is an efficient way of accelerating the algorithms.

In this dissertation, I aim to explore approaches to analyze algorithms in structural and functional brain connectivity analysis (e.g. high-order diffusion tractography and ICA-based fMRI analysis), map them on parallel architectures, and integrate into pipeline that enables "interactive" fiber tractography and "real-time" whole brain fMRI analysis system, for both research and clinical usage.

The structure of the thesis is organized into 5 chapters, which start with a detailed background of both diffusion tractography and functional MRI analysis pipeline, as well as a recap of GPU architecture from an implementational perspective. Chapter 3 reports the two accelerated tractography frameworks that achieve significant speedups and Chapter 4 proposes a framework that focuses on real-time data-driven fMRI analysis framework, mainly with the acceleration of FastICA algorithm on GPU. Finally, Chapter 5 concludes the work and state possible future directions.

Chapter 2

Background

2.1 Parallel Computing using CUDA

Graphical processing units (GPUs) were initially designed for creating, manipulating and rendering image frames on a computer display. Due to their energy-efficient massive parallel processing capability and the advent of both programmable shader processors and floating-point support, they have become attractive for scientific studies to improve application performance. Since 2001, the paradigm of general purpose GPU (GPGPU) has seen wide use in many research areas with applications proposed that achieve speedups ranging from a few to several orders of magnitude, for example, the N-body problem in astronomy (Zwart et al., 2007), digital correlator for radio astronomy arrays (Harris, 2008; Wayth et al., 2009), and registration of deformable 3D image in neuroimaging (Mousazadeh et al., 2011).

To make GPUs more programmably accessible, GPU vendors and related organizations have developed sophisticated middle-layer frameworks, such as NVIDIA's Compute Unified Device Architecture (CUDA) (Luebke, 2008) and Open-Computing Language (OpenCL) (Stone et al., 2010) by Apple and Khronos. With CUDA and OpenCL, users are able to explicitly manipulate different GPU memory storage layers, and configure and launch massive parallel processes to solve problems. Today, most contemporary supercomputers are powered by hundreds or thousands of GPU processors and GPGPU is playing an important role in modern day large-scaling data processing.

2.1.1 GPU System Overview

A typical heterogeneous CPU-GPU computing system (shown in figure 2.1) has both the 'host' system (i.e. CPU system) and the dedicated GPU accelerator (i.e. device system in CUDA terminology), and they work as two independent hardware components that communicate by transferring data. They also have separate memory storages, and can not perform cross access. In other words, the data on CPU system is required to be copied to GPU memory system so that the GPU can access and vice versa. This leads to the most common issue that inter-system data transfers are inevitable, which may limit performance in some applications. CUDA version 6 and above enable an apparent unified memory space, hiding the explicit data transfer between CPU and GPU, and making GPU programming simpler, although less explicit in many cases (Landaverde et al., 2014).

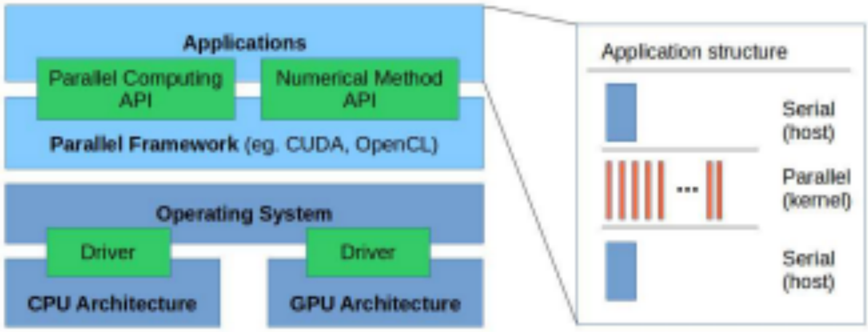


Figure 2.1: GPU Application Overview

GPU applications rely on the middle layer (i.e. parallel frameworks, such as CUDA and OpenCL) to interact with the host operating system and utilize the GPU accelerator (i.e. Device). Both host and device systems need to be properly driven by the operating system (OS) on which the parallel frameworks are installed and configured. Applications exploit the parallel framework by using their application programming interfaces (APIs). Common numerical and functional methods are implemented and included in the framework for the purpose of easy usage.

A classic GPU execution model involves a serial portion that runs on the host, and a parallel portion runs on the device (i.e. the GPU kernels). A so-called ”kernel” is defined as a piece of code that is executed on a GPU or multiple GPUs in the form of concurrent lightweight threads. According to the well known the Amdahl's law (Amdahl, 2013), the theoretical performance (i.e. speedup) of a parallelized algorithm is limited the portion of the serial part in the algorithm. In other words, larger proportion of parallelized code in the algorithm leads to higher speedup. In the case of accelerating algorithm on GPUs, the degree to which an algorithm can be parallelized, and/or the level to which it can be accelerated by using GPU hardware, depends on how well the algorithm can be mapped onto the GPU architecture (within the limit determined by the Amdahl's law), which is the main consideration before attempting to map algorithms onto GPUs.

2.1.1.1 GPU Chip Architecture

For historical reasons, contemporary GPU architectures are designed as a modified form of the traditional stream processor such as vector processors/array processors (Johnson, 1989), for the purpose of turning the graphics accelerator's shader pipeline into massive general-purpose computing resources, known as the paradigm of general-purpose GPU (GPGPU). Unlike CPU chips, a GPU chip contains a large number of relatively low-clock-rate cores which are organized in groups called “Streaming Multi-processors” (SMs). Within each SM, there are mainly four types of processing cores in most of the modern GPUs, which are integer and floating-point arithmetic processing units(ALUs), double-precision processing units (DPUs), special function units (SFUs) and data load/store units

(LD/ST units). As shown in figure 2.2, the layouts of CPU and GPU chips are different in the way that most of the silicon on a GPU are computing units and on-chip DRAM, and the GPU processes smaller control circuits and on-chip cache compared to a CPU chip.

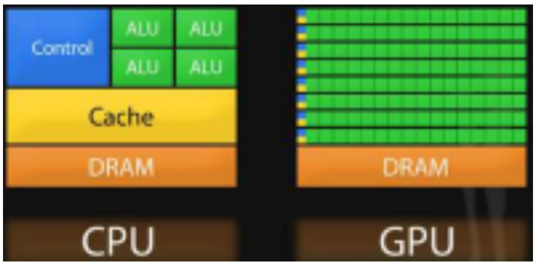


Figure 2.2: Different design between CPU and GPU architectures (http://www.frontiersin.org)

The ALUs and DPUs are the main processing power that perform computing operations on floating-point and double precision data types. SFUs are hardware components designed to accelerate frequently used functions, such as sine, cosine, and square root, which are known as fast approximate transcendental operations. The LD/ST units are in charge of copying data from/to on-chip cache and DRAM on board. Due to this highly parallel architecture, GPUs are capable of yielding several orders of magnitude higher performance than conventional CPUs, providing their ALUs and/or DPUs can be fed data at a sufficiently high rate to exploit their capacity.

2.1.1.2 Memory Hierarchy

GPUs support access to various types of memory on both host (i.e. CPU memory) and device (i.e. GPU global memory, shared memory, register). GPU memory is separated from host memory, which means that host memory is not accessible by processes running on devices, and vice versa. During the stage of initializing a GPU program, data residing on the host memory is explicitly copied to the GPU memory space which is also known as the on-board GPU global memory. After GPU kernels complete, results are optionally copied back onto the host memory for the purpose of visualizing, post-processing and storing.

The properties of these types of the GPU memory are different, in terms of capacity and speed. The global memory has the largest capacity, but relatively slow access performance. It is a specific type of Dynamic Random Access Memory (DRAM) that is designed for GPU architectures, and is several times faster than conventional CPU memory. However, it is still relatively slow, compared to the high throughputs that contemporary GPUs can offer (Luebke, 2008). Because of this, the latency of accessing the global memory becomes one of the major bottlenecks that limit the overall performance of most GPU applications.

This problem is partially solved by introducing on-chip fast cache space: shared memory and registers. As shown in table 2.1, those memory storages have limited sizes (e.g. 48KB shared memory and 65,535 registers on each SM), but much higher bandwidths, compared to the global memory. Also, they are designed with different access scopes. As

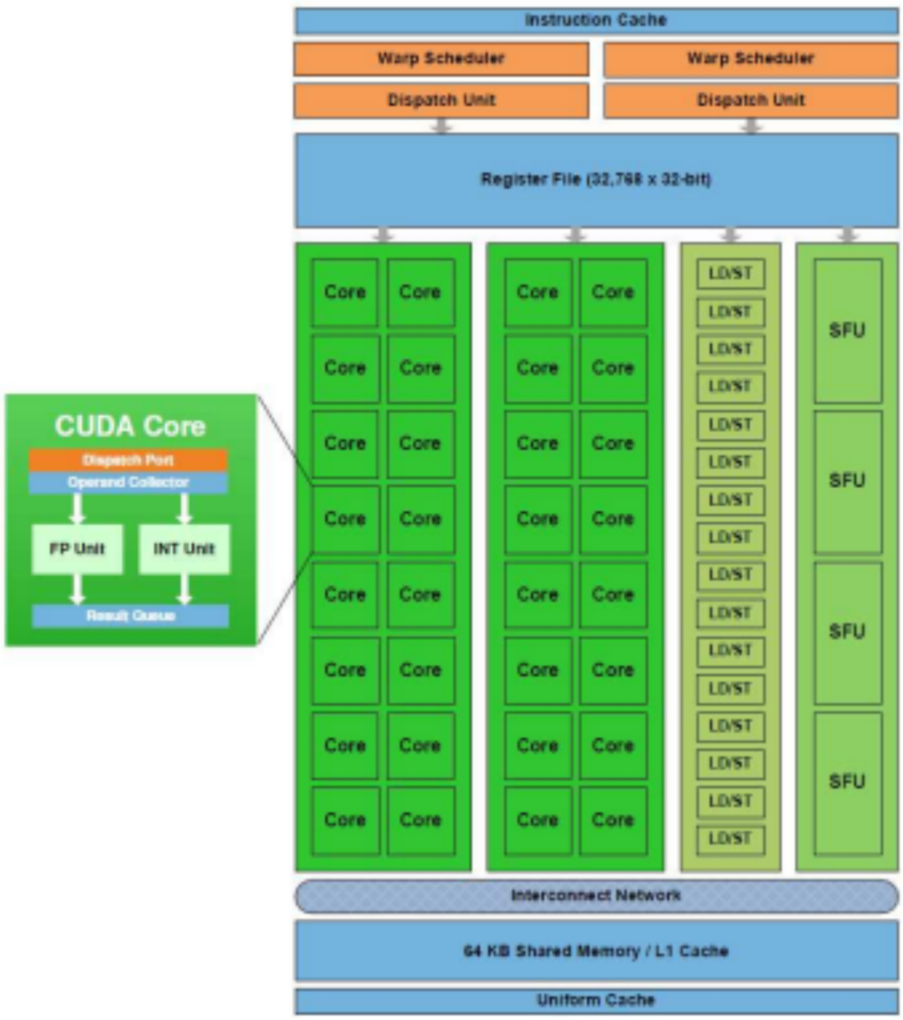


Figure 2.3: GPU Streaming Processor Architecture (Fermi) (Wittenbrink et al., 2011)

shown in figure 2.4, shared memory is only accessible for all the threads from the same group (i.e. thread block in CUDA terminology or work group in OpenCL). By caching the necessary data in shared memory, efficient inter-thread communication is enabled and redundant global memory are reduced.

Register memory, on the other hand, resides on the GPU chip and is the closest to the GPU processing cores. It is the only type of memory that can match the throughput of the GPU cores. It is the fastest memory storage on GPU chip, however, has the smallest size. For example, a commodity GPU processor based on Maxwell architecture has only 65,535 32-bit registers on each SM. Unlike other types of memory, registers are private to each thread, i.e. only accessible for the corresponding thread, and accordingly no communication or data sharing can be achieved using registers. Therefore, register memory is usually exploited to store variables that are small in size and frequently used, and local and temporary variables.

The strategic utilization of these different memory and limited cache resources is one of the key factors to achieve high performance. Besides, each SM has its own Level-1

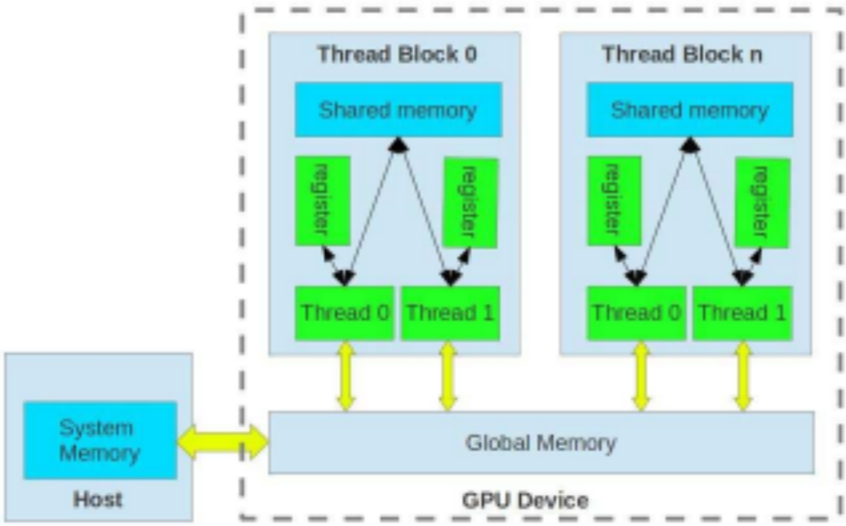


Figure 2.4: Typical GPU Memory Hierarchy

	M2070	K20
CUDA cores	448(32*16 SMs)	2496 (192 * 13 SMs)
Shared Memory Size	16KB/48KB	16KB/48KB
Registers capacity	32768*32-bits	65535*32-bits
Global Memory Capacity	6 GB	5 GB
Global Memory Frequency	150 GB/s	208 GB/s

Table 2.1: Sample Specifications of Two GPUs

cache, while Level-2 cache is shared by all the SMs on the same chip. Texture memory is also used by some GPU applications, as it supports built-in interpolation functions.

2.1.2 CUDA Programming Model

In the CUDA programming framework, a kernel program that runs on the GPU is executed by all the threads in parallel, in a manner of Single Instruction Multiple Data (SIMD). The SIMD parallel model applies the same instruction to different chunks of the data synchronously. The SIMD model is typically ideal for data-parallel tasks, however, its performance is still highly related to the algorithm itself and optimization of the implementation.

Each thread can be identified by its grid index, thread block index and local thread index, which are the typical identifiers used to manage large number of threads on GPU frameworks like CUDA and OpenCL. Threads can be grouped into blocks indexed in 1D, 2D or 3D. As shown in figure 2.5, the dimension of a block is defined by two variables: blockDim.x and blockDim.y. Several blocks can be further grouped together into grids, which can also be in 1D/2D indexing space. To identify a thread in the global level, its id can be computed using its block dimension, block index, and its local thread index: threadIdx.x, threadIdx.y and threadIdx.z. The configuration of the threads is determined during the kernel launching process, by specifying the grid and block dimension information.

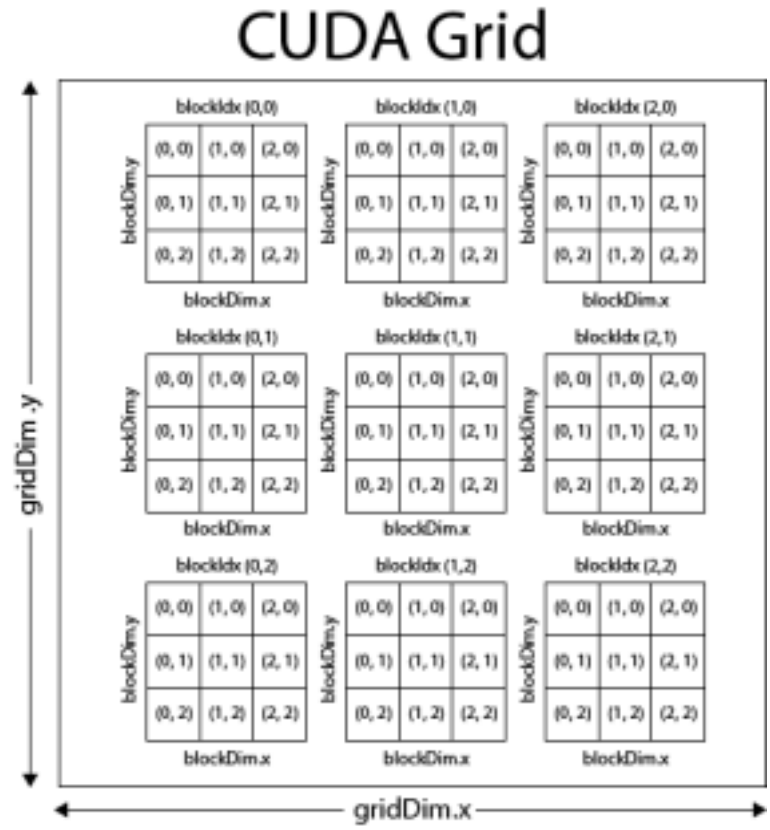


Figure 2.5: CUDA Thread Grid Management (Lindholm et al., 2008)

The efficiency of the GPU is principally determined by the number of active thread warps running at the same time. A warp is a group of 32 threads within the same thread block, that is dispatched by the on-board thread scheduler. Thread blocks with more than 32 threads are separated into multiple warps and executed on the same SM processor in an interleaved manner. Also, multiple thread blocks can be allocated onto a single SM processor. The dispatching of thread warps starts with memory allocation (i.e. register and shared memory allocations). The number of warps that can be assigned to a SM processor is restricted by the usage of both register and shared memory by the kernel and the memory (i.e. register and shared memory) capacity available on the chip. Usually, the number of threads allocated to a SM processor is larger than the actual number of processing cores on that SM processor. This allows the SM to switch and execute other thread warps while the threads in the current warps are accessing the global memory. This is the mechanism that GPU architectures use to ameliorate the effects of the high memory access latency.

A typical parallel program in CUDA can be seen in Listing 2.1, which contains two essential parts of a CUDA program, which are device code and host code. In this example, the host code in the "main" function, including data declaration, initialisation, memory transfers and kernel invocation, is executed on the CPU, while the kernel function (i.e.

sumvect function) that is declared by the keyword, *__global__*, is the device code that runs on the GPU. Explicit data copy is required by invoking the built-in memory copy kernel function. The kernel function is called by specifying the thread configuration and passing the parameters. And, explicit memory releases are necessary on both host and device sides. Besides, an one-to-one value correction check is usually conducted to make sure the kernel provide correct results.

Listing 2.1: Sample CUDA Program of Vector Summation

```
#include <stdio.h>
#include <cuda.h>

__global__ void sum_vect(float  * x, float * y, float * z, int n)
{
        int idx= blockIdx.x * blockDim.x + threadIdx.x;
        if(idx <  n)
        {
                z[idx] = x[idx] + y[idx];
        }
}

int main()
{
        float *x_h, *y_h, *z_h;
        float *x_d, *y_d, *z_d;
        int n= 100000; //Vector size
        int i;

        size_t size= n * sizeof(float);
        x_h= (float *)malloc(size);
        y_h= (float *)malloc(size);
        z_h= (float *)malloc(size);

        for(i=0; i < n; i++)
        {
                x_h[i]= (float) i;
                y_h[i]= (float) i;
        }

        //allocate GPU device memory
        cudaMalloc((void **)&x_d, size);
        cudaMalloc((void **)&y_d, size);
        cudaMalloc((void **)&z_d, size);

        cudaMemcpy(x_d, x_h, size, cudaMemcpyHostToDevice);
        cudaMemcpy(y_d, y_h, size, cudaMemcpyHostToDevice);

        int block_size= 256;
        int num_blocks= (n + block_size - 1) / block_size;

    //Launch kernel
        sum_vect <<<num_blocks,  block_size>>> (x_d, y_d, z_d, n);

        cudaMemcpy(z_h, z_d, size, cudaMemcpyDeviceToHost);

        //Correction checking
        for(i = 0; i < n ; i++)
        {
```

```
                if(z_h[i] == (x_h[i] * y_h[i]))
                        printf("Success\n");
                else
                        printf("Fail\n");
        }

        free(x_h);
        free(y_h);
        free(z_h);

        cudaFree(x_d);
        cudaFree(y_d);
        cudaFree(z_d);
}
```

Dynamic Parallelism

In CUDA 5.0, dynamic parallelism was introduced, which enables a running GPU kernel to create new tasks and spawn/launch new kernels on the GPU. Before dynamic parallelism, the only way to create and execute GPU code was to invoke kernels from the host thread. In CUDA dynamic parallelism, a parent thread grid can directly launch kernels without involving CPU executions, which are called child grids. A child grid inherits certain properties from its parent grid, like the L1 cache/shared memory configuration. Also note that a kernel launched with dynamic parallelism is executed by every thread in its parent grids. For example, if the parent grid has a configuration of 10 thread blocks of 128 threads, there will be 1280 kernel launches. To restrict child kernel launches to certain threads or thread blocks, thread branching statements are needed, as shown in Listing 2.2:

Listing 2.2: Thread branching in dynamic parallelism

```
#launch from certain threads
if(threadIdx.x == 0){
    child_kernel<<<thread configuration>>>();
}

#launch from certain thread blocks
if(blockIdx.x ==0){
    child_kernel<<<thread configuration>>>();
}
```

Child grids that are launched with dynamic parallelism are fully nested. As shown in figure 2.6, grid A is the parent grid for grid B. In other words, grid B is launched from grid A with dynamic parallelism. In this case, grid A will not complete until all the threads in grid B finish, even without explicit synchronisations code.

A parent grid and its child grids share the same global and constant memory storage, and the view of the global memory is consistent for both the parent and child grids. This means that if a parent grid writes to a location in global memory, then the child grid launched with dynamic parallelism will see the same value. Similarly, when the child grid updates values in the global memory during its execution, its parent grid will see the updated values. Memory consistency is also guaranteed for several child grids that execute sequentially from the same stream.

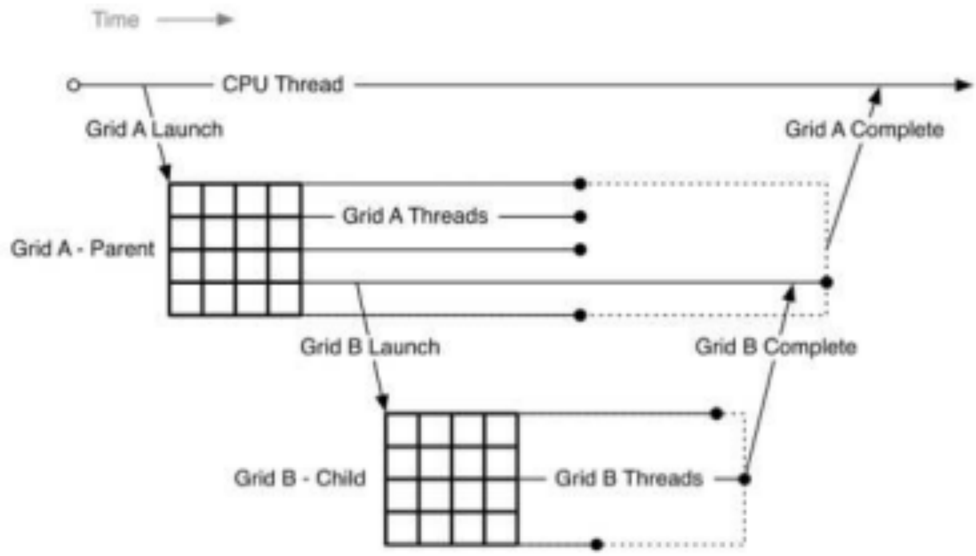


Figure 2.6: Nested Structure of Dynamic Parallelism (Nvidia, 2011)

Table 2.2: Corresponding Terminology in CUDA and OpenCL

CUDA	**OpenCL**
thread	work-item
thread block	work-group
shared memory	local memory
global memory	global memory
local memory	private memory

The introduction of dynamic parallelism technique enables recursive programming patterns and allows more dynamic parallel designs on GPU platforms. Formerly, kernels are restricted to be launched from host threads and the configurations of the grid (including number of threads per block and number of blocks) are determined before the kernel invocation and remain fixed during the period of kernel execution. The single-level of parallelism may not be sufficiently efficient to solve problems that can gain extra benefits by using recursion and irregular loop structures. For example, an image processing program requires to process part of the image in a finer grained scale, which can be efficiently achieved using dynamic parallelism by invoking additional child threads for the targeted part of the image (Wang et al., 2013).

CUDA and OpenCL

The parallel model can be efficiently accessed by using the sophisticated GPU computing frameworks, such as NVIDIAs Compute Unified Device Architecture (CUDA) (Wilt, 2013) and Khronoss Open Computing Language(OpenCL) (Stone et al., 2010) standards. They are both extensions of the C programming language model with additional keywords and types. CUDA and OpenCL have a very similar framework design, including thread management and memory hierarchy. Table 2.2 shows the mapping of terminology between CUDA and OpenCL.

2.1.3 Parallel Mapping Patterns on GPU

The performance of a parallel mapped algorithm on GPU can be latency-bound, compute-bound and/or memory-bound. Most parallel mappings can be categorized as one of these three types, and each is caused by different constraints and can be partially or fully ameliorated in different ways.

2.1.3.1 Latency-bound Pattern

The main way to approach peak utilization of the GPU is to hide computational latency by having fast warp switching between a large number of running threads. Like CPU programs, latency is mainly caused by accessing memory, which can be compensated by interleaving warp executions. When there are no ready-to-compute warps to schedule, the streaming processor will be idle, and the program is latency-bounded.

The toy example in figure 2.7 illustrates one of most common cases of latency. The sample program starts with instructions of loading two variables from (global) memory storage (i.e. Instruction 0) which usually costs a few GPU cycles, and compute the sum of the two variables(i.e. Instruction 1). Instruction 1 can not be executed until Instruction 0 completes. In this toy example, the GPU kernel is launched with 4 warps of threads running on a GPU with 3 streaming processors (SPs). On the first SP, while threads in warp 0 load data from global memory, the latency is hidden by switching and executing warp 3. After warp 0 completes loading data, it continues the execution of summation. On the other hand, due to insufficient number of warps to cover the latency and dependency on the previous instruction, the second and third SPs in the example keep idle. In this case, the sample program is latency-bounded and computing resources are wasted.

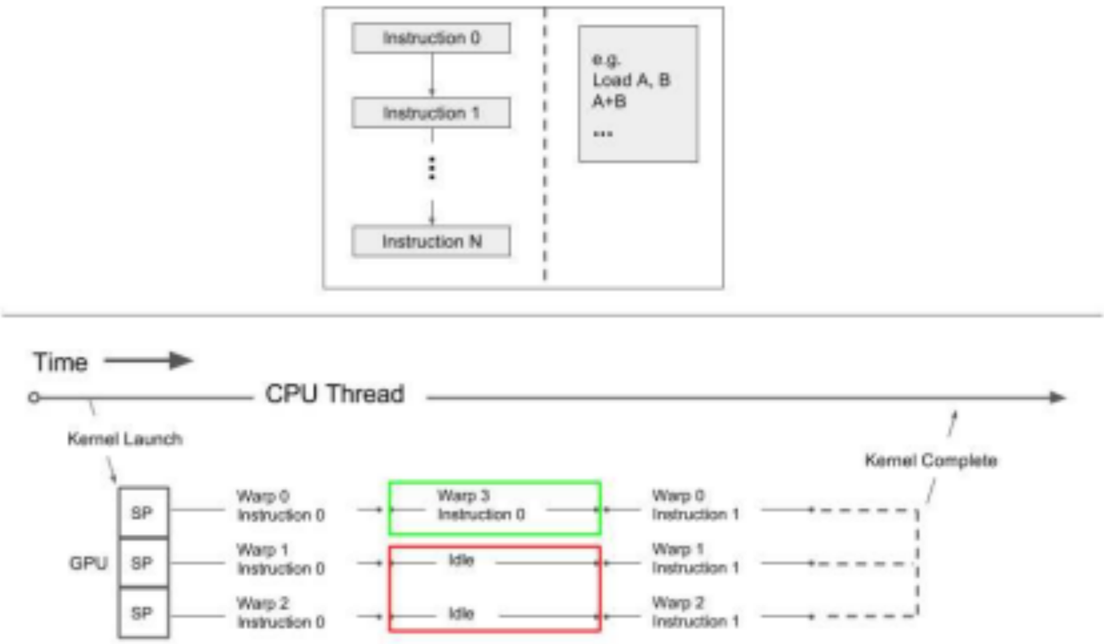


Figure 2.7: Illustration of Latency-bound Pattern

Table 2.3: Specifications of Limitations of Different Compute Capabilities

Limitations	**Compute 2.x**	**Compute 3.x**	**Compute 5.0**	**Compute 5.2**
Max Threads / Block	1024	1024	1024	1024
Warp Size	32	32	32	32
Registers / SP	32768	65536	65536	65536
Shared Memory / SP	16KB/48KB	16KB/32KB/48KB	64KB	96KB
Max Blocks / SP	8	16	32	32
Max Warps / SP	48	64	64	64
Max Threads / SP	1536	2048	2048	2048

The solution is to minimize the impacts of processor stalls/idles, by optimizing the GPU occupancy, which is the ratio of the number of running threads to the theoretical maximum of threads that can be run on the device(i.e. GPU). Occupancy can be controlled by the choice of thread block size, usage of shared memory, usage of registers, and impacted by limitations of the hardware. Some sample specifications of GPU hardware with different compute capabilities can be seen in table 2.3. To optimize latency-bound kernels, users need to adjust the kernel launching configurations(i.e. thread block size and grid size), and optimize the usage of shared memory and registers in the source code of kernels, for a specific target GPU. Besides, an important point to mention here is that the thread topology must be de-coupled from the data topology in order to achieve a decent performance.

Higher occupancy usually means better performance, but it is not always the case. Arithmetic latency and dependency (i.e. the sequential order of computation required by the algorithm) in a GPU kernel can be overcome with 'enough' work, due to the fast warp switching technique. It also points out that wisely use registers is the key factor to achieve the peak performance of a particular GPU hardware, since only registers are fast enough for the GPU processing cores. This may result in more registers are used in a kernel and correspondingly lower occupancy, while obtain higher performance. Besides, when all the SM processors are 'fully' occupied by the running threads, additional gains in occupancy will not further improve the performance either.

2.1.3.2 Compute-bound Pattern

GPU parallel designs follow the SIMD (i.e. Single Instruction Multiple Data) parallel model, which means that all the threads in a thread warp (e.g. usually 32 threads for mainstream GPU architectures) execute identical instruction series and a full efficiency can be achieved. However, in some GPU applications, conditional branches are introduced, which causes thread branching/warp divergence. A typical pattern of thread branching kernel written in CUDA shown in listing 2.3.

When threads of a warp diverge via a conditional branch, the warp serially executes each branch path. In the the above example, the statements within if-else branching block are executed by all the threads within the same warp, although threads other than thread 0 are idle when executing instructions in the if block and vice versa. A clearer illustration can be seen in figure 2.8. The section highlighted in red rectangles are the compute cycles where some of the cores are idle, which leads to nearly 50% compute resources being wasted when executing the sample kernel. The hardware is underutilized when thread branching occurs. Therefore, kernels with compute-bound pattern (e.g. thread branching) should be avoided in mapping algorithms onto GPU processors: this is one of key rules to follow in order to achieve high efficiency.

Listing 2.3: A typical CUDA code pattern of thread branching

```
__global__ void kernel(){
        int lid = threadIdx.x;  /* Thread local ID */

        //Thread 0
        if(lid==0){
                #does something
        }
        //Other threads
        else{
                #does another thing
        }
}
```

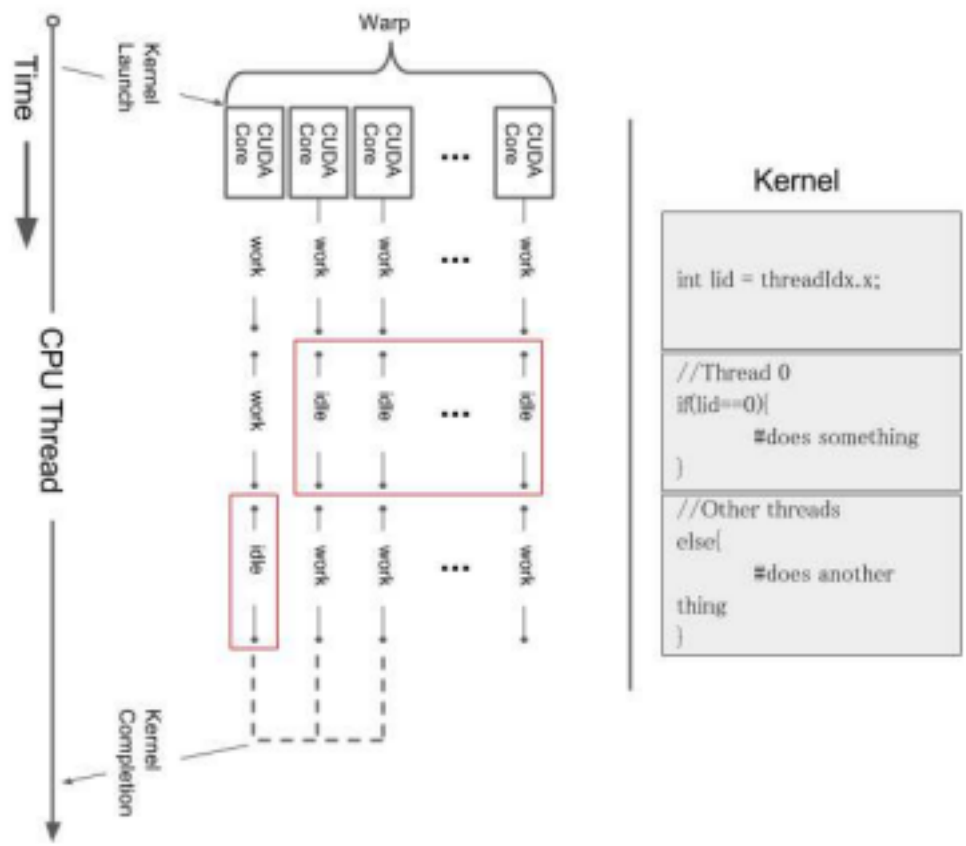


Figure 2.8: Illustration of Compute-bound pattern (Thread branching)

2.1.3.3 Memory-bound Pattern

Memory-bound kernels are very common in GPU applications, which is related to proper configuration and usage of different types of GPU memory stores: global memory, shared memory and registers. As discussed in the section on GPU memory hierarchy (2.4), the GPU memory subsystems have different properties of capacity and speed. Global memory has the largest capacity, but the lowest bandwidth. Shared memory is on-chip storage, which provides high bandwidth, but very limited size. Its main function is to improve data locality (i.e. make data closer to the GPU cores) and enable inter-threads communications within a thread block. Registers are the fastest, but smallest on-chip storage.

Understanding global memory access patterns (i.e. proper alignment and reading/writing of memory) is the key to improving the performance of memory-bounded kernels. Global memory is most efficient when all the threads in a warp access aligned and contiguous regions of the memory locations. In this way, individual thread requests can be combined/coalesced (by the hardware) into fewer transactions. As shown in figure 2.9, there are four common memory access patterns, which are distinguished from two perspectives: alignment to memory boundary and access contiguity.

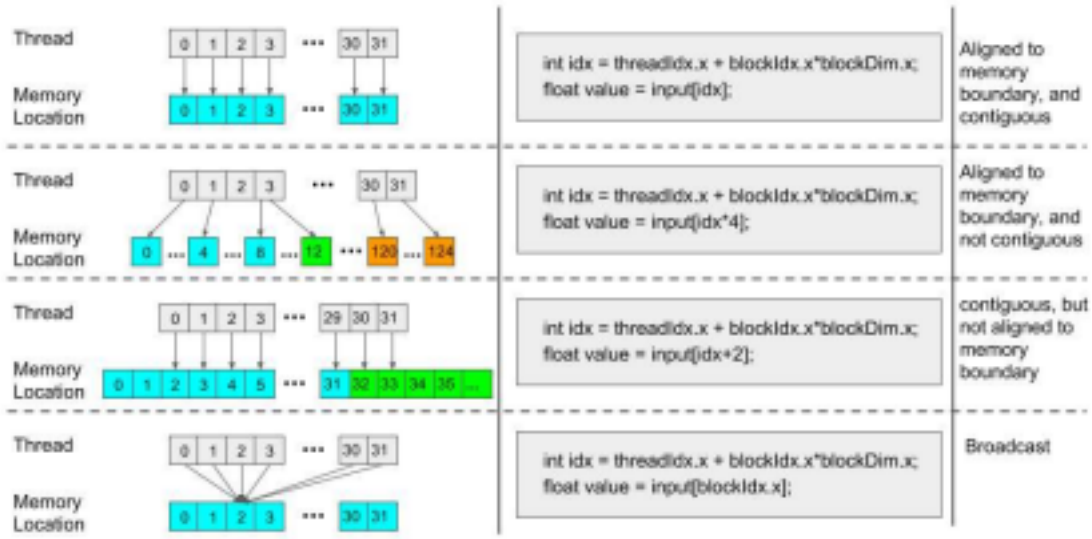


Figure 2.9: Common Global Memory Access Patterns

Grouping memory reads/writes from/to a continuous memory space (i.e. memory coalescing technique) can significantly reduce the number of memory access requests, therefore, minimize the the global aggregate latency and maximize the effciency of utilizing memory bandwidth. As shown in the first pattern of figure 2.9, threads in a warp access aligned and contiguous regions of memory. Coalesced memory accesses of threads can be reduced into a minimal number of memory transactions, in order words, the memory access latency is minimized. In contrast, the second pattern shows a less efficient way of accessing memory locations that are aligned to boundary, but strided by 4 elements (i.e. not contiguous). To fetch the same amount of data in this case, it needs 4 times more memory transactions and loads 4 times larger chunk of memory including a large portion (3/4) of unused data. In the third pattern, unlike the first pattern, it requests a contiguous region of memory that are not aligned to memory boundary, which costs more transactions of data fetching. The last pattern, known as broadcast, in which all the

Table 2.4: An Example of Efficiencies of Different Memory Access Patterns

	Request Size	Fetching Size	No. of Transactions	Efficiency
Pattern 1	128 Bytes	128 Bytes	4	100%
Pattern 2	128 Bytes	512 Bytes	16	25%
Pattern 3	128 Bytes	160 Bytes	5	80%
Pattern 4	4 Bytes	32 Bytes	1	12.5%

threads in a warp access the same element, and the entire corresponding chunk of memory is fetched, with majority of the data are not used.

In table 2.4, it shows an example of the efficiencies of loading 128 bytes by a thread warp where each thread requests 4 bytes data, using the four different memory access patterns. The efficiency in this case is defined as the percentage of data used out of the data that are actually fetched from the memory. Assume that each chunk of global memory is 32 bytes, which is also the smallest unit that can be fetched from the global memory and can be completed in one memory transaction. In pattern 1, 2 and 3, fetching 128 bytes of data requires 4, 16 and 5 transactions respectively, and the actual fetched data are in different sizes: 128 bytes, 512 bytes and 160 bytes, which results in the efficiencies of 100%, 25% and 80% correspondingly. In pattern 4, 32 bytes of data is fetched in one transaction, but only 4 bytes are used, so the efficiency is as low as 12.5%. Therefore, it is clear that choosing coalesced memory access pattern can achieve the best efficiency of access global memory and avoid redundant data fetching, which can significantly improve the performance of an memory-bound application.

Avoiding frequent global memory access using on-chip registers and shared memory can significantly improve the performance by reducing global memory accesses. As discussed in the previous sections, a GPU has fast caching space, in both thread-level scope (i.e. registers) and thread-block-level (i.e. shared memory). By pre-loading frequently used data into cache, an application can, to some extent, avoid the high access latency of accessing global memory. However, optimally using cache memory is not straightforward, nor always possible, due to the very limited cache space in the GPU.

On a GPU chip, the shared memory is divided into equally sized memory 'banks' that can be accessed simultaneously, for achieving high bandwidth for concurrent accesses. For example, a GPU with Kepler architecture (from NVIDIA) has 32 banks and each bank has a bandwidth of 64-bits per GPU clock cycle and 32-bits in previous generations of architectures (e.g. Fermi). To yield an effective bandwidth, any memory read/load or write/store of addresses that span distinct memory banks can be serviced at the same time. Otherwise, if multiple threads request data from addresses that are mapped to the same memory bank, the accesses are serialized, which is known as 'bank conflicts'.

When bank conflicts occur in shared memory accesses, the memory requests are split into as many as conflict-free requests as necessary, which results in a decreased efficiency of using bandwidth by a factor of the number of the conflicted memory requests. An exception is when all the threads in a warp access the same bank and the same element in the bank, which is known as a 'broadcasting' in figure 2.10. In this case, multiple shared memory accesses to the same location by any number of threads in a warp can be served simultaneously. Similarly, a multicasting can also be an efficient way of accessing shared memory, in which, several threads in a warp request the same value from a particular bank and the value will be read only once from the shared memory.

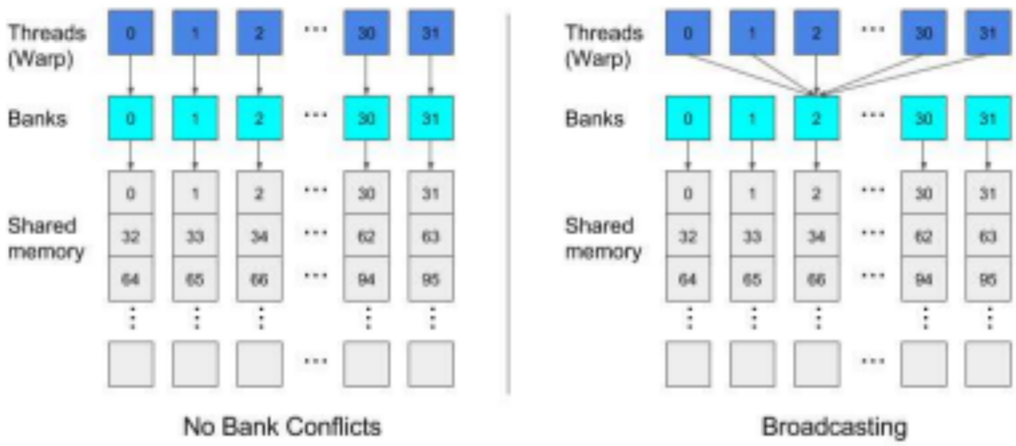


Figure 2.10: No Bank Conflict and Broadcasting

To avoid and minimize bank conflicts, it is necessary to understand the way that memory addresses are mapped to memory banks. The shared memory space is organized in words. A word is 4-byte memory space (i.e. 32 bits), which can store numeric types, such as 32-bit int and float. In a GPU architecture that has 32 banks, words are assigned to successive banks and the bandwidth is 32 bits per bank per clock cycle (and 64-bits for architectures like Kepler and Maxwell; here we use 32-bits bandwidth per bank for illustration purpose.). For example, as shown in figure 2.10, bank 0 has all the words with indices that are divisible by 32; bank 1 has the words that leave a reminder of 1 when divided by 32 and etc. In figure 2.11, it shows an example that a 2-way bank conflicts occur, where every thread with odd indices access the elements striding by 31. For example, thread 0 requests word 0 and thread 1 requests word 32, through the same bank (i.e. bank 0). In this case, these two shared memory accesses are serialized and completed in 2 clock cycles.

There is an argument that bank conflicts might not be the most significant performance bottleneck, since shared memory access is much faster than accessing global memory. Optimizing global memory coalescing access and caching frequently used data in shared memory or registers can bring most of the performance improvements in majority of GPU applications. Also, other optimization mechanisms, such as using texture memory and constant cache for built-in interpolation and read-only data input should be considered in certain type of applications.

This section described the three common memory access patterns: latency-bound, compute-bound and memory-bound patterns. To optimize the performance of a GPU application, it is important to design the parallel mapping that has a high memory access efficiency, such as caching data and wisely use on-chip fast memory storage, avoid thread branching, and configure an optimal thread occupancy.

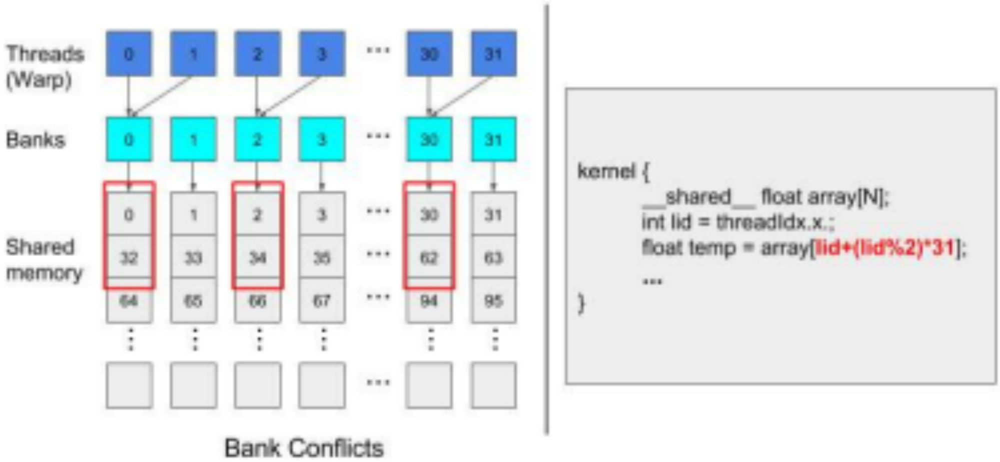


Figure 2.11: A Simple Illustration of Bank Conflicts

2.2 Magnetic Resonance Imaging in Medicine

Neuroscience is the study of structure and function of the human brain. The brain is the least understood and most intricate biological system known, which is made up of tissues that can be roughly categorized into grey matter (GM) and white matter (WM). Grey matter is mostly composed of neuronal cell bodies, and white matter contains most of the long-range projections of the neurons, called axons. The open spaces between the brain and skull are filled with cerebrospinal fluid (CSF). An illustration of a diagram of a slice through a human brain is shown in figure 2.12. It is clear that the GM is on the folded exterior surface of the brain (and subcortical (deep) gray matter which can not be seen from the figure), while the WM is stretched inside and seems to reach every region of the brain. Many studies, including this project, research new methods and algorithms to explore the brain's structure and function, and the relationships between them.

The advance of modern medical imaging technology and tools have been changing the way of exploring human brain from traditionally studying anatomy to non-invasive medical imaging, especially imaging mechanisms like Magnetic Resonance Imaging (MRI). MRI is an imaging technique able to acquire 2-, 3- and 4-dimensional pictures of soft tissue. It is also widely used in neuroscience (i.e. neuroimaging) to image and characterise the GM and WM, due to its ability to image different tissue contrasts by applying different scan sequences. In neuroimaging, a typical process includes three stages: data acquisition, image reconstruction and signal/image analysis. The 'image' in question is not restricted to 2D information, instead, it can also be in 3-dimensional and 4-dimensional forms (i.e. 3D position information plus element-wise angular information, or 3D information across time). In figure 2.13, some common examples and keywords involved in MRI processing pipeline are shown.

In the next section, some of the basics of MR and the mechanisms that are essential for this project are introduced. These include the basic physical theory of MR, an overview of a MR System and the main methods that are used in MRI to compute images from the MR signals.

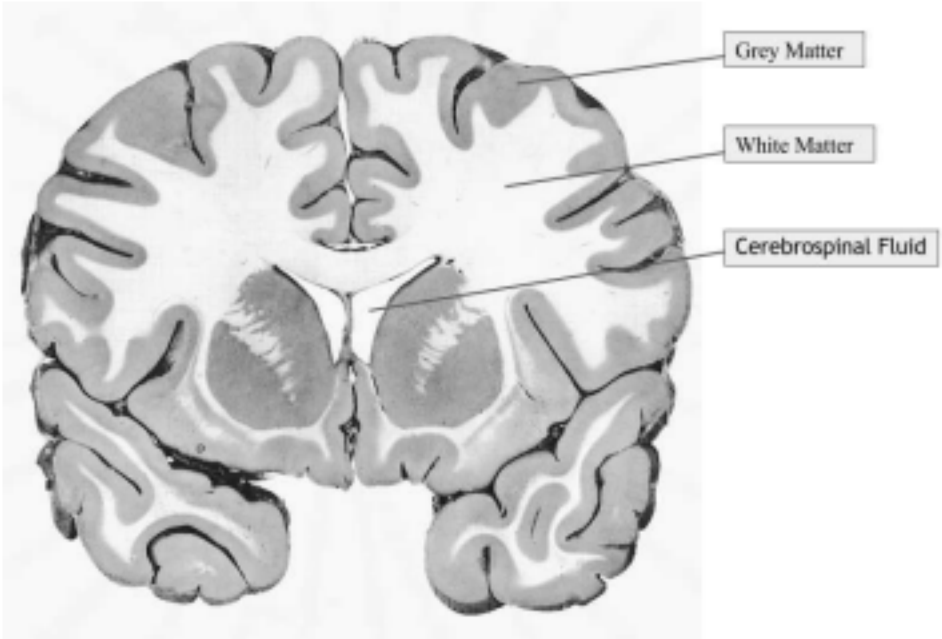


Figure 2.12: A diagram of Human Brain(GM, WM and CSF)

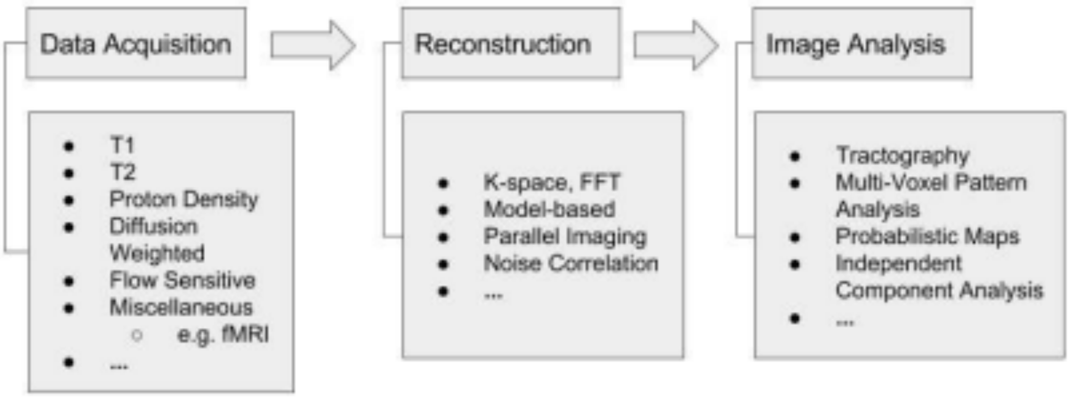


Figure 2.13: An Overview of the MRI Processing Pipeline

2.2.1 Basics of MR

Magnetic Resonance Imaging (MRI) is a technique, based on the theory that "any spinning charged particle creates an electromagnetic field" (Hornak, 1997). An individual proton spins about its axis and generates a magnetic field. The net magnetic field is determined by the directions and strengths of all the magnetic fields created by the spinning protons in the group. The cancellation of all the magnetic fields results in a zero net magnetic field, which is usually regarded as a balanced state. An externally generated and applied magnetic field will alter the states of individual protons and cause them to spin around the direction of the external magnetic field. The angular frequency of precession of the spin can be determined by the Larmor equation:

$$\omega_0 = \gamma B_0 \tag{2.1}$$

where ω_0 is the precession frequency (i.e. Larmor frequency) in MHz, B_0 is the strength of the external magnetic field in Tesla and γ is the gyromagnetic ratio that is measured in MHz/Tesla and is a constant value for a corresponding atom. Hydrogen is the nuclei with unpaired protons that is widely used as the proxy in MR imaging, since it is contained in the water molecules and water is abundant in human body.

A random state is shown on the left in figure 2.14, each proton spins around different orientated axis and produce a balanced state. When an external magnetic field (normally named as B0) is applied to the environment, the protons change their status and spin around the direction of B0 and generate a net magnetic field M. The field components perpendicular to B0 are cancelled each other out for a large ensemble and therefore the orientation of M is along B0. For M to be detectable by a typical MR system, B0 must be sufficiently strong to produce a reasonable signal-noise ratio (SNR). For example, clinical scanners usually have fields of 1.5-3 tesla, some of more advanced research-purpose scanners are equipped with 7 tesla, and fields of 9 tesla and above can also be seen in some animal scanners (because they are smaller bore and the superconducting magnet can be closer to the sample).

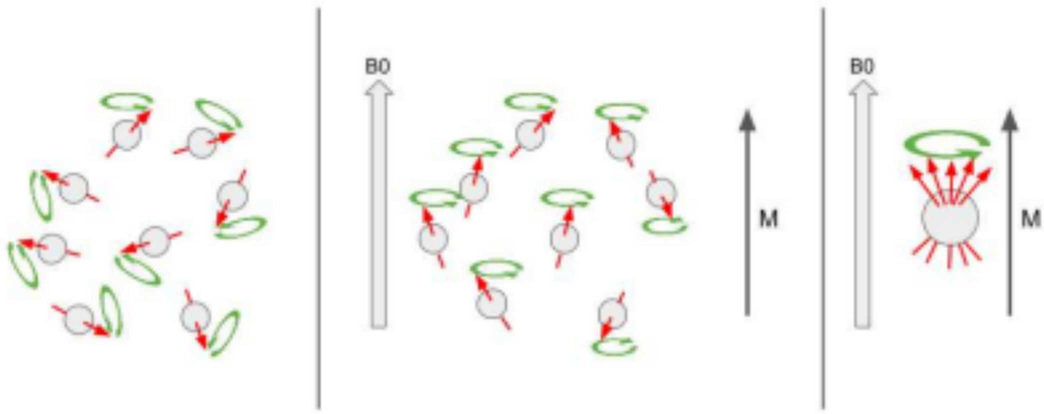


Figure 2.14: Effects with External Magnetic Field B0

Under an external magnetic field, the protons precess around its axis with an angular frequency that is proportional to the strength of the external magnetic field. At the same time, if an electromagnetic radio frequency (RF) pulse is presented, it will introduce changes to the status of the protons. In MR, if the RF applied is at resonance frequency, γ , then the protons absorb the energy and jump to a higher quantum energy level, which is known as 'resonance'. In this case, the orientation of the net magnetic field, M, is rotated by an angle relative to (away from) B0, α , which is determined by the strength and duration of the RF pulse. This process is known as 'excitation', and an illustration can be seen in figure 2.15. When a RF pulse of a specific strength and duration is applied to the system, it causes a 90-degree flip angle of the net magnetic field, M, which then precesses around z-axis on the x-y plane, at the resonance frequency, ω_0, as shown in figure 2.15(c). In practice, this transverse magnetic vector is caused by the phase change of the protons, as all the phases of the protons become synchronized (i.e. in phase).

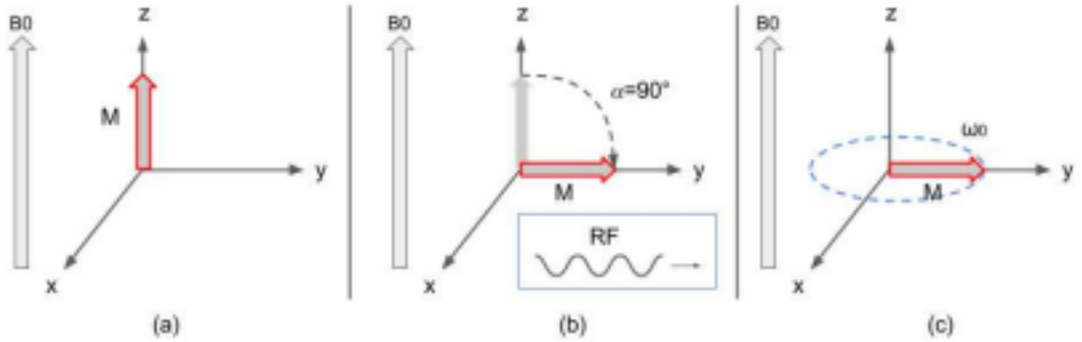


Figure 2.15: Illustration of Excitation

Once the RF is turned off, three things start to happen simultaneously. Firstly, the RF energy absorbed in the excitation process is retransmitted at the resonance frequency, which is detectable, known as the nuclear magnetic resonance (NMR) signal which is proportional to proton density. Secondly, without RF, the net magnetic field (i.e. vector M) begins to return its original state and point to the orientation of the external magnetic field, B0, due to the system tending to recovery to state of thermal equilibrium. The recovery process in this case can be modeled as an exponential function and the time of recovery (i.e. the time taken by the process to recover 63.2% of the longitudinal magnetization) can be measured by a time constant, known as T1, which is one of the key detectable signals used in MR image contrast reconstruction. Thirdly, the excited and in-phase protons begin to dephase when the RF is turned off, which results in an exponential decay of the signal. This decay is also known as spin-spin relaxation, and the time of signal decay is another time constant value, named T2. Moreover, due to the inhomogeneities of B0, the spins dephase much rapidly than T2 in reality and the time is named as T2*. In MR imaging, T1, T2 and T2*are the main signals used to reconstruct into meaningful images.

The T1 relaxation is also called spin-lattice relaxation. When the RF pulse is turned off, the protons jump from the high energy state to low the state and start to realign with the B0 direction (i.e. the magnetization component along the B0 direction is recovering back to M0), as shown in figure 2.16. Besides some of the energy being retransmitted as an RF emission, the rest goes to heating up the surrounding tissue, which is known as the lattice. The spin-lattice relaxation is the process of reestablishing thermal equilibrium, by releasing the energy that are previously absorbed by the protons. The time that the system takes to return to the state of thermal equilibrium is measured as a constant value, T1, that is the time of recovering 63.2% of the longitudinal magnetization after the RF excitation pulse. A tissue's T1 indicates its property of returning to thermal equilibrium, which is measured by the amount of time. Accordingly, measuring T1 of different tissues can generate contrasts that distinguish different tissues from T1-weighted images, based on the recovery of their longitudinal magnetization realignment after an RF pulse. For example, figure 2.16(d) illustrates the T1 relaxation curve of fat and water, and fat has a faster recovery time of T1 than water. Correspondingly, on T1-weighted images, fat will appear brighter than water.

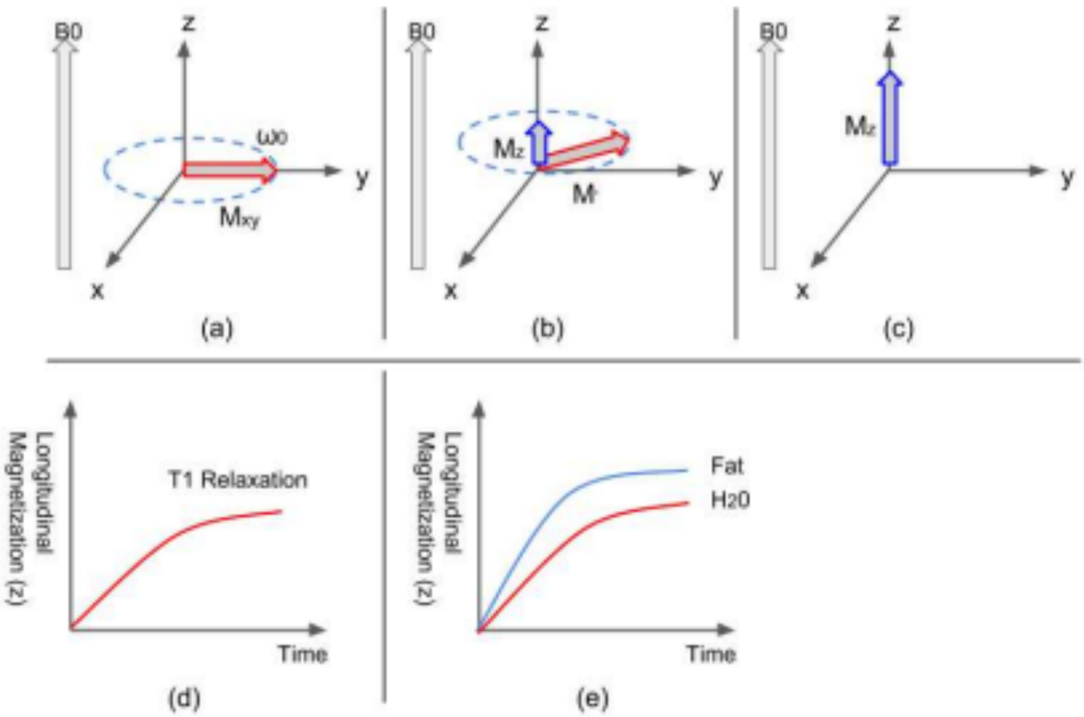


Figure 2.16: T1 Relaxation Illustration and T1 Contrasts

The random interactions between excited spins of protons that causes a cumulative loss of phase result in an overall loss of signal, which is known as spin-spin relaxation and measured by another constant value, T2. The precessing frequency of any particular proton is dependent on the magnetic field that it experiences. When the RF pulse turned off, protons are not only affected by B0, but also experience the magnetic fields generated by other protons nearly. Due to the random motion of the protons, the mutual interactions of the magnetic fields tend to drive the system back to a balanced state, therefore, the spins on the transverse plane are becoming out of phase, and the overall signal of the system is decreasing, as illustrated in figure 2.17.

Similar to T1, the signal decay caused by T2 relaxation can also be modeled as an exponential function. The T2 is the time cost of the transverse signal magnitude that reduces by 63.2% (i.e. 36.8% of the original signal amplitude), after the RF excitation is turned off, as shown in figure 2.17(a). T2 is not related to the strength of external magnetic fields, unlike T1. As a proxy of producing contrasts, T2 is also unique for individual kinds of tissue but also depends on a chemical property, namely magnetic susceptibility.

Magnetic susceptibility measures the degree to which a substance can be magnetized and is commonly categorized in three types: diamagnetic, paramagnetic and ferromagnetic substances. Rapid magnetic changes can be detected in paramagnetic elements, as they are easily magnetized when presented with an external magnetic field, and also quickly demagnetized when the external field is removed. On the other hand, substances that

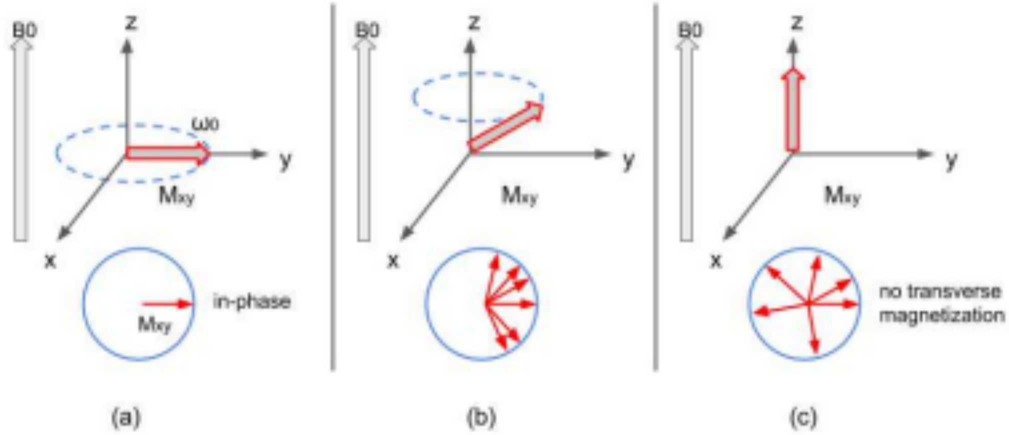


Figure 2.17: Illustration of T2 Relaxation

are of diamagnetic type normally can not be magnetized, whereas ferromagnetic types can remain magnetized after the external magnetic field disappears. In MR imaging, human tissues fall into the category of paramagnetic, and so different kinds of tissues show different magnetic susceptibility, in other words, unique T2 constants, which can be used to generate image contrasts.

The changing magnetic moment of the net magnetic vector, M, is known as the free induction decay (FID), after the RF pulse is turned off. The FID signal, S, can be mathematically modelled as:

$$S = M_{xy} * \sin\alpha * e^{-t/T2} \tag{2.2}$$

where α is the flip angle, T2 is a constant value unique for different types of tissues and M_{xy} is the portion of the longitudinal magnetization that is tipped onto the x-y plane (Weishaupt et al., 2008); and an illustration of the decay is shown in figure 2.18. From this equation, it is clear that the rate of the decay is determined by the constant, T2, and the signal is oscillating at the resonance resonant frequency, ω. Therefore, by acquiring a series of signal samples on the transverse plane, the T2 value can be estimated. It is important to note that the FID decay does not contain any positional information.

Besides the spin-spin relaxation which is entirely a function of the random interactions of spins, the extent of T2 decay is also dependent on external factors, like field inhomogeneity. In reality, the increase of the T2 relaxation time caused by the inhomogeneity of the external magnetic field, B0, can be modelled and captured as T2*. The dephasing of spins measured in T2* is much quicker than T2, as shown in figure 2.19(b). T2* also varies between tissues. For example, figure 2.19(c) shows the T2* relaxation of fat and water, and water has a much longer T2* relaxation, because water molecules move quickly, as a result, there is less field inhomogeneity.

The spin dephasing captured in T2* can be mitigated using a refocusing pulse in the so-called spin-echo sequences, which is designed to allow the measurement of pure T2 signals (i.e. T2-weighted images), rather than T2* weighted. As shown in figure 2.20,

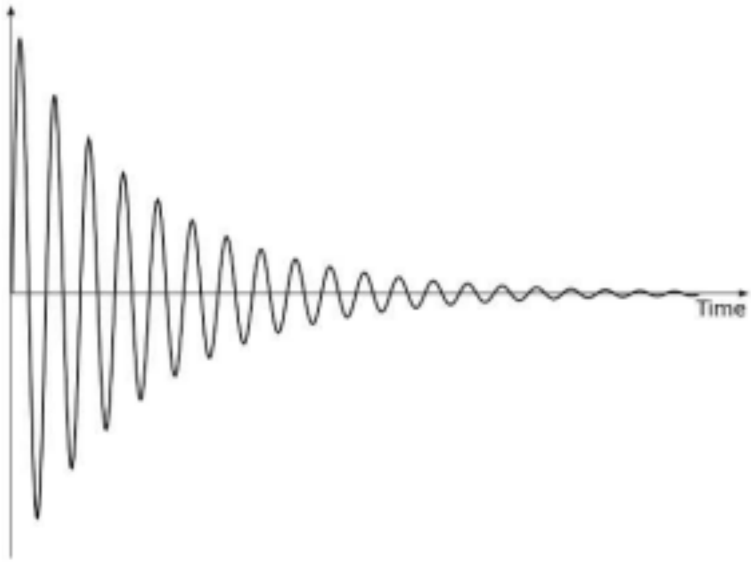


Figure 2.18: Illustration of FID

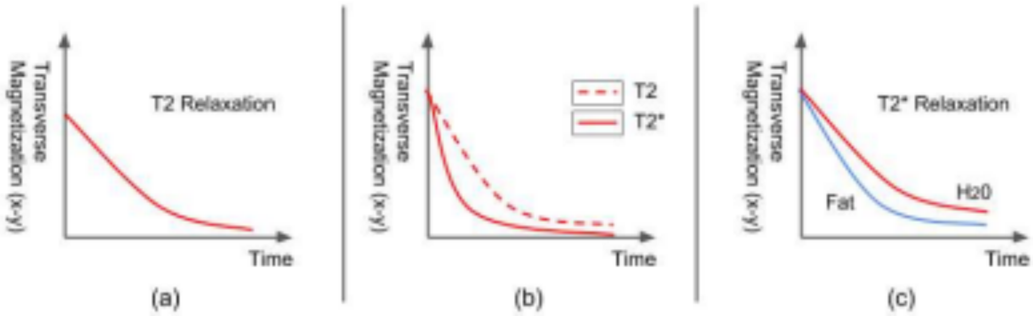


Figure 2.19: Illustration of T2, T2* Relaxation, and T2* Contrasts of Water and Fat

after a 90° RF pulse applied, wait for TE/2 time and apply another 180° RF pulse to flip the phase of the spins. TE refers to "Echo" time, which is the time point that the spins are back in phase and a "readout" is conducted (i.e. acquiring the signal strength). Due to the re-phasing of the spins, a stronger signal on the transverse plane can be detected, which is the primarily T2-weighted signals. From the figure 2.20, it also can be seen that the direction of the transverse magnetization vector is flipped, namely, an echo, which is the reason that the name of "TE" comes from. Furthermore, TR is the repetition time between two consecutive 90° excitations.

During the process of signal attenuation, multiple "echoes" can be collected to estimate the pure T2 decay mechanisms. After a 180° pulse that reverses the T2* dephasing process, the spins re-phase in half of TE; then they immediately begin to de-phase again. To refocus the phases, a second 180° can be applied, generating another echo. When the process is repeated for a few times, the curve of T2 relaxation can be acquired. As shown in figure 2.21, the red curve is the T2 relaxation that we want to measure, and the blue line is the magnetic changes of T2* using spin echo recovery. The idea of refocusing

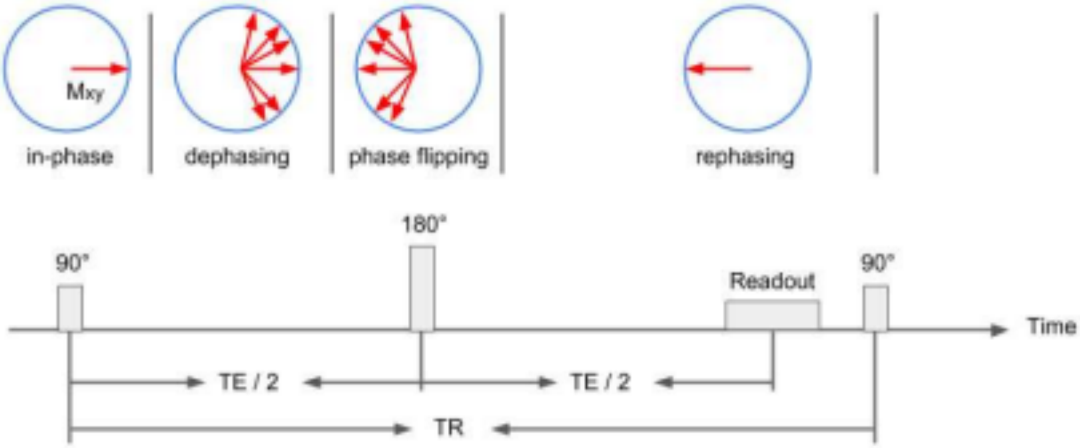


Figure 2.20: Illustration of Spin-Echo Sequence

the phases another pulse (e.g. 180° in spin-echo) are also used in other sequences, like diffusion-weighted imaging that will be introduced later on.

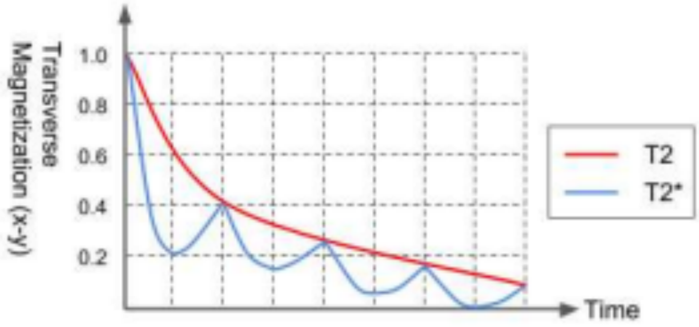


Figure 2.21: T2 Acquisition via Spin Echo Recovery of T2* Signals

2.2.2 MR Scanner System Overview

Most of commercial MR scanner on the market follow a similar system design. Based on the data acquisition process introduced in the previous sections, a common scanner system includes primary functional components, such as a strong, superconducting magnet, various Gradient system, RF pulse generator, image processor and computer system. As shown in figure 2.22(a), a subject is in the scanner and surrounded by the coils: main magnet coil, gradient coils and RF coil. The main magnetic coil generates a strong and (ideally) stable magnetic field, B0, with a strength of 1.5-7T (for human scanners). The most common type of MR magnets uses superconductive materials to generate the magnetic field, due to the physical properties that superconductive materials show nearly zero electrical resistance when they are cooled and kept below their critical temperature. Together with the shimming system (and the shimming coils that adjust the homogeneity of a magnetic field by changing the electrical currents), the static B0 magnetic field can be adjusted to be spatially uniform, even in the presence of other magnetic materials near the installation site. Then, three gradient coils are installed to allow spatial localization, by generating linear magnetic field along the three perpendicular directions. They are driven by the gradient waveform generator and controlled by the computer system. The Radio-Frequency (RF) system controls the RF excitation pulses and signal acquisition. The entire hardware is controlled by a centralized computer system, firmware and software. Its function is to provide operational control of the scanner by the radiographer, and on-the-fly image reconstruction using a dedicated image processor to afford assessment of image quality at the time of scanning.

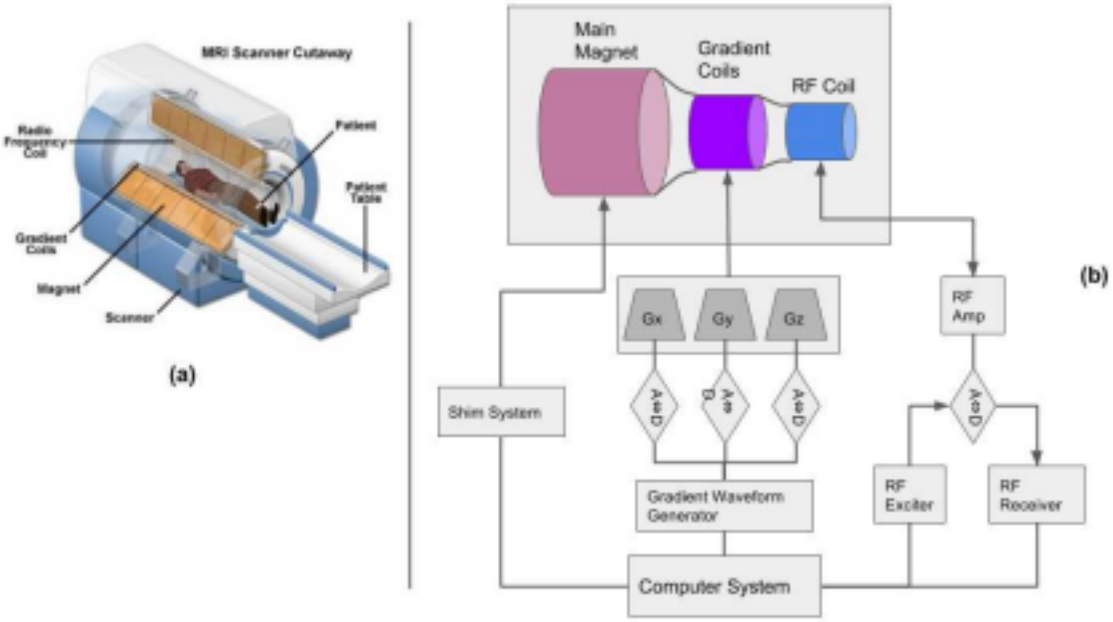


Figure 2.22: Overview of a MR Scanner System (https://pancreaticcanceraction.org)

2.2.3 Spatial Localization and K-space

The signals that generated by a MRI sequence is from all the tissues within the receiver coils of the scanner. It is necessary to have some mechanisms to enable "spatial localization", in order to produce meaningful images. An additional gradient coil is introduced to create a magnetic field whose strength varies linearly depending on the position. As shown in figure 2.23, in addition to the external magnetic field, B0, a gradient coil is installed along the B0 direction (i.e. normally z direction) to create a gradient magnetic field (figure 2.23(b)). According to the Larmor equation, the precession frequency is determined by the strength of the magnetic field. Therefore, in this environment, the overall magnetic field (contributed by both B0 and the gradient) causes the protons spin at different frequency along the z direction. At the isocenter where there is no net magnetic field added by the gradient field, the protons still precess(spin) at the frequency of ω_0. Either side of the isocenter, the gradient field decreases or increases the total magnetic field experienced by protons, and correspondingly they precess slower or faster respectively. This can be exploited to accomplish "slice selection" along the z-axis (in the scanner space). Slice selection is the primary method of spatial localization along the z-axis in MR imaging.

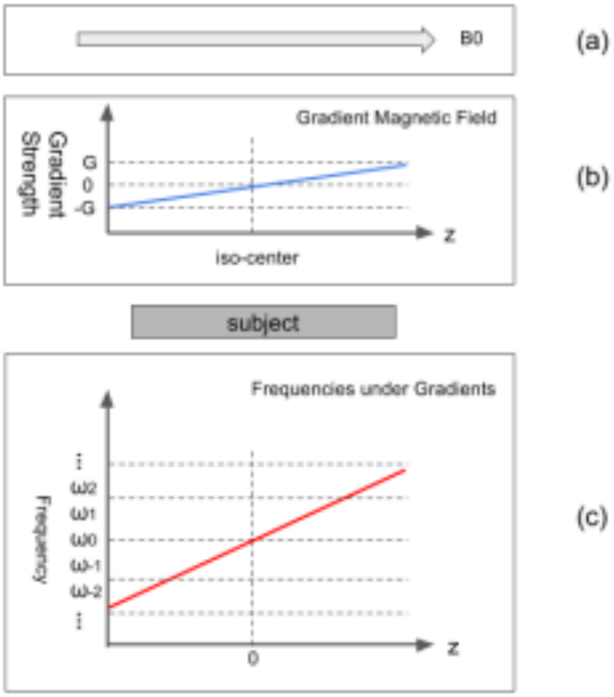


Figure 2.23: Illustration of Gradient Magnetic Field and Its Effects on Spin Frequency

With the additional gradient field, selecting a particular slice to image can be achieved by applying RF pulse that matches the frequency at the desired slice position. According to the theory of magnetic resonance, only the protons that are in the slice will be excited, since only those protons can receive the energy from the pulse and change their energy states accordingly. An illustration can be seen in figure 2.24 - a spin-echo sequence with slice selection, where the RF pulses are at the frequency, ω_{sl}. The T2/T2* signal acquired during the relaxation process only comes from the selected slice. Then the signal from the slice can be reconstructed into a single slice image, with frequency-encoding and phase-encoding that will be introduced in the following content.

With the slice-selection technique, it is possible to acquire the signal from the desired slice position, however, the entire slice is experiencing the same magnetic field and excited protons in the slice spin at the same frequency, which is not sufficient to obtain position information within the slice. By applying another gradient magnetic field that runs along one of the two in-plane dimensions of the slice (i.e. either column or row of an image), protons in each row or column of the slice will experience different strengths of magnetic field and it will cause them to precess at different frequencies. During sampling process, the signal acquired is the information that is composed of a series of signal components, and each of which has a different frequency. This technique is known as frequency encoding.

Using the mathematical technique called Fourier Transformation, the signal with frequency encoding can be decomposed as separate components representing the signal from

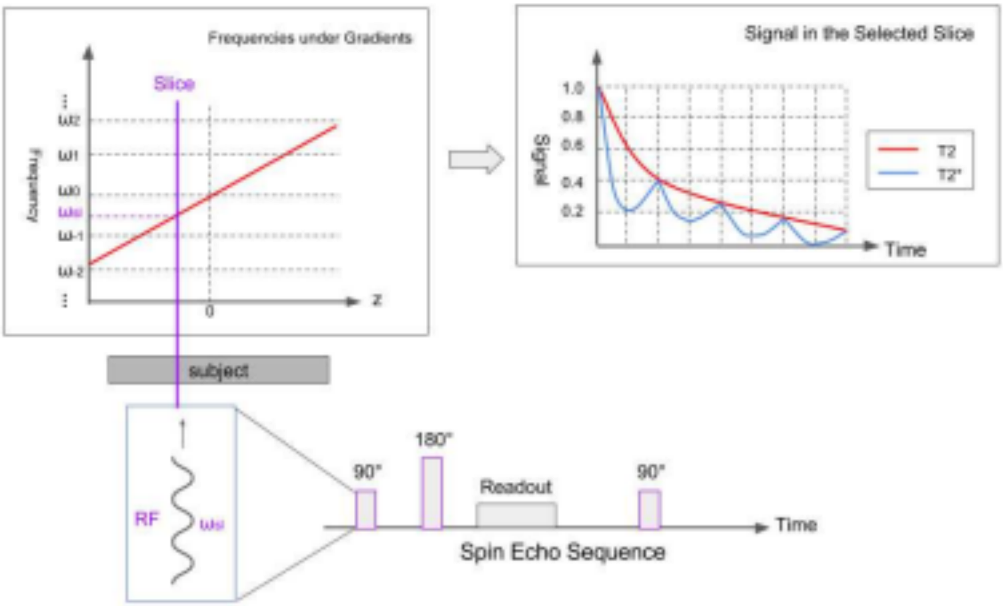


Figure 2.24: Illustration of Slice Selection

the row/column of the selected slice. As shown in figure 2.25, a gradient magnetic field is applied along the row/column direction during readout in an MRI sequence, and the protons experiencing different gradient strength will have frequency shifts. In signal acquisition, a scanner collects the signal from the entire slice (i.e. a combination of the signals under different frequencies). Fourier transformation can decompose the mixed signal into a series of signals that are distinguished by their frequencies.

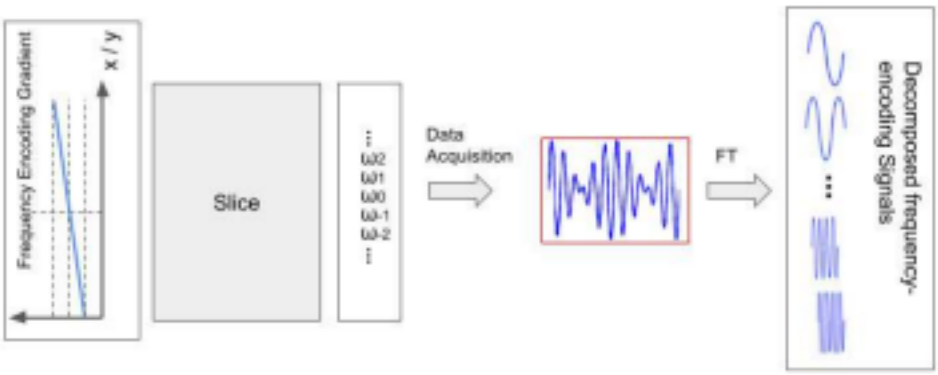


Figure 2.25: Illustration of Decomposition of Frequency Encoding Signal

By applying frequency encoding, a single signal value that describes a slice now is expended to be a 1-dimensional frequency-encoded information. To enable another dimension and be able to represent a 2D image, a second gradient magnetic field is applied to the system, as shown in figure 2.26. The phase-encoding gradient is applied between the slice-selection RF pulse and the frequency-encoding gradient. When the phase-encoding gradient is turned on for a short amount of time, a phase shift is applied to all the protons in the slice and the phase changes vary based on the strength of the phase-encoding magnetic field they are experiencing (i.e. the distance from the isocenter). Then the phase-encoding gradient is turned off, but the phase shift is accumulated and embedded in the signals. However, multiple phase-encoding gradients are required to measure more than one phase for a particular frequency, by repeating the phase-encoding processes with different phase-encoding gradients. The different colored lines in figure 2.26, represent the gradient with different strengths. Then, the measured signals can be fed into Fourier transformation, perform a 2D Fourier transform and obtain the slice images that can be displayed.

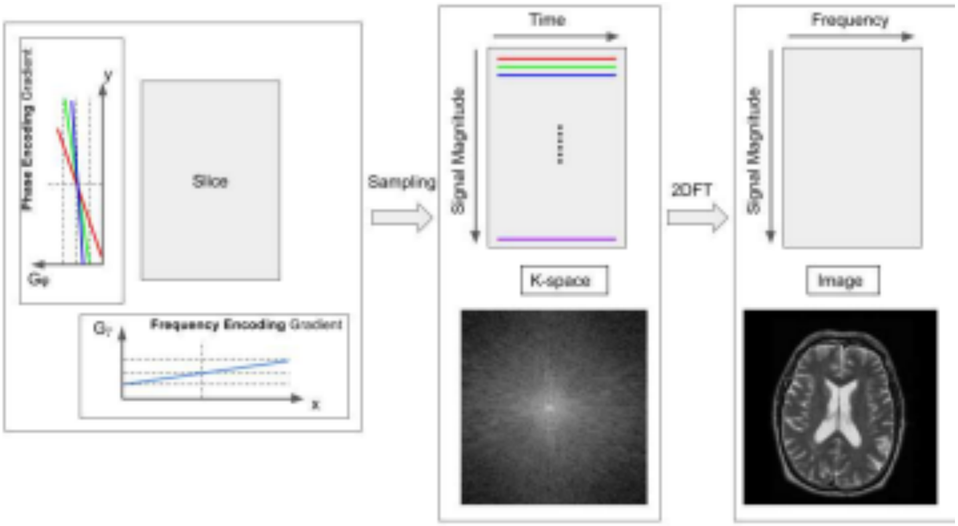


Figure 2.26: Illustration of Phase Encoding and K-space

The matrix that stores the MR signal samples (e.g. x-axis) and each phase-encoding repetitions is known as the K-space, which sometimes is also called the time domain data. The size of a k-space/matrix is determined by the sampling rate of the signal and the number of iterations for phase encoding ,and the matrix size of a K-space is identical to the corresponding image that is transformed by 2D Fourier transformation. As seen in the k-space plot in figure 2.26, it is brighter (more densely sampled) in the center and darker at the periphery.

Figure 2.27 shows the signal decay pattern of T2/T2* in a spin-echo sequence. To collect the echo, i.e. the T2-weighted signal, multiple samples are required before and after the echo. The signal amplitudes increase before the exact point of the echo and decline after reaching the peak, which leads to the k-space "image" having a brighter center. And, the magnitudes of the phase-encoding gradient increase during each phase-encoding repetition, i.e. the the slope of the phase-encoding gradient gets higher. It leads to a decrease of the signal magnitude that is caused by the signal cancellation due to applying increasing frequency- and phase-encoding gradients which can contribute to dephasing of the signal. Figure 2.27 demonstrates that no phase-encoding gradient is applied for the first line (i.e. the middle line of the k-space), and higher slopes of the phase-encoding gradients are applied in the following repetitions, which correspond to the decreasingly detectable signals (i.e. darker lines in the k-space image). Steeper gradients, by the way, correspond to sampling smaller-scale details in the tissue being imaged.

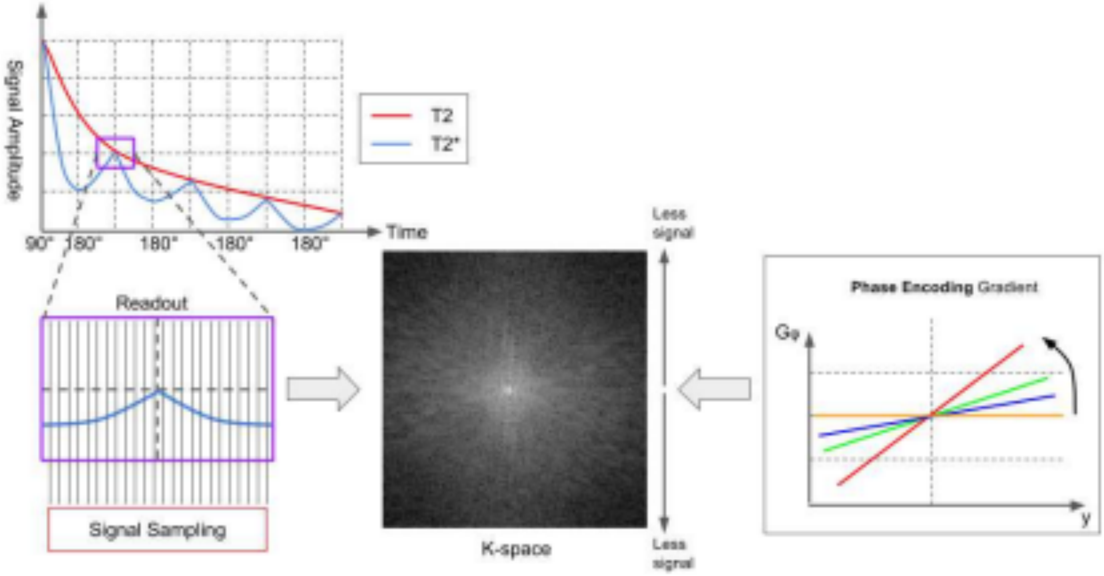


Figure 2.27: Illustration of Sampling Signal of A Spin-echo Sequence and Generating K-space

The periphery of the k-space that has very low signal amplitude, but rich and important amount of information which is encoded in frequency, is related to the spatial resolution. Reducing the k-space matrix size (i.e. excluding part of the periphery of the k-space) will dramatically change the image quality and decrease the image resolution, for example, edge information is significantly blurred. On the other hands, the contrast of an MRI image is mainly contributed by the high amplitude information at the center of a k-space. Excluding the center k-space information, images after Fourier Transformation will remain the spatial resolution or edge definition, but have less contrast and lower overall image brightness, due to loss of high amplitude signals in the center of k-space.

2.3 Diffusion Tractography

Diffusion-weighted MRI is one of the most common ways to map brain anatomical networks that represent the topological configurations of white matter connectivity patterns in the human brain. White matter (WM), along with grey matter and CSF, are the main types of tissue in the brain. WM contains fiber bundles that are composed of axon bodies and serve the function of physically connecting different functional cortical areas. Investigations of structural connectivity with diffusion MRI enable studying the "circuitry" of the brain and its diversity in aging (Sullivan and Pfefferbaum, 2006) and disorders, such as impaired reading ability (Beaulieu et al., 2005) and mild cognitive impairment (Wee et al., 2011).

In the process of studying structural connectivity, the technique that is used to process diffusion MRI data and reconstruct tracts that simulate brain fibers, is known as diffusion tractography. Its pipeline involves: diffusion data acquisition, orientation distribution modelling and generating streamlines that represent the brain fiber bundles. As shown in figure 2.28, the tractography procedure starts with acquiring diffusion-weighted (DW) signals with particular DW MR sequences. The DW signals are measured and collected from multiple orientations, which can be seen as an angular sampling process. Acquisition is followed by fitting a model that represents voxel-wise orientation distributions. Models can be tensor-based in Diffusion Tensor Imaging (DTI) and non-tensor-based in High Angular Resolution Diffusion Imaging (HARDI). Then, a certain class of algorithm is used to track and reconstruct streamlines from a single seed, a Region-of-Interest (ROI), or entire brain. The tractography algorithms track/propagate based on the orientation distribution function of each consecutive voxels in either deterministic or probabilistic ways. Finally, the outputs from tractography algorithms are streamlines that are regarded as the stochastic reconstruction (this means they are statistically representative and not exact one-to-one) of fiber networks of a subject, which can be rendered as 2D/3D images and also used as inputs for post-processing methods.

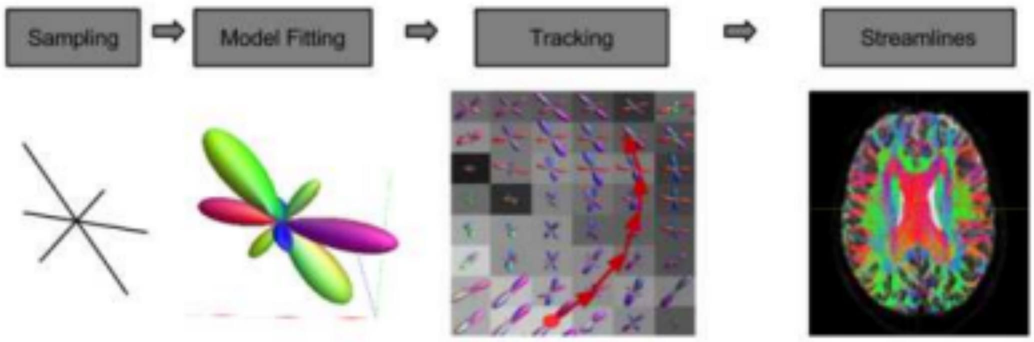


Figure 2.28: Pipeline of Diffusion Tractography

In the following, the basic theory of diffusion-weighted MRI is introduced, followed by the background of modelling mechanisms and tractography algorithms in detail.

2.3.1 Diffusion MRI Data Acquisition

Diffusion-weighted MRI exploits the nature of anisotropic diffusion in organic tissues, such as white matter fibers in the human brain. The concept of (molecular) diffusion is a phenomenon related to Brownian motion/Random motion, which refers to random motions in a free environment. In other words, a particle has an uniform probability distribution of moving in any direction in the space, which is defined as “isotropic” diffusion. On the other hand, the motion of a particle in a physically restricted environment has an ”anisotropic” diffusion, i.e. its movement is constrained by the presence of barriers. An illustration of comparing two different types of diffusion can be seen in figure 2.29, which is a simulation of particle movements. In figure 2.29(a), the particle in the middle of the space can move to any place, given sufficient period of time. In figure 2.29(b), its motion is restricted by the circular-shape barrier.

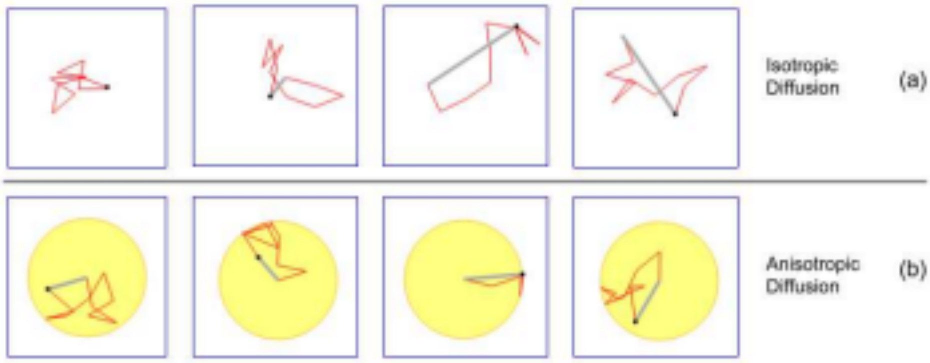


Figure 2.29: Illustration of Isotropic and Anisotropic Diffusion

Brain white matter (myelinated) fibers, principally cell membranes, can be regarded as a restricted environment that that constrains enclosed water molecules to anisotropic diffusion (Sullivan and Pfefferbaum, 2006). Extended from the simulation in figure 2.29 to three-dimensional, figure 2.30 shows an anisotropic diffusion of water in a pipe. The probability of its motion is like an ellipsoid, with larger diffusion coefficients along the direction of the pipe (i.e. x-axis), and relatively smaller diffusion values in the y-z plane. Therefore, in diffusion-weighted MR imaging, the technique measures the diffusion coefficients or the random motion of the water molecules as indications of WM fiber orientations (Tanner, 1979).

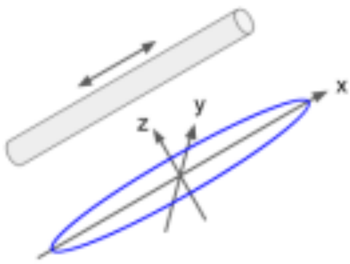


Figure 2.30: Illustration of Anisotropic Diffusion Distribution

The principle of diffusion-weighted imaging is to detect and image signals that characterise the diffusion of water in tissues. Similar to the application of phase-encoding for spatial localization discussed in the previous sections, DW imaging technique uses a gradient pulse, introduces phase shifts, and then applies a reverse pulse with exactly same magnitude. During this process, particles that have moved will introduce phase shifts, while no phase shift will occur for particles that are stationary. Phase shifts lead to the attenuation of signals, and the magnitudes of signal loss is proportional to the underlying diffusion properties. Therefore, by measuring the signal loss, the strength and shape of the diffusion field can be measured.

To make a MRI sequence to be specifically sensitive to diffusion, two diffusion-sensitizing gradient pulses with opposite directions are added. As shown in figure 2.31, it is a diffusion-weighted spin-echo imaging sequence. One diffusion-sensitizing gradient pulse is applied before the 180° RF pulse, which creates additional phase shifts of protons, apart from the phase shifts introduced by the phase-encoding gradient. After the 180° RF, another gradient pulse is applied with the exactly same magnitude, but reversed direction (since the first 180° RF pulse has flipped the precession, therefore, the second diffusion gradient that is applied, has the same direction, but a reversed effect.). The diffusion gradient pulse can be manipulated by controlling the strength of the gradient, G (in Gauss/cm), the duration of the gradient, t (in second), and the interval between the paired diffusion gradient pulses, Δ (in second). The detectable signal loss in DW imaging is proportional to G, t and Δ, and the term b-value (in s/mm^2) is defined as:

$$b = \gamma^2 G^2 t^2 \left(\Delta - t/3\right) \tag{2.3}$$

where γ is the gyromagnetic ratio. Therefore, a large b-value refers to either a stronger gradient or a longer duration, a wider spacing between two pulses, or a combination of all these factors.

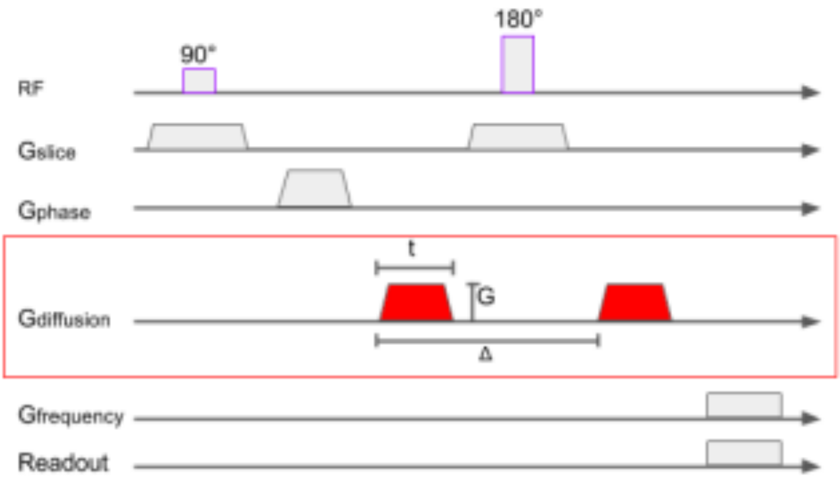


Figure 2.31: Illustration of Diffusion-Weighted Imaging with Spin-Echo Imaging Sequence

The dephasing and rephasing processes driven by the paired diffusion gradients can distinguish stationary and moving protons by measuring the signal loss between diffusion-weighted signal/image and baseline signal/image (i.e. b-value is zero). In some context, the diffusion information that is calculated by comparing DW images with a baseline image (i.e. zero b-value) is defined as the apparent diffusion coefficient (ADC). The term, ADC, often refers to the mean diffusion in a voxel and along a single direction, and the image is also known as ADC map. In reality, b-value of 50 $second/mm^2$ is usually used, instead of zero b-value. Usually, diffusion measurement with larger b-value leads to lower signal-to-noise ratio (SNR).

2.3.2 Diffusion Orientation Distribution Modelling

Above, the concept of diffusion-weighted (DW) magnetic resonance (MR) imaging has been introduced. Using diffusion-weighted sequences can acquire diffusion-sensitizing information of water molecules in tissues that can be processed to yield orientation information on the underlying anisotropic distribution of elongated fibers, such as in the white matter of the brain. Among the methods that model voxel-wise orientations, Diffusion Tensor Imaging (DTI) is a traditional approach that represents orientation information and its uncertainty as a tensor. As MR field strengths increase and scan protocols are improved, the spatial and angular resolution that can be achieved have reached the point where DTI methods are being replaced in favour of non-tensor models, so as to allow for modelling more complex fiber orientations per image voxel. In the next section, DTI and non-tensor-based models will be introduced, and the spherical harmonics representation of diffusion signals will be introduced, as it is the focus of this work.

2.3.2.1 Tensor-based Model

The tensor is a mathematical concept, and an extension of "vector". Both scalars and vectors are tensors in general. Scalars are rank 0 tensors, and vectors are rank 1 tensors. Scalars can be represented with only one magnitude value; vectors need three components (i.e. three magnitude values and three orthogonal unit vectors). In DTI, a rank 2 tensor is used, which can be expressed as a 3x3 matrix. Due to the symmetry property of a DTI tensor, 6 values are sufficient for its representation. A definition of a (possibly DTI) tensor can visualized as an ellipsoid that can be determined by its three orthogonal eigenvectors: v_1, v_2 and v_3. Also, it can be expressed in matrix form:

$$\begin{bmatrix} \lambda_{xx} & \lambda_{xy} & \lambda_{zx} \\ \lambda_{yx} & \lambda_{yy} & \lambda_{yz} \\ \lambda_{zx} & \lambda_{zy} & \lambda_{zz} \end{bmatrix}$$

where at least 6 values are necessary: $\lambda_{xx}, \lambda_{yy}, \lambda_{zz}, \lambda_{xy}(\lambda_{yx}), \lambda_{yz}(\lambda_{zy})$ and $\lambda_{xz}(\lambda_{zx})$.

In diffusion tensor imaging, a tensor representing the diffusion profile within a voxel is derived from the measurements of ADCs. As there are 6 unknown values to be determined, at least 6 diffusion-encoded measurements are required, which are usually along the axes of x, y, z, xy, yz and xz. As shown in figure 2.32, the matrix D contains different ADC values along 9 directions (technically, 6 different values due to the symmetric property of diffusion, i.e. D_{xy} and D_{yx}, D_{xz} and D_{zx}, D_{yz} and D_{zy} are the same). Given 6 diffusion-encoding measurements (excluding b0 image), a tensor can be determined. In real-life applications, more than 6 diffusion-encoding images are acquired, then a tensor can be estimated, using Least Square approach, to reduce the impact of measurement noise (e.g. electronic noise and amplification noise in the coil system). Furthermore, a diffusion tensor can be expressed by eigenvalues and eigenvectors. The former determines the strength of the diffusion, and the eigenvectors represent the corresponding directions in a tensor model.

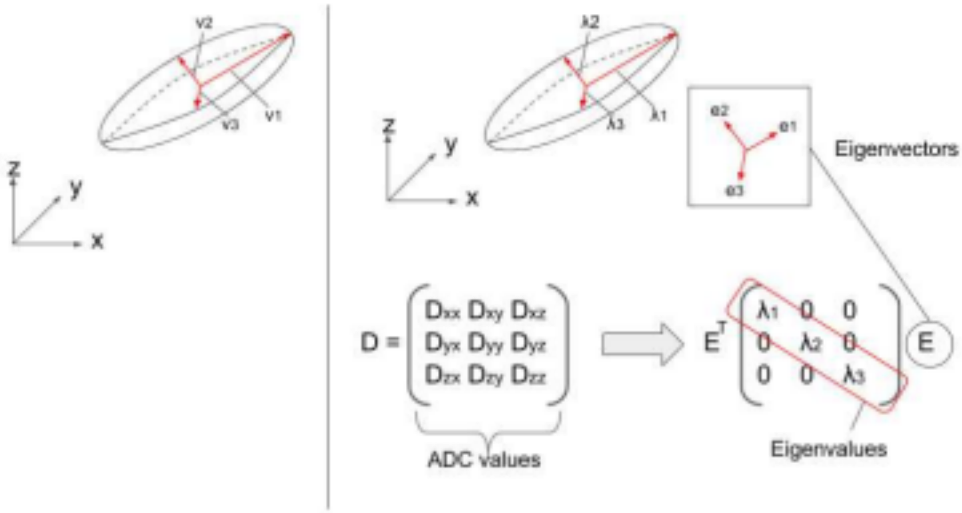


Figure 2.32: Illustration of Diffusion Tensor

The diffusion tensor is capable of capturing major orientation information (usually with one dominant orientation) in a voxel. Figure 2.33(a) shows fiber bundles with a single direction passing through a voxel. Its corresponding diffusion profile can be modeled as an elongated ellipsoid that indicates the dominant orientation that matches the underlying fiber orientation. In the case of multiple directions (figure 2.33(b)), two crossing fiber bundles go through the voxel, whose tensor representation is like a disk that indicates the equally weighted orientations, and, in this case, the diffusion tensor model fails to accurately depict the fiber bundle configurations. The tensor model is simple, and capable of capturing orientation information in white matter regions that contain single predominant fiber bundle within one voxel. A diffusion tensor image can be colour-coded (figure 2.33(c)) and can provide a compact overview of the WM fiber structures.

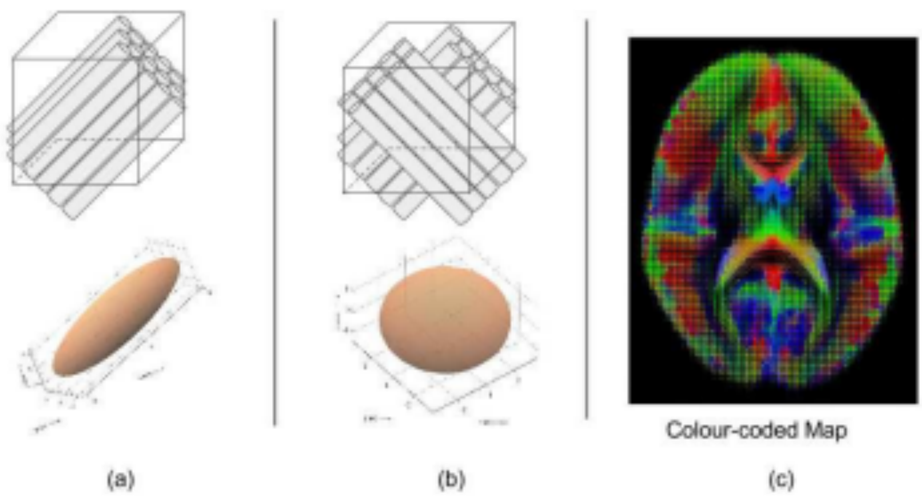


Figure 2.33: Illustration of Diffusion Tensor Representation of Fiber Orientations

2.3.2.2 Constrained Spherical Deconvolution(CSD)

However, the diffusion tensor model is known to be insufficient for representing multiple orientations, which is necessary in the brain WM regions that have crossing, branching and merging fiber bundles (Tournier et al., 2011). The tensor-based model assumes that the voxel-wise diffusion profile can be described by an elliptically-shaped probabilistic density function with at most one dominant direction corresponding to a single fiber population. This can lead to inaccurate results in regions of multiple crossing fibers (Behrens et al., 2007; Assaf and Pasternak, 2008).

To overcome this shortcoming, more advanced techniques like diffusion spectrum imaging (DSI) (Hagmann et al., 2004), Q-ball imaging (Tuch, 2004), Spherical Deconvolution (SD) (Tournier et al., 2007) and other non-tensor models have been developed. Among these methods, the spherical Deconvolution model and its improved version: Constrained Spherical Deconvolution (CSD) have proved to provide robust and accurate representations of complicated fiber structures (Fillard et al., 2011).

In theory, the spherical deconvolution (SD) model assumes that the voxel-wise orientation profile can be represented as a fiber orientation density function (FOD) convolved with an universal response function (Tournier et al., 2004). Convolution is a mathematical operation defined as “the integral of the product of two functions (one of the functions is reversed and shifted)” (ONeil et al., 1963). In the SD model, it describes that convolution is an operation over the unit sphere, of the FOD function, F, with the response function, therefore, the $Diffusion_{voxel}$, D_v, can be written as the formula:

$$D_v\left(\theta,\phi\right) = F\left(\theta,\phi\right) \otimes R\left(\theta\right) \tag{2.4}$$

where θ and ϕ are the elevation and azimuthal angles in spherical coordinates. The response function can be computed from the data by averaging the data from voxels known to share a single dominant orientation; the FOD function is the function that encodes the probability distribution of the orientation and is the field to be derived from diffusion measurements.

Extracting the FOD function, F, that is used to represent the local diffusion orientation model is a deconvolution process (Kazhdan et al., 2003; Atkinson and Han, 2012). It can be described as:

$$F\left(\theta,\phi\right) = D_v\left(\theta,\phi\right) \otimes^{-1} R\left(\theta\right) \tag{2.5}$$

where the symbol $\otimes^{-1}$ stands for a deconvolution operation. Given the FOD function, the amplitudes of any spherical orientations (θ, ϕ) can be calculated. The way to estimate the FOD function is usually using a least square solver, and to iteratively find the optimal coefficients that represent a FOD model with least residuals compared to the raw diffusion signal, $D_v(\theta, \phi)$.

Any functions defined on the sphere in Cartesian space can also be expressed in its corresponding angular frequency domain, known as spherical harmonics domain (Atkinson and Han, 2012). This is analogous to the Fourier transformation between 1D or 2D time and frequency domains. Like the frequency basis functions defined in the Fourier domain, the orthogonal basis functions known as the spherical harmonics are defined as the equation:

$$Y_{l,m}\left(\theta,\phi\right) = N_{l,m} * P_l^{|m|}(\cos\theta) * e^{im\phi} \tag{2.6}$$

where order l determines the the frequency of the basis function, m is the degree on a certain frequency order, $P_l^{|m|}$ are the associated Legendre Polynomials (Liu, 2008), and $N_{l,m}$ are the normalization constants, defined as:

$$N_{l,m} = \sqrt{\frac{(al+1)(l-|m|)!}{4\pi(l+|m|)!}} \tag{2.7}$$

Therefore, the SH basis functions $Y_{l,m}$ are orthonormal. Figure 2.34 shows some example spherical harmonics from order 0 to 3.

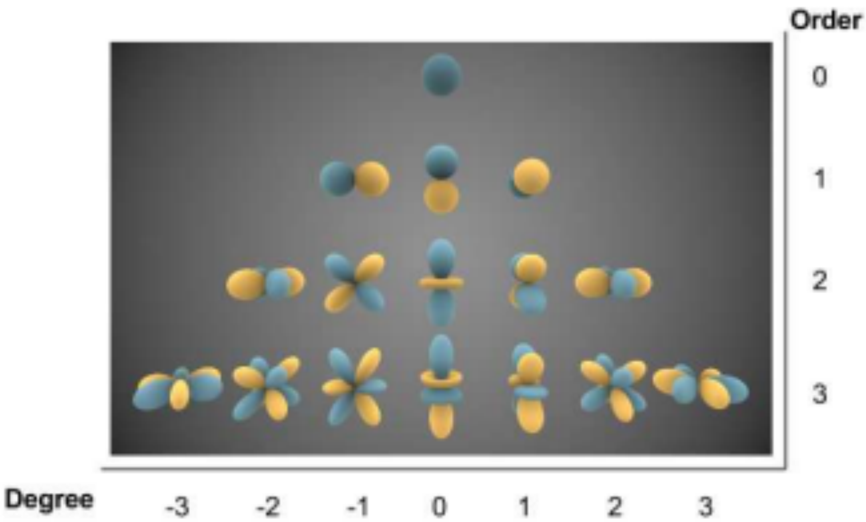


Figure 2.34: Illustration of Spherical Harmonics

Since only real-valued functions are used in application, the real part of the SH basis functions can be extracted as:

$$Y_{l,m} = \begin{cases} \sqrt{2}N_{l,m}\cos(m\phi)P_l^m\cos(\theta) & \text{if } m > 0 \\ N_{l,0}P_l^0(\cos(\theta)) & \text{if } m = 0 \\ \sqrt{2}N_{l,m}\sin(-m\phi)P_l^{-m}\cos(\theta) & \text{if } m < 0 \end{cases} \tag{2.8}$$

In general, a basis set of order l have $(l+1)^2$ functions. However, since diffusion profiles are considered to be symmetric, only even order functions are applied in this case, like $0^{th}, 2^{nd}, 4^{th} order$, and etc. Therefore, a basis set with of order l contains $\frac{(l+1)(l+2)}{2}$ functions/harmonics.

One of the properties of the spherical harmonics domain is that the convolution operation can be computed in the form of matrix multiplication. For example, the convolution in Cartesian space is described as an integral of a kernel/response function, $R(\theta, \phi)$ and another function (i.e. FOD):

$$D_v = \iint_{i,j \in (-inf, inf)} F(\theta, \phi) R(\theta - i, \phi - j) \tag{2.9}$$

where the integral goes over all the possible values of (θ, ϕ) in R. Rather than performing summation or integral of weighted function, FOD, the convolution in the spherical harmonics domain can be done in a simpler form:

$$D_v(\theta, \phi) = R_{l,0}(\theta) Y_{l,m}(\theta, \phi) c_{l,m} \tag{2.10}$$

where $c_{l,m}$ are the coefficients, l and m are the SH order and degree, and $R_{l,0}(\theta, \phi)$ is a symmetric kernel function composed of zero-degree basis (i.e. SH basis with degree of 0, like $C_{0,0}, C_{2,0}, C_{4,0}$ and etc).

Only limited number of diffusion-weighted orientations are measured. In the data acquisition process, a certain number of evenly distributed orientations (θ, ϕ) are measured, which results in a set of diffusion magnitudes,ρ, and their corresponding orientations. The number of diffusion weighted orientations in HARDI typically ranges from 30 and to 200. By measuring diffusion along certain directions, a set of diffusion values in the form of (ρ, θ, ϕ) can be obtained. And, for each measurement, k, yielding the equation:

$$R_{l,0}(\theta_k) Y_{l,m}(\theta_k, \phi_k) c_{l,m} = \rho(\theta_k, \phi_k) = S(\theta_k, \phi_k) \tag{2.11}$$

where ρ_k is the diffusion magnitude measured at angle (θ, ϕ).

Therefore, the corresponding deconvolution operation to estimate the FOD function, given the diffusion signal and an estimated response function, becomes the process of iteratively finding the optimal SH coefficients that have the least residuals. Given the diffusion measurements of multiple orientations, the problem of "deconvolution" is to solve an overdetermined equation system and estimate the optimal SH coefficients $c_{l,m}$. And the FOD function is a linear combination of the spherical harmonic basis, weighted by the corresponding coefficients.

Importantly, using the spherical harmonics domain to represent a function can result in negative regions which are not physically meaningful in the case of modelling a diffusion profile, because the FOD (i.e. the probabilistic density function) has no negative values. Therefore, a constraint is added to the SD model to prevent unrealistic negativity, yielding Constrained Spherical Deconvolution (CSD) (Tournier et al., 2011). To eliminate negative regions, the CSD algorithm projects the voxel-wise signals to the SH domain, randomly samples a certain number of directions (e.g. 300) and sets the amplitude of orientations whose amplitudes are below a particular threshold, to be zero. The process can be mathematically expressed as:

$$R_{l,0}(\theta_k) Y_{l,m}(\theta_t, \phi_t) c_{l,m} = \rho(\theta_t, \phi_t) = 0 \tag{2.12}$$

where t is the index of the orientation whose amplitudes are set to be 0s.

From the perspective of linear algebra, adding constraints means appending these restriction functions (Eq. 2.12) to the equation system to be solve. More specifically, it is an overdetermined system, which has more equations (i.e. rows) than the unknowns (i.e. columns). This type of problem is usually solved through "Least Square" approach to obtain an approximate solution that have the minimal residuals, and known as the problem of Ax=b:

$$R_{l,0}(\theta_{k,t})Y_{l,m}(\theta_{k,t},\phi_t)c_{l,m} = \rho(\theta_{k,t},\phi_{k,t}) \Rightarrow Ax = b \tag{2.13}$$

where A is the matrix constructed by the multiplication of kernel function and the Spherical Harmonic basis functions defined on the measured directions and restricted directions, b is the measured signals followed by the 0 values indicating the constrained directions, k is the index of orientation whose amplitudes are non-zero, and t is the index of constrained orientations (i.e. zero-amplitudes).

An overdetermined system that depicts the problem of CSD can be solved using Linear Algebra approach. The equation system can be expressed as:

$$\begin{bmatrix} M \\ N \end{bmatrix} * \begin{bmatrix} F \end{bmatrix} = \begin{bmatrix} S \\ 0 \end{bmatrix} \tag{2.14}$$

where M and N are the spherical harmonics transformation matrix of the diffusion-weighted orientations and constrained orientation whose amplitudes are set to be zero, S are the measured diffusion signals, and F are the unknown SH coefficients to be estimated. Directly solving the inversion of matrix $\left[\frac{M}{N}\right]$ is difficult, instead, a common way is to use QR decomposition and solve the inversion of the Q and R matrix separately (Lawson and Hanson, 1974), like:

$$F = R^{-1}Q^T \begin{bmatrix} S \\ 0 \end{bmatrix} \tag{2.15}$$

where Q^T is equal to the inversion of Q matrix Q^{-1} because Q is an orthonormal matrix, and R^{-1} is the inverted matrix of the upper triangle matrix, R. And the coefficients, F, can be obtained by solving the following equation using the conventional back-substitution method:

$$RF = Q^T \begin{bmatrix} S \\ 0 \end{bmatrix} \tag{2.16}$$

After a few iterations, the coefficients will converge, and the optimal solution is then used as the voxel-wise representations of the orientation distribution. To conclude, the overall workflow of solving the CSD problem is shown in figure 2.35, it starts by constraining the orientations that have negative amplitudes and iteratively solves the equation system in the form of "Ax=b" until an optimal solution with least residuals is obtained.

After explaining the CSD model in detail, it is useful to visually compare the tensor and CSD representations of the same voxel. As shown in figure 2.36, it is clear that the CSD model on the right can represent much more detailed information of the underlying

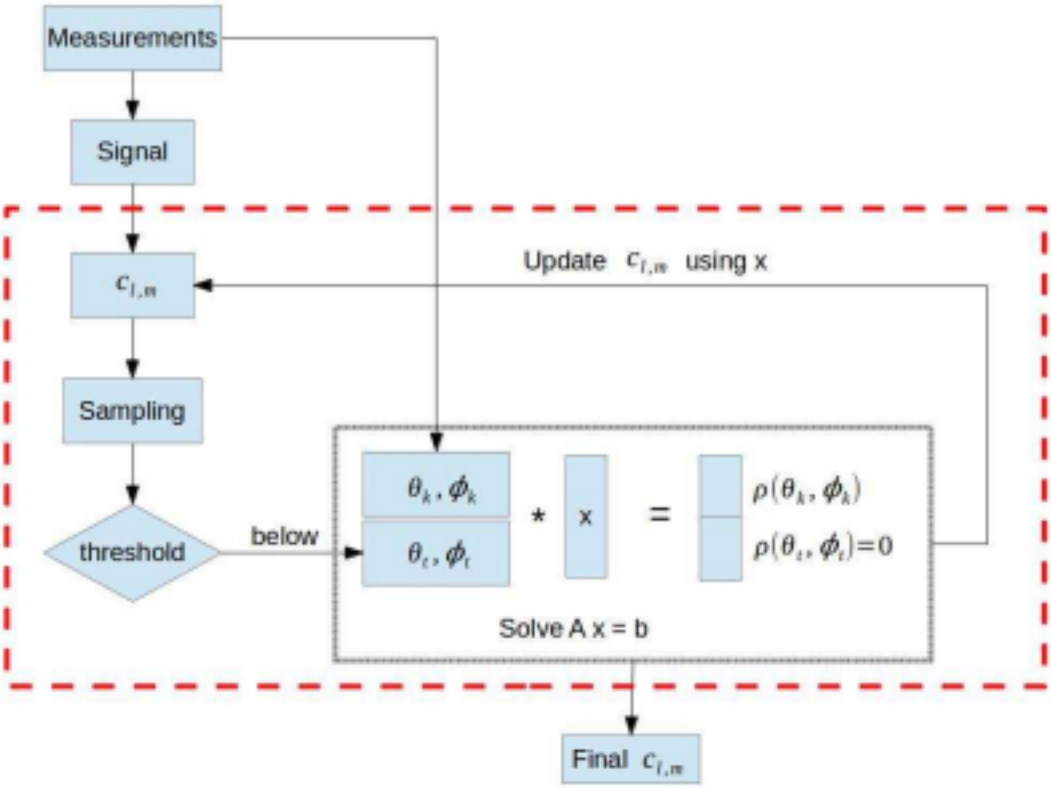


Figure 2.35: Workflow of Solving the CSD Problem

fiber structures. Instead of showing only one dominant direction in the tensor model (on the left of figure 2.36), CSD technique is able to model multiple orientations, which can be fiber crossing, branching or merging. The quality of modeling of the voxel-wise orientation distributions has a critical effect on the reconstruction of the streamlines (i.e. fibers) in the tractography algorithms.

Figure 2.36: The Comparison between Tensor and CSD Representations

2.3.3 Tractography

Tractography is the class of algorithm that reconstructs streamlines, which represent the fibers bundles of the WM network, based on the voxel-wise orientation information.The output of tractography, the "connectome", is a description of the structural connectivity off the spatially distributed brain cortical regions. The concept of "streamlines" does not refer to actual brain WM fibers or bundles of axons, instead, it is in general a stochastic proxy for the WM network. In neuroscience studies, diffusion tractography is predominantly used as a 3D modelling technique and input data for further statistical analysis.

The standard assumptions of tractography are that (i) fibers only exist in WM and connect regions of GM; (ii) fibers are generally smooth (i.e. low curvature). These are based on the post mortem studies of the brain anatomies, and inform the choice of parameters in tractography, such as curvature level and termination conditions. A typical tracking process starts from a position (i.e. a seed) in the brain, is propagated by some step size, and ends based on some termination conditions (Savadjiev et al., 2008). The start point of a reconstructed streamline is usually chosen from a priori interested position in the brain or a region (i.e. Region of Interest (ROI)), which typically is a cortical area in the GM. The rules of propagation vary, according to the model of the voxel-wise orientation information, and the tractography algorithm. The two broad categories of tractography algorithms are deterministic and probabilistic methods. Both are based on line propagation (i.e. generation of streamlines from seeds), but probabilistic tractography presents more information about the reliability of the reconstructed streamlines, compared to deterministic tractography, because it includes estimations of confidence for generated connections (Clatworthy et al., 2010).

2.3.3.1 Deterministic Tractography

Deterministic tractography is the traditional method for modelling and visualizing brain WM fibers, and its major property is that "repeating the tracking process from the same set of seeds reproduced the same result" (Alexander, 2010). In most of cases, deterministic tractography is used with a tensor-based voxel-wise orientation model. The tracking process starts from a chosen seed, and steps to the next position that is closest to the current direction, i.e. the next position is determined by the principal eigenvector of the interpolated diffusion tensor at that location. Interpolations in deterministic tractography can either be conducted using simple "nearest neighbour" approach or more advanced bilinear and/or trilinear interpolation techniques. Additional constraint on the angle of curvature to prevent the tract from sharp turns; and the tracking process is restricted in the whole brain region or specific ROIs by using image masks.

An illustration of deterministic tracking process is shown in figure 2.37, the seed is the start of any tracking processes; each voxel has a tensor representation of its orientation profile; and a streamline is a collection of positions or a train of vectors that depict one possible reconstructed fiber bundle. For any given position, an interpolation method is used to estimate the diffusion tensor of that position, based on the tensors in nearby voxels. Then, the propagation direction is adjusted, according to the principal vector of the interpolated tensor. Tensor-based deterministic tractography can successfully map most of the fiber structure in the regions that do not contain crossing fibers (Tournier et al., 2011; Assaf and Pasternak, 2008).

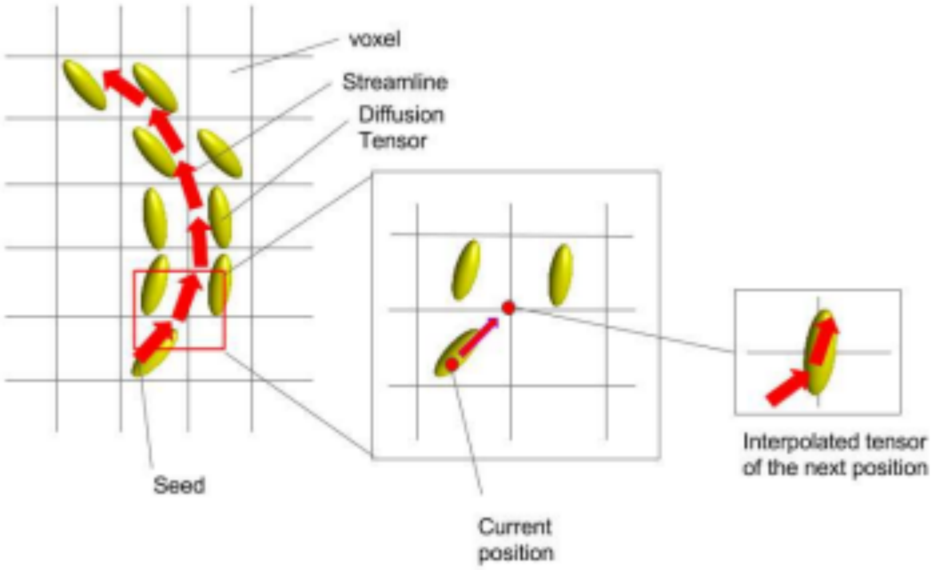


Figure 2.37: Illustration of Deterministic Tractography

2.3.3.2 Probabilistic Tractography

Uncertainties introduced by noise in the diffusion signals limits on the representation of the voxel-wise orientation distribution can lead to unexpected results in deterministic tractography. In the process of propagating streamline, even small deviations can result in large differences in trajectories in total, when using deterministic tractography. Conversely, probabilistic algorithms have proven to be more reliable approaches, as they also encode the uncertainty and provide an estimate of the precision of reconstructed streamlines (Behrens et al., 2003). In other words, the output from probabilistic approaches also contains the information regarding the likelihood of a connection. This enables mapping the WM connectivity, while preserving probabilistic relations between spatially different cortical regions.

The basic idea of probabilistic tractography is measuring the certainty of a generated streamline by repeating the tracking process from the same seed/region for multiple times, and the next step for a particular track is randomly chosen. Unlike deterministic tractography where the tracking process follows the principal/main orientation in the voxels, probabilistic algorithms choose many random positions within a cone whose axis is the current direction of the track, compute the orientation probability, and propagate into those positions that are more likely (i.e. the probability of the orientation is above certain threshold). In probabilistic approaches, the tensor or non-tensor models in a given position are used as the fiber orientation probabilistic density function (PDF), which describes the likelihood that the fiber goes into each direction at that position.

2.3.3.3 CSD-based Probabilistic Tractography

Probabilistic tractography is more commonly used with non-tensor models (e.g. CSD) in order to be more robust to uncertainties in the measured signals and in the models themselves. In this work, we focus on the CSD model, as in typical circumstances it is more robust to noise than other available models (Fillard et al., 2011). Given the voxel-wise fiber orientation distribution (FOD) function, the "greedy" tractography starts with a seed position within a brain mask and proceeds by sampling the FOD in random directions and picking the next direction based on some criteria. For each propagation step, the FOD amplitude of the orientation (θ, ϕ) that points from the current position to the next random position within the propagation region is examined. A valid propagation region is usually defined as a cone around the current direction. The amplitude value can be calculated by the formula:

$$A = \sum_{l,m} c_{l,m} * Y_{l,m}(\theta, \phi) \tag{2.17}$$

where A refers to the FOD amplitudes for a particular direction (θ, ϕ), and $c_{l,m}$ are the spherical harmonics coefficients of the position (i.e. the SH representation of the fiber probability density function). The first position and direction that pass the criteria are accepted as the next position on the tract. The process repeats until a streamline is terminated when either the tracking reaches the image mask boundaries or there is a failure to find the next possible position after certain number of trials (e.g. 50 trials).

Figure 2.38 shows a probabilistic tracking process that starts from the same seed position, and yields different streamlines in multiple iterations. Any random position along the current direction is chosen as long as it exceeds the pre-defined threshold in terms of the FOD amplitudes. The algorithm is capable of capturing fiber crossing, e.g. branching, which is shown in the area that is highlighted in green. By looking at the orientation distribution function on the right, it is clear that the voxel contains crossing fiber information, and multiple directions can be accepted as the next possible positions, which leads to that streamline branching from this point, can be reconstructed, while certain restrictions need to obey, such as a forwards cone of a predefined and constant opening angle works to prevent a fiber going in one of those principal directions from bending sharply to join the crossing bundle.

2.4 Functional MRI Analysis

As well as being used to study structural connectivity (i.e. the anatomical network of WM studied with diffusion tractography and related algorithms), MR imaging can also be applied to probe functional connectivity. Functional connectivity refers to temporal relationships and associations between spatially separate neural regions (He and Evans, 2010). In other words, two cortical regions are said to be functionally connected when their activities are correlated over time. Generally, structural and functional connectivity are complementary description of the brain, and using both modalities can help us better understanding the biological operation of the brain (Damoiseaux and Greicius, 2009). Among the functional connectivity studies, researchers are becoming more interested into neurofeedback techniques, which are widely applied in medical neuro-therapies, such as dealing with pain (Jensen et al., 2007), depression (Hammond, 2005), and insomnia (Arns

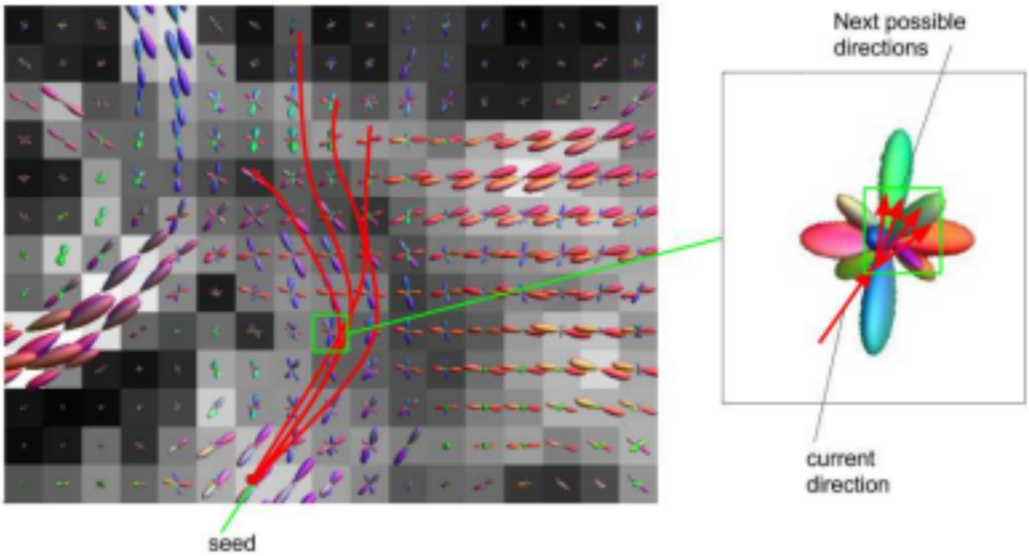


Figure 2.38: Illustration of CSD-based Probabilistic Tractography

and Kenemans, 2014), and also non-medical conditions, like enhancing short-term memory (Nan et al., 2012) and musical performance (Egner and Gruzelier, 2003). In this section, the basics of functional MRI and its related analysis pipeline will be introduced, followed by the background to neurofeedback and the challenges associated with achieving real-time functional MRI for neurofeedback applications.

2.4.1 Basics of fMRI

The brain is a functionally sub-specialized system, wherein the functionalities of the brain are spatially located on different parts of the cortical areas. A sketch of the functional areas in the human brain can be seen in figure 2.39, which shows a coarse-grained illustration of the cortical regions that deal with the high-level functions. For example, performing a simple finger tapping exercise involves a specific region in the brain (in this case precentral gyrus at the frontal lobe primary motor cortex highlighted in figure 2.39) to coordinate the finger's movement. When performing the operation, the neurons at that area experience increased neural activation, which requires increased local uptake of metabolic substances, especially glucose and oxygen to provide sufficient energy. Oxygen is transported to the site by hemoglobin in the blood.

The increase of the metabolic demand is met by a vascular response, also known as hemodynamic response (i.e. a dilation of the local blood supply vessels). In other words, at that point in time, there will be increase in the blood supply to the particular area of the brain, compared to other regions. Before neuron activation occur, oxygenated hemoglobin is diamagnetic; during activation, tissues strip off the oxygen molecules from the hemoglobin in the blood supply, which leads to an increase of the ratio of deoxygenated hemoglobin to oxygenated hemoglobin. The (deoxygenated) hemoglobin become paramagnetic. The oxygenated and deoxygenated hemoglobin have different effects on the MR

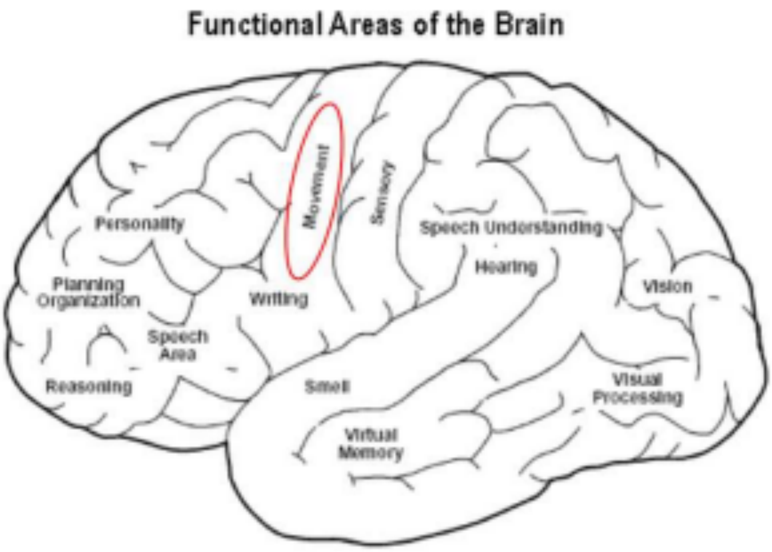


Figure 2.39: Rough Idea of the functional area in the brain (Semrud-Clikeman, 2016)

signals that can be detected. Usually, the deoxygenated hemoglobin leads to a decline in the MR signal due to its high magnetic susceptibility.

In the scenario of increasing blood supply to a local region, the concentration of the oxygenated and deoxygenated hemoglobin varies over time. A simplified illustration of the so-called hemodynamic response can be seen in figure 2.40. There is some baseline concentration when the subject is at rest or the region is not activated. At the point of time when the metabolic demand rises due to neural activities at that location, the tissue starts to extract increasing amounts of oxygen from the bloodstream, which leads to a sudden decline in the ratio of oxygenated to deoxygenated hemoglobin (stage ①). In a very short period of time, the blood supply to this location rises, and then the ratio increases steeply. The response surpasses the amount the initial baseline and stops when the neural activity ceases (stage ②). Then, the ratio returns to the baseline at a relatively slow speed (i.e. usually around several seconds) (stage ③).

The MR signal changes as a function of the ratio of the oxygenated to deoxygenated hemoglobin. When the tissue extract oxygen from the bloodstream, there are more deoxygenated hemoglobin which causes the MR signal intensity to decline; After a short period of time, the blood supply increases and and more oxygenated hemoglobin conveyed to this location, which results in the signal intensity to rise again. When an imaging technique (e.g. functional MRI) that is sensitive to this effect, is applied, this change can be detected in signal.

If a slice of the brain or several slices of the brain is/are imaged, this type of signal changes (i.e. the curve of hemodynamic response) can be essentially measured for each voxel in the brain. When simultaneously looking into multiple areas over time, the regions that related to the particular task/stimulus show neural activity, while the signal in the other unrelated areas remaining at the baseline level. For the voxels that match a task/stimulus (e.g. finger tapping), an image of encoding the statistical probability can

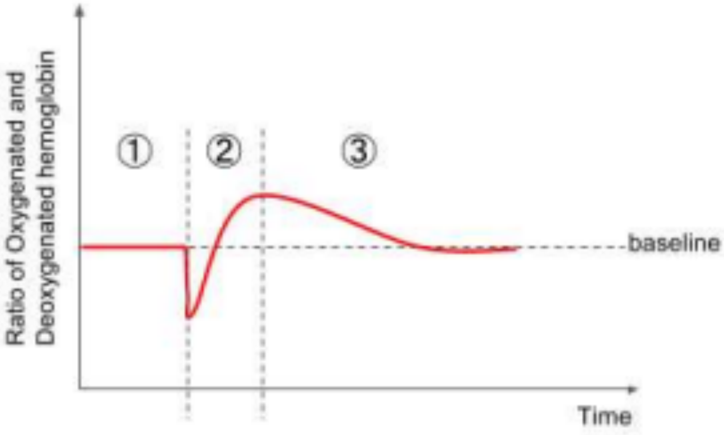


Figure 2.40: Ratio of Oxygenated and Deoxygenated Hemoglobin

be generated to locate the region in the brain that are correlated with temporal pattern of the task. In figure 2.41, it shows a neural activation map of finger tapping (Moritz et al., 2000), where the color-coded area is the activation area related to finger tapping. Furthermore, after the onset of the stimulus, the signal rise due to additional blood supply can be detected after a period of time, and the signal magnitude of the difference compared to the baseline is small, e.g. 2-3%, which is also proportional to the strength of the external magnetic field.

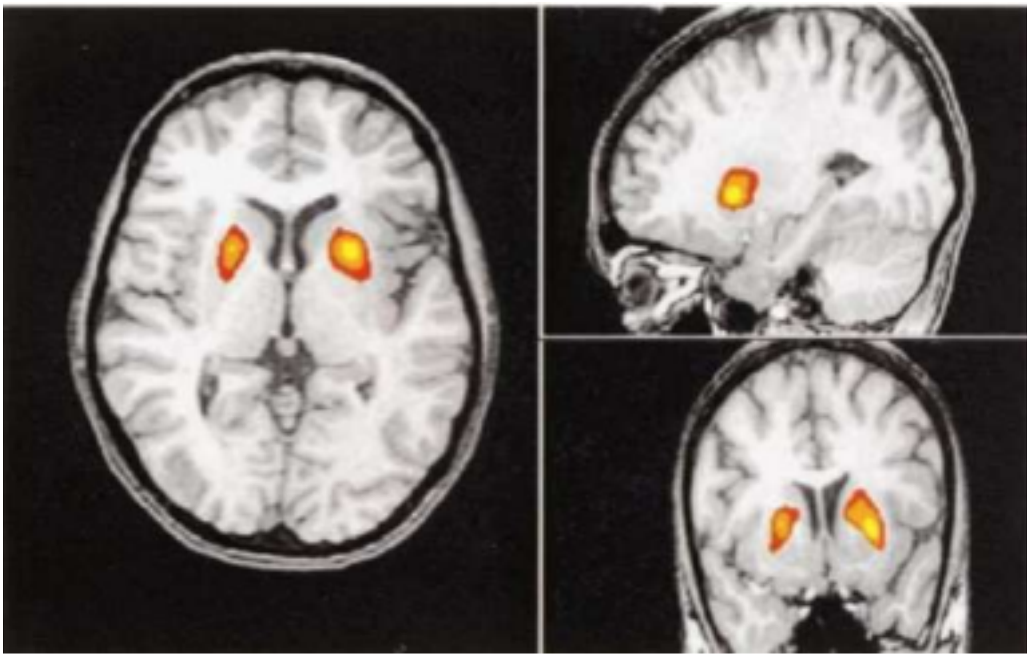


Figure 2.41: Finger tapping fMRI image illustration (Moritz et al., 2000)

2.4.2 Functional MRI Data Acquisition

The concentration of the oxygenated and deoxygenated hemoglobin and its corresponding signal intensity changes are the basis of functional MR imaging or fMRI, also known as the Blood-Oxygen-Level-Dependent (BOLD) contrast imaging. BOLD contrast was first demonstrated in 1990 (Ogawa et al., 1990). Its basic assumption is that the magnetic properties (e.g. paramagnetic and diamagnetic) of the hemoglobin depend on the amount of oxygen that it carries. In other words, an activated region of the brain needs more oxygen, and the corresponding BOLD level increases which affects the T2* of the brain tissues in fMRI measures. The fMRI contrast is generated based on the difference in T2* relative to surrounding tissues. The fMRI signals are measured for a certain period of time and the activation patterns for each voxel or ROIs during the measured period are formed.

Functional MRI signals are inherently weak (i.e. small signal magnitude difference) and transient, and therefore demand a fast imaging sequence. Generally, an echo-planar imaging (EPI) sequence is used to acquire functional data. In EPI acquisition, a complete K-space is formed from a single excitation (i.e. single-shot). This enables rapid signal acquisition and therefore a higher temporal resolution at the expense of spatial resolution. Therefore, EPI is commonly used in cardiac imaging, perfusion imaging and functional imaging.

Echo-planar imaging is a pulse sequence that performs a trick in spatial encoding, with multiple echoes of different phase encoding steps, and rephasing gradients instead of repetition of excitation following RF pulses (in spin-echo imaging). Figure 2.42 illustrates a typical EPI sequence in detail. It uses rapidly reversing frequency encoding gradients and short phase encoding with high magnitudes, and samples the signal at the end of each oscillation/switching. Each readout (highlighted in red in figure 2.42) fills one line in K-space. In single-shot EPI, the entire range of phase encoding steps (e.g. 128 phases encodings) can be acquired within one TR. Therefore, provided the receiver in the scanner has a broad bandwidth and is capable of sampling signals at a high rate, EPI enables the capture of a large amount of information in very short period of time, i.e. it allows acquisitions of fMRI data with high temporal resolution.

2.4.3 Basic FMRI Data Analysis

The purpose of functional MRI data analysis, also known as functional connectivity analysis of neuroimaging data, is to study the relationships between activation signals recorded in spatially distinct regions in the brain, based on their temporal correlations. The so-called functional connectivity is usually represented as a network of nodes and edges in a graph, where voxels or ROIs are the nodes and connections between them indicate the level of correlation or similarities in their response. An example of functional connectivity can be seen in figure 2.43, which depicts a complex configuration of the functional network across the entire brain (Bullmore and Sporns, 2009).

The process of fMRI data analysis involves data acquisition, preprocessing, data analysis and finally, the computation of functional connectivity measurements. A brief overall

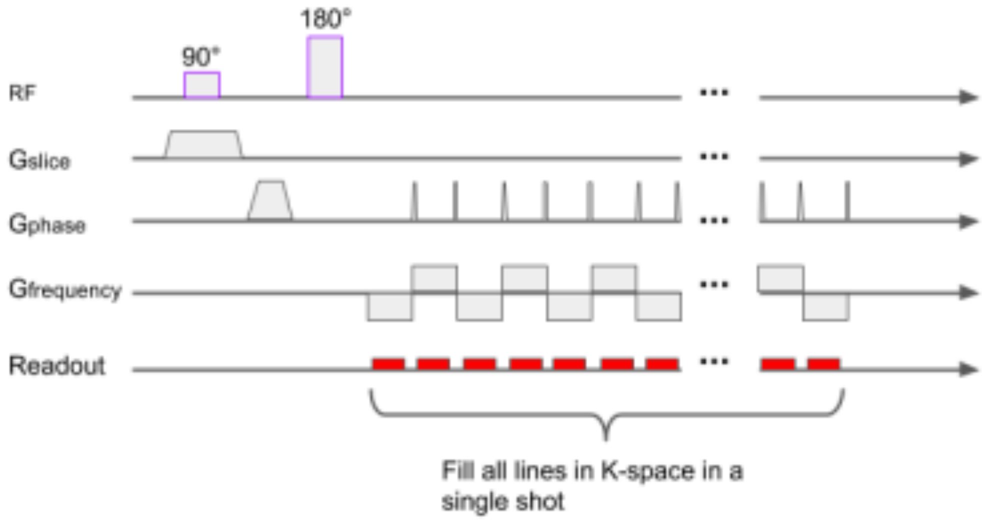


Figure 2.42: Illustration of EPI Sequence

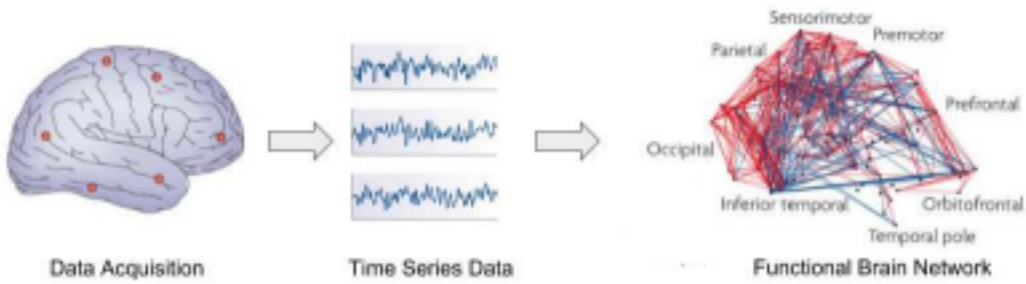


Figure 2.43: Example of Functional Connectivity (Bullmore and Sporns, 2009)

of the fMRI analysis pipeline can be seen in figure 2.44. Raw fMRI time series data is acquired with fast MR imaging techniques, like EPI, followed by a series of pre-processing steps: (i) realignment and motion correlation to make sure that the functional data can be correctly registered with atlases or structure images; (ii) smoothing, filtering and normalization to suppress noise and known imaging artifacts. The analysis of pre-processed functional data falls into two parts: modeling the signals and producing regions of activations. In terms of modeling the functional signal, and segmenting activation brain regions over time, there are mainly three concepts to be mentioned: signal modeling, network-based segmentation and more advanced data-driven segmentation techniques.

2.4.3.1 General Linear Model

The most commonly used model-driven approach is based on General Linear Model (GLM) (Friston et al., 1994; Beckmann et al., 2003), which assumes that the observed signal is a linear combination of some known dependent variables (i.e. predictors variables). In other works, to model the signal, a series of coefficients are estimated as the

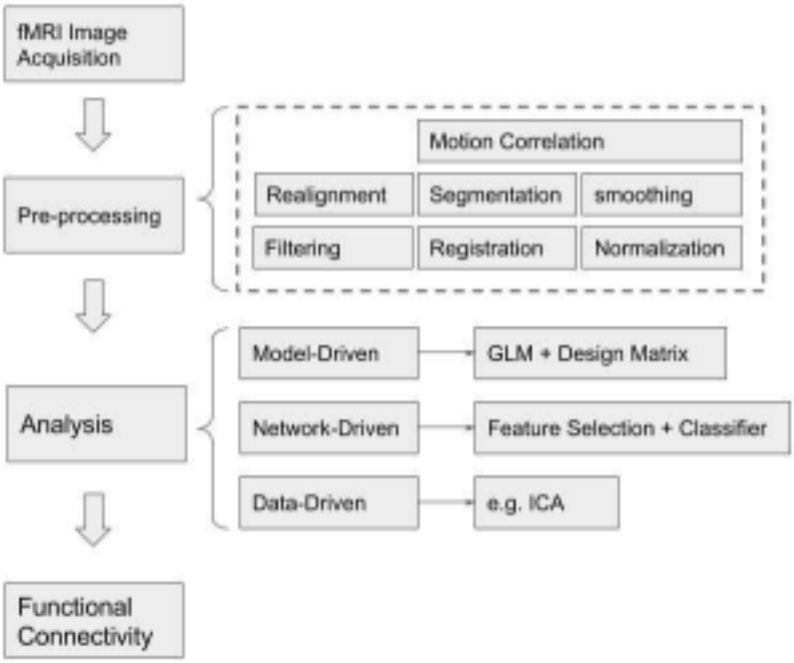


Figure 2.44: Functional MRI Analysis Pipeline

weights of the predictors to form an approximation of the observed signal. It can be mathematically described in a matrix form as:

$$Y = \beta X + \varepsilon \tag{2.18}$$

where Y is the observed data, X is the known variables (i.e. design matrix in fMRI analysis), β is the coefficients to be estimated, and ε models errors. The design matrix is the numerical description of known sources of variance in the experiment, such as information about stimulus conditions. The coefficients,β , are usually estimated using Ordinary Least Square (OLS), which finds the 'optimal' solution that has the least residuals. After the coefficients of a GLM model are estimated for each voxel, specific hypothesis can be tested using parametric analysis, like F- or t-test statistics.

Figure 2.45 shows an example of using GLM to model an imaginary time series data from a voxel (Monti, 2011). The design matrix contains three regressors of interests which are corresponding tasks; a number of nuisance variables which include six motion regressors and a linear drift; the parameters (β_1, β_2,..., β_F) are the coefficients to be estimated via the OLS approach; and the error term, ε.

The GLM model in fMRI analysis is relatively intuitive and flexible, however, there have been many discussions about its limitations, principally because it relies on assumptions that may not hold in all situations. It is suggested that checking the assumption is essential, and it can be done with diagnostic tools and more advanced models are encouraged to develop (Poline and Brett, 2012; Monti, 2011).

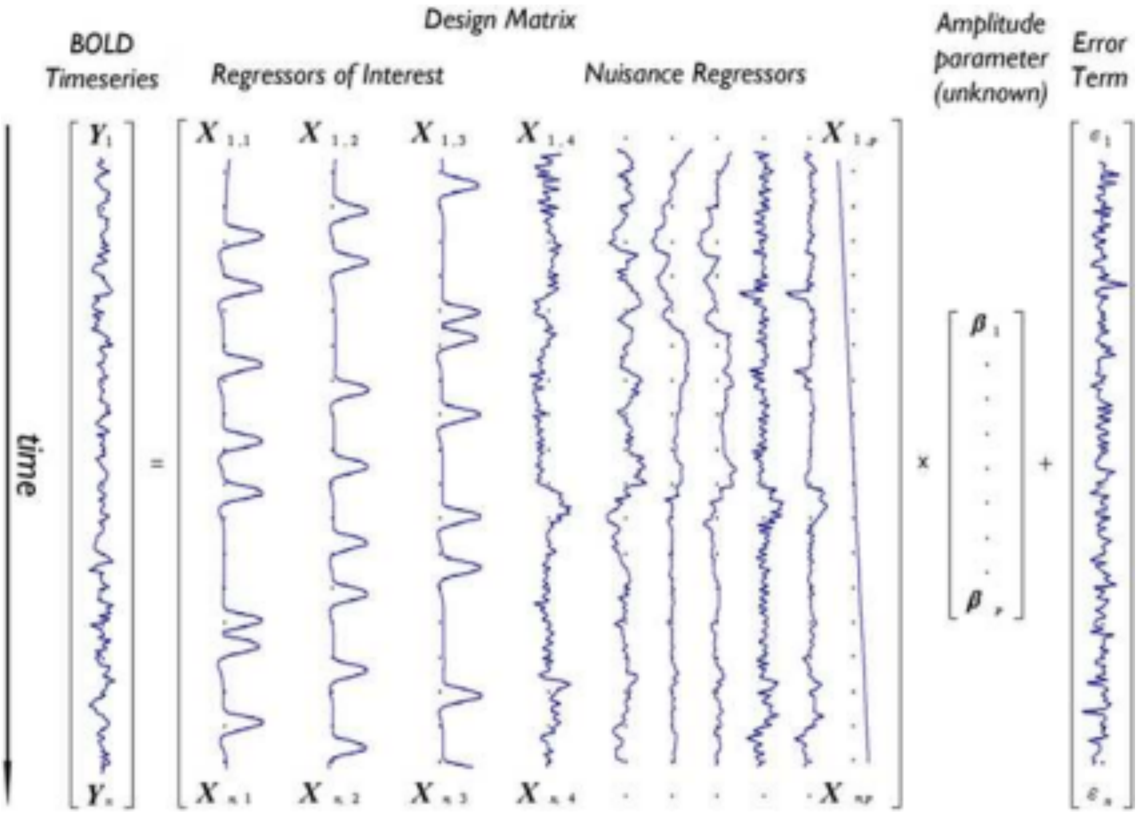


Figure 2.45: Illustration of General Linear Model (Monti, 2011)

2.4.3.2 Nework-Based Analysis

To infer the coherence in the fMRI time-series data, regions of interest (ROI) are commonly used for cross-correlation analysis. In the context of graph analysis, the ROIs are the nodes in the graph and their correlations are the edges. There are many ways of defining the ROIs or nodes in a graph. In fMRI, spatial ROIs may be obtained using brain atlases or from functional localizer tasks (Sato et al., 2012). An illustration can be seen in figure 2.46, which shows a study focusing on fractional Amplitude of Low Frequency Fluctuations (fALFF), Regional Homogeneity (ReHo), and Resting-State Network (RSN) (Sato et al., 2012). The ROIs are selected using an atlas named 'CC400'. The signals within the ROIs are fed into a trained classifier which will produce the activation patterns.

To infer the coherence in the fMRI time-series data, regions of interest (ROI) are commonly used for cross-correlation analysis. In the context of graph analysis, the ROIs are the nodes in the graph and their correlations are the edges. There are many ways of defining the ROIs or nodes in a graph. In fMRI, spatial ROIs may be obtained using brain atlases or from functional localizer tasks (Sato et al., 2012). An illustration can be seen in figure 2.46. Provided the pre-processed fMRI data, the ROIs can be chosen by using a standard atlas. Figure 2.46 shows a study focusing on fractional Amplitude of Low Frequency Fluctuations (fALFF) (Zou et al., 2008), Regional Homogeneity (ReHo) (Zang et al., 2004), and Resting-State Network (RSN) (Sato et al., 2012). The ROIs are selected using an atlas named 'CC400'. The signals within the ROIs are fed into a trained classifier which will produce the activation patterns.

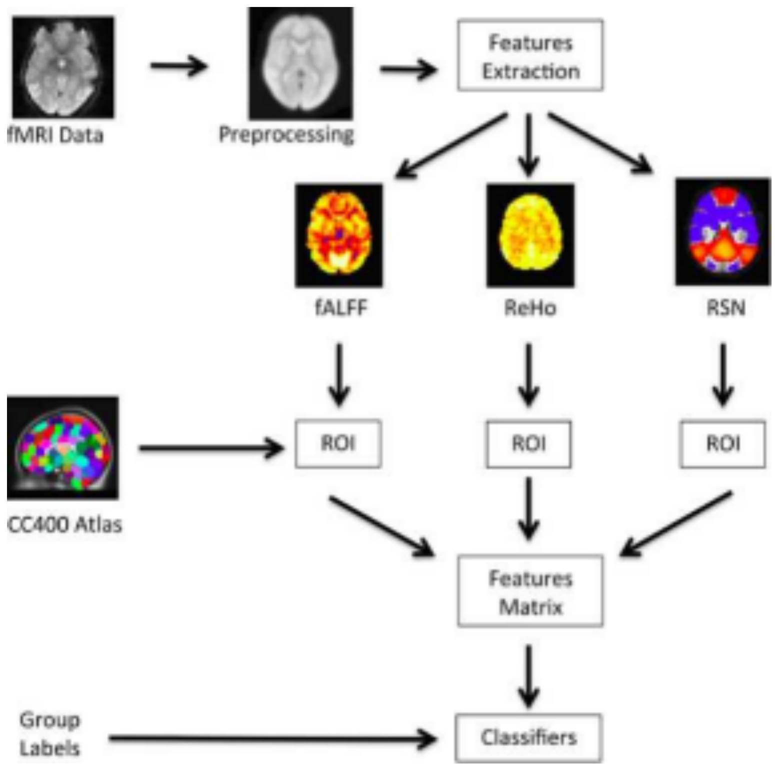


Figure 2.46: Network Analysis based on Atlas (Sato et al., 2012)

Among the networks, the default mode network (DMN) is commonly studied using the network-based approaches, with resting state data. It is a task-independent approach, based on fMRI time-series data to monitor the brain when the subject is rest, and not performing any experimental tasks. The purpose of measuring the resting-state activity is to explore the special brain activation patterns (i.e. networks) that predominate when no complicated tasks are processed, except accessing working memory. It is found to be active by default and is mutually-exclusive with other task-orientated networks (Greicius et al., 2003). Also, network-based studies on DMN is used in diagnosing Alzheimer's (Greicius et al., 2004), Parkinson's diseases (van Eimeren et al., 2009) and etc.

2.4.3.3 Data-Driven Approach

More recently, some studies have begun applying model-free analysis with independent component analysis (ICA) to identify ROIs and activation patterns free of a priori models (i.e. brain networks). ICA is an algorithm that is widely used in the field of blind source separation (Zibulevsky and Pearlmutter, 2001), which is a process to extract original signal sources, given the the linear combination of the sources. A well-known example of the scenario is the "cocktail party" problem, where multiple recorders are placed in different locations in a room and record the sound that is composed of different sources; and ICA in this case is able to separate different sound sources by analyzing the recorded signals (Haykin and Chen, 2005).

In fMRI data analysis, ICA can be used to define independent components (i.e. spatial maps), which can be ROIs or brain networks. The number of components can be defined within the ICA algorithm, affecting the interpretation of the estimated components. If a small number of components are extracted, then the ICs are most likely a combination of correlated regions, which can be regarded as brain networks (Calhoun et al., 2008). A brain network refers to a set of non-contiguous regions having similar activation patterns or time courses. These brain networks usually correspond to some functional system in the brain. On the other hand, ICs are more likely to be independent functional regions, if a higher number of components are extracted from ICA algorithm (Calhoun et al., 2003; McKeown et al., 1998). In common network-based analysis, these smaller regions are regarded as nodes (Kiviniemi et al., 2009). An illustration of applying ICA in fMRI time-series data can be seen in figure 2.47. It shows that fMRI data can be decomposed into temporal and spatial components. Usually, we are more interested in the spatial map that stands for the correlations between spatially distinct regions based on their temporal similarity of activation patterns.

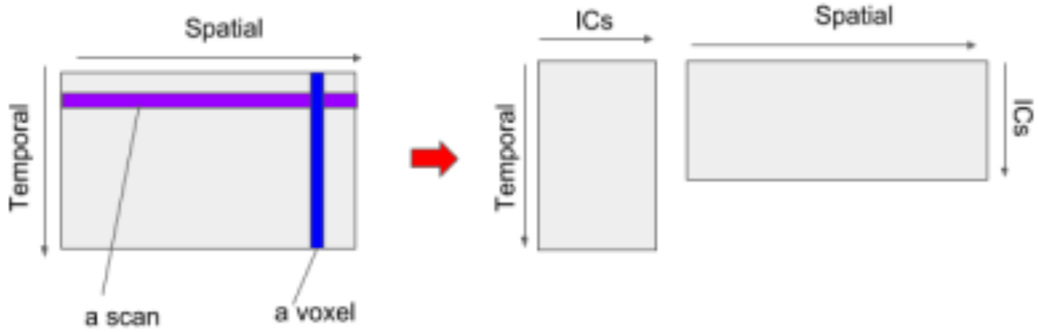


Figure 2.47: An Illustration of Spatial ICA in FMRI

So it is unclear how to define a reasonable number of ICs to extract in fMRI time-series data. For a particular functional experiment and in lieu of a hypothesis (e.g. known regions that correspond to the stimulus), it is necessary to have multiple attempts at various number of ICs and visually inspect the results. Still, ICA is one the methods that are highly recommended by many studies for its flexibility (Malinen et al., 2007).

2.4.4 Neurofeedback

Among the applications of fMRI, neurofeedback (NF) is becoming more attractive in both research and clinic settings. It is a technique that entails the presentation of brain activity back to the subject while scanning, with a sub-second latency, and prompts the subject to manipulate his/her brain functions in response to the presented information, which yields a 'neurofeedback' loop. Figure 2.48 illustrates the concept of a neurofeedback loop, which includes fast data acquisition (e.g. EPI sequence), a real-time processing component, construction of feedback signals (e.g. audio, vision) and feedback to the subject while they are still in the scanner, being imaged. The feedback is usually simple, such as a number, a shape, or a simple "thermometer" type display for a variable. Based on the feedback, the subject needs to manipulate their function activities to achieve a positive feedback. For example, the feedback is a scale between 1 and 10, and subject is asked to try increasing the number as large as possible. Also, it is important to note that the subject is not told how to effect the desired change. It is an implicit learning task at which some subjects will naturally be better or worse at than others.

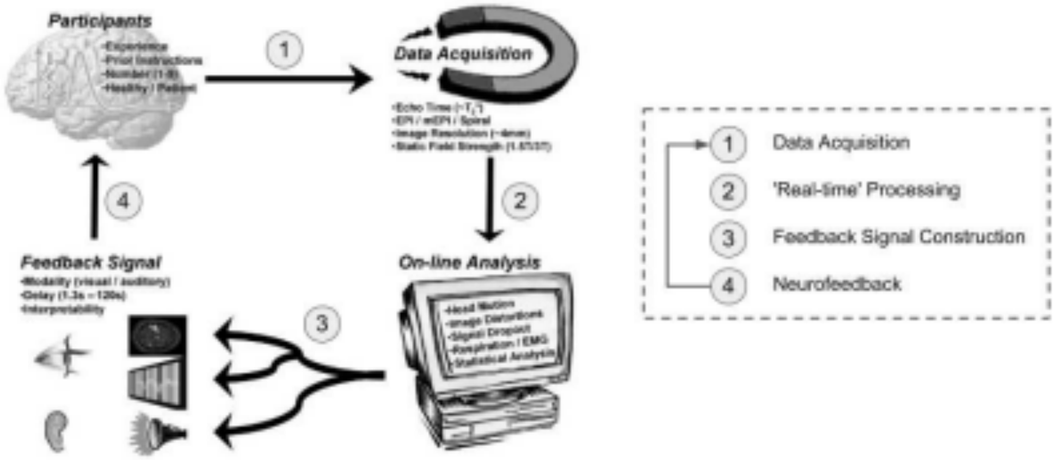


Figure 2.48: Illustration of Neurofeedback Closed Loop (Sulzer et al., 2013)

While electroencephalogram (EEG)-based NF has been applied as a non-invasive tool for exploring cortical activities, fMRI-based neurofeedback is a nascent technique. Some early applications of neurofeedback based on fMRI can be found in a study that investigates the short-term behavioural impacts of fMRI-neurofeedback on subjects' speech processing and language-related performance, by selectively activating the right inferior frontal gyrus (IFG) (Rota et al., 2009). It hints at a rich future for fMRI-based NF studies. Furthermore, some researchers are already studying the influences of different feedback presentation methods to improve the measurement of BOLD signals, like combining structural information with diffusion MRI (Koay et al., 2012).

It is challenging to develop systems that can perform fMRI analysis in real-time, because of the increasing fMRI data set size and the computational intensity of some advanced algorithms. To accomplish real-time fMRI neurofeedback during scanning time, it is essential to optimise the four steps: data acquisition, preprocessing, feedback signal construction and feedback with audio or vision. Fast data acquisition mechanisms have been developed, like multiple-Echo-Planar Imaging (Multi-EPI) (Weiskopf et al., 2005) and adaptive multi-resolution EPI (Yoo et al., 2001), and imaging under high magnetic fields (Hollmann et al., 2008). These new techniques improve the speed of acquiring data, at the same time, results in increased data size. For this reason, most of the current neurofeedback implementations are still hypothesis-driven, which requires less computing power than data-driven approaches, such as ICA.

Chapter 3

Acceleration of Tractography

3.1 Introduction

Diffusion-weighted (DW) magnetic resonance images can be processed to yield voxel-wise information about the underlying anisotropic orientation distribution (i.e. model fitting) and generate streamlines that represent/simulate the white matter fibers in the brain (i.e. tractography) (Tournier et al., 2011). Traditional diffusion-tensor-based methods are insufficient to model crossing fiber structures (Tournier et al., 2011; Farquharson et al., 2013). Therefore, to overcome the shortcoming, more advanced techniques, like diffusion spectrum imaging (DSI) (Hagmann et al., 2004), Q-ball imaging (Tuch, 2004), constrained spherical deconvolution models(CSD) (Tournier et al., 2007) and other non-tensor models have been developed. Also, as MR field strengths increase and scan protocols are improved, the spatial and angular resolution that can be achieved have reached the point where diffusion-tensor-based methods are being replaced in favour of non-tensor methods, so as to allow for more complex model of fiber distribution to be fitted. Based on those orientation models, streamlines can be constructed by the tractography methods, and among those, CSD-based tractography has proven to be one of the robust methods to accurately represent the underlying fibers structures (Fillard et al., 2011).

However, one of the trade-offs of using the non-tensor models (e.g. CSD) is the complexity of the processing pipeline, which leads to high compute and storage demands, particularly when applied with high spatial resolution datasets. The use of higher order models requires more parameters (e.g. 45 parameters for spherical harmonic order 8 in the CSD model) to represent the approximate diffusion profile. This increases not only the size of the data files, but also the complexity of computation during tracking processes. In research studies, like investigating the human connectome, a large number of streamlines (e.g. at least 10 million) that are generated in a single subject is necessary for using post tractography filtering of tractogram (Smith et al., 2013) and providing a level of statistical confidence. It substantially increases the time required for processes like tractography. On the other hand, the “real-time” generation of fiber tracts is essential to meet clinical demands, such as pre-surgery planning and more importantly for in-surgery navigation.

Generation of higher spatial resolution images from large-scale brain imaging projects, e.g., Human Connectome Project (HCP) (Van Essen et al., 2013), also demands accelerated techniques that can generate tens of millions of white matter tracts in each individuals within a reasonable timeframe (Van Essen et al., 2012). The generation of streamlines (i.e. stochastic representations of white matter fiber bundles) for the whole brain is computationally dense, intrinsically parallel and is eminently suited to acceleration with the contemporary graphical processing unit (GPU). GPU(s) has/have been used in many studies that are mentioned in the background section (i.e. Chapter 2), and several attempts have been made to accelerate the tractography algorithm, but most of them are dealing with tensor-based deterministic tracking methods. Therefore, in this study, we are focusing on acceleration of non-tensor-based (i.e. CSD-based) probabilistic tractography.

In this chapter, I extend from my supervisors' work (Raniga et al., 2012), analyze the algorithm, further improve the performance of the coarse-grained parallel mapping; then, we design and implement another model to accelerate the CSD-based probabilistic tractography problem (Tournier et al., 2007, 2012) using GPU(s). In the following sections, we describe the design and implementation of two new independent parallel mappings of the CSD based tractography algorithm, and present measured performance gains for the two implementations across contemporary GPU hardware. We conclude with a discussion of the results and some commentary about future directions for further acceleration and potential application of fast tractography.

3.2 Methods

In this section, we present our two parallel mappings of the probabilistic tracking algorithm based on the CSD model. The first mapping, which we will refer to as the coarse-grained parallel mapping, is naturally motivated by the seed-based data parallelism of the problem, in which threads propagate streamlines from different seeds independently. However, its performance is expected to be limited by the inefficient use of on-chip cache, since each thread requires a separate cache space to store the frequently used data for the current position. This leads to a second mapping, the fine-grained model, which further divides and parallelizes the process into multiple threads in a cooperative manner. Besides illustration of the two models, several other optimizations that are essential to the overall performance improvement will also be discussed, followed by the experimental setup within which we implemented, validated and performance tested the models.

3.2.1 Coarse-grained Seed-based Parallel Mapping

The first mapping we propose is based on seed-wise independence. In this mapping, each tracking process has its private workspace and no communication with other tracking processes is required. The parallelism is achieved by evenly distributing the number of seeds to all the working threads on a GPU and each tracking process generates 1 streamline at a time. The large number - potentially millions - of tracking processes are executed as individual, independent threads on the GPU. Each thread performs a single entire tracking process, whose constituent tasks are: (i) initialise the current position with a new seed; (ii) interpolate the FOD function at the current position; (iii) randomly sample the function space and propagate the track in the first direction found that satisfies the threshold condition; (iv) evaluate if a streamline has terminated; (v) if so write the track to memory and re-seed; if not, then sample another position in the space and return to step (ii).

Like a typical GPU parallel program, the implementation of the coarse-grained mapping includes both the serial portion (e.g. data loading) that runs on CPU and the parallel code (i.e. kernel functions) running on GPU(s). In the serial code (i.e. preparation stage), all the necessary data are loaded onto host (i.e. CPU) memory, and those required by the parallel tracking processes are explicitly copied to the device (i.e. GPU) memory, including the voxel-wise FOD data, seeds, the affine matrix, and the whole brain mask file. The FOD data contains the SH coefficients for each voxel, e.g. an array of 45 floating point numbers to represent the orientation distribution of a voxel (in the case of lmax value of 8); Seeds (i.e. around 2 million) are positions randomly selected from a ROI or whole brain volume, and they are stored as a text file in the format of (x,y,z); the affine matrix is used to transform the position from image space to scanner space, as the tracking process occurs in scanner space; and the brain mask is also used to prevent the generated streamlines exist the WM of the brain.

The parallel part of the implementation (i.e. the kernel code) is executed by all the threads that are allocated during the kernel launch. It can be determined by configuring the number of threads per block, and the number of thread blocks. For example, a kernel is launched with a setting of 32 blocks of 128 threads, which results in 4096 threads in total that are allocated to GPU cores and potentially executed in parallel. Each thread works on an independent subset of the the seeds which are divided and assigned to the threads based on their global IDs. In the case of a configuration with 4096 threads, thread with ID N deals with seed N, N + 4096, N + 2*4096, and etc, for the purpose of achieving coalescing memory access. It is also worth mentioning that the seeds are usually not evenly distributed, therefore, the last subset of the seeds (e.g. some number, K, that is smaller than 4096) are assigned to the first K threads, while the remaining (4096 - K) threads keep idle.

In the coarse-grained mapping, threads in a block do not communicate, i.e. no data sharing occurs between threads. As shown in figure 3.1, it is an illustration of the execution process of the coarse-grained implementation on GPU, and it is important to point out that the execution of the those steps in the same thread blocks are in synchronization, which may lead to an issue of imbalanced streamline lengths. In other words, the execution time is determined by the tracking process that generate the longest streamline while other threads keep idle. As highlighted in figure 3.1, some of the threads in the block are idle, which results in wasted compute cycles. However, this type of imbalanced workload can not be avoided, due to the contemporary GPU architecture. On the other hand, thread blocks are assigned to individual streaming processors (SM) and executed independently. With sufficient number of thread blocks, the on-chip scheduler seamlessly switches the warps from different thread blocks to achieve a high occupancy and correspondingly high throughputs.

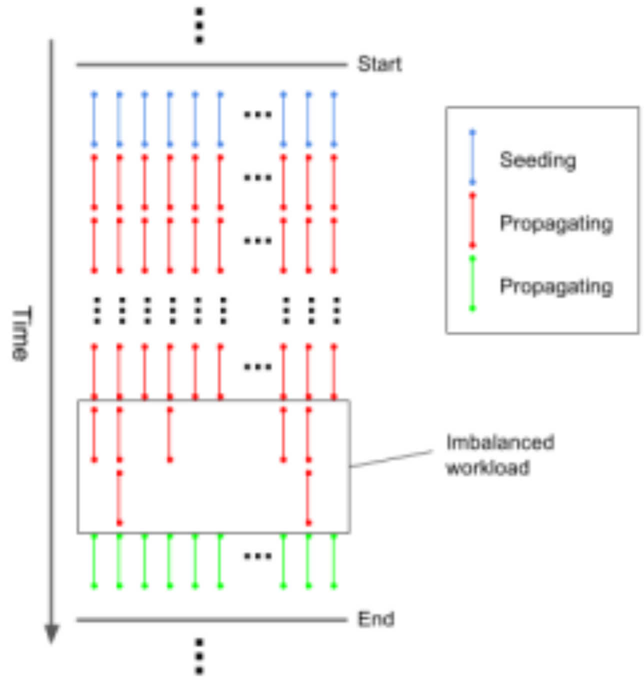


Figure 3.1: Illustration of Imbalanced Workload

In the coarse-grained mapping (figure 3.2), memory usage is optimized via using shared memory to store SH coefficients and thread-wise random numbers, using the interpolation functions from texture memory, and using local registers for temporary data. SH coefficients are frequently used in the tracking process, as multiple trials (e.g. 50 trials) are needed to determine the initial direction at the position of a seed, and every next possible positions during the propagation. Therefore, using shared memory to store the coefficients reduces the access to high latency global memory. The number of trials to find the initial direction and next position are predefined, eg. 50, and these trials are executed in sequence, until one accepted position is found without reaching the maximal trail limits. Texture memory is used to obtain the interpolated SH coefficients at a given position. The built-in interpolation function of texture memory is originally designed for rendering frames, therefore, it is the fastest interpolation option on GPU and supports both linear and bilinear interpolations. Temporary variables in the kernel resides on registers, such as the control variables and temporary positions. Furthermore, the implementation reads the seeds and stores the generated streamline points directly from/to the global memory, as these operations are efficiently coalesced. Besides, independent private memory (i.e. two buffers) are allocated to each thread, where all the accepted tract points are stored.

Since there is no sophisticated pseudo random number generator implemented in the CUDA toolkit, we use linear congruential random number generators (LCRNG) (Park and Miller, 1988), which generates a random number based on the previous value. The

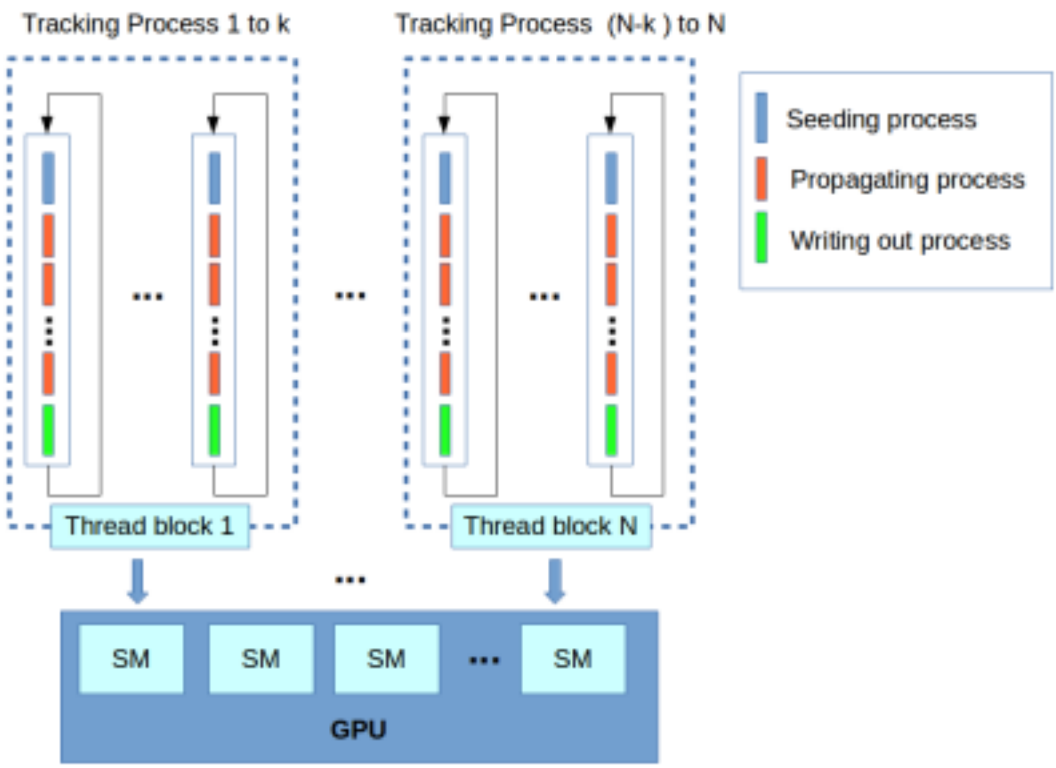


Figure 3.2: Work flow of the Coarse-grained Voxel-based Parallel Mapping

seeds for this in-kernel LCRNG function are generated by CPU functions, because no random number generator functions are implemented on GPU, and then copied to the GPU global memory for use by the LCRNG generator. The pseudo random number generator is invoked frequently, and each function call involves a pair of read and write memory operations. Storing the random numbers in the shared memory can further reduce the chance of global memory access latency. The LCRNG function is not the ideal random number generator, but it is still acceptable in this case.

The kernel implementation is designed as a state machine shown in listing 3.1, which indicates the status of the tracking process and determine the operations need to be executed. In the kernel, status/stage 0 means the seed initialization (e.g. loading seed position); status 1 is the stage of getting the interpolated SH coefficient at a given position and estimating the orientation at the position; status 2 means that propagation is in the forward direction (i.e. the direction is originated from the position orientation of the seed); while status 3 means the tracking is in the backward direction (i.e. along the negative orientation from the seed). The kernel is launched with default state (state 0), then switches to state 1 if the SH orientation amplitude at the position of the seed is above a certain pre-defined threshold. The threshold is used to prevent unexpected streamline from regions that are not in WM. Usually, streamlines propagate from forward direction until it terminates, then grow from the other direction from the seed. In each propagation step, first position that satisfies the selection criteria is accepted as the next position in the streamline. A track is terminated when no random position can be selected after certain number of trials (e.g. 50). Furthermore, the operation of calculating the FOD amplitude at a given position can be further optimized based on the SH degrees (e.g. lmax value of 4, 6, 8, and 10 which are most commonly used). The last step in the kernel is to check by each thread whether their private buffer is full. If so, the current tracking states are saved and the thread is terminated.

Listing 3.1: Pseudocode of the Coarse-grained model

```
BEGIN
AllocateDoubleBuffer();
    While 1:
        (Stages):
        0: Initial stage:
            GetANewSeed();
            IF success:
                Go to stage 1
        1: Init a seed:
            GetLocalInterpolatedSHcoefficent();
            InitDirectionForTheSeed();
            IF success:
                Go to stage 2
        2: Grow a track in forward direction:
            PickALocation(step_size):
                GetInterpolatedSHcoefficentOnTheLocation();
                Try n times:
                   CalculateFODamplitude():
                   Switch(lmax value):
                       4: CalculateFODampFor4();
                       6: CalculateFODampFor6();
                       8: CalculateFODampFor8();
                       10: CalculateFODampFor10();
                       Default: CalculateFODampForOtherLmaxs();

                   IF > threshold:
                       PickThePointAsNewLocation();

        3: Grow a track in the reverse direction:
            //Same As stage 2

        IF buffer is full:
            Stop the kernel
```

Although the coarse-grained model has largely parallelized the traditionally serial tractography process, it is worth of pointing out that this model is limited by the overuse of shared memory, and it results in performance bottleneck. As described above, each thread has its cache to store, for example, 45 floating-point numbers representing SH coefficients and random number values in a normal spherical harmonic configuration (i.e. order of 8), which are 180 bytes(45 * 4 bytes/float) in size. A contemporary high-end GPU has only 48KB shared memory per symmetric multi- processor, which limits the number of threads that can be allocated. Therefore, we developed an alternative mapping which aims to reduce the overall shared memory usage and potentially improves the performance.

3.2.2 Fine-grained Seed-and-Task-based Parallel Mapping

Fine-grained mapping utilizes both the independence of seeds and the operation of randomly sampling the FOD space to choose the next possible position in the streamline. During the tracking process of coarse-grained mapping, the propagation is done by randomly sampling positions along the current tracking direction in a sequential way, which is clearly parallelizable. Therefore, the proposed fine-grained mapping combines a group of processes (i.e. threads) to track the one streamline at a time cooperatively, and specifically parallelizes the random sampling operation of FOD space and computing the corresponding FOD amplitudes.

The major difference compared to the coarse-grained mapping is the tracking workflow in the kernel function. Figure 3.3 shows the basic workflow of the fine-grained parallel mapping, whose concept is similar to the well-known Mapreduce model (Dean and Ghemawat, 2008). It includes two types of threads: a master thread which controls the main tracking procedures and executes the process of loading seed and writing out completed streamline positions; computing threads (i.e. working threads) which execute the operations of sampling random positions and calculating their corresponding FOD amplitudes. The general process is that (i) The master thread initializes the process by reading the seed and loading the data, then (ii) waits for the worker threads to compute the values of the FOD amplitude at randomly sampled directions; and (iii) master thread chooses the first direction that satisfy the criteria as the next position on the streamline and saves the position; (iv) master thread sets the selected position as the current position and repeats step iii; if a streamline is terminated, return to step i with a new seed; The process repeats until all the seeds are completed.

In the implementation, each tracking process (including 1 master thread and the remaining working threads) is configured as one thread block. As the kernel code is executed by all the threads, the individual local ID (i.e. IDs within a thread block) is used to distinguish between the master thread (usually with ID 0 in a thread block) and working threads. In our implementation, 16 threads are assigned in a thread block, which means one master thread and 15 working threads. The reason is that 60 trials are sampled for each position, therefore, it can be completed within 4 rounds. The 'Map-Reduce' pattern is achieved by using thread branching that is introduced in the background section. As shown in figure 3.4, thread synchronization is explicitly placed at places where data dependence occurs. For example, synchronizations are required between the operation of loading seed and computed the SH coefficients at the seed position (① in figure 3.4); before the master thread determines the next possible position based on the computed FOD amplitudes(② in figure 3.4). To summarize, every time a new position is chosen, its interpolated FOD coefficients are updated by the master thread, while the operations of computing FOD amplitudes on randomly sampled position are done by working threads.

The purpose of this design is to reduce the usage of shared memory so as to increase thread occupancy and therefore performance. Compared to the coarse-grained mapping, it only requires one set of FOD coefficients for one thread block, which significantly reduces the shared memory usage. The coefficients are also stored in the shared memory, because all the threads within the same group (i.e. thread block) need to access to these values during orientation random sampling process. However, extra shared memory needs to be allocated to the FOD amplitudes along the randomly sampled orientations and their corresponding positions. They are stored as an array of 4D data, (x,y,z, ρ), where (x,y,z) is the position and ρ is the amplitude value. During the process, the master thread can determine the next position by comparing those values against the pre-defined threshold and choosing the first acceptable position. Overall, the usage of shared memory is significantly reduced.

Listing 3.2: Pseudocode of the kernel of fine-grained mapping

```
BEGIN
AllocateDoubleBuffer();

    While 1:
        (Stages):
        0: Initial stage:
            GetANewSeed();
            IF success:
                Go to stage 1

        1: Init a seed:
            IF Master thread:
                  GetLocalInterpolatedSHcoefficent();
            IF Worker threads:
                  CalculateFODamplitudesInParallel();
            IF Master thread:
                  CollectResultsFromWorkers();
                  InitDirectionForTheSeed();
                  IF success:
                        Go to stage 2

        2: Grow a track in forward direction:
            PickALocation(step_size):
                GetInterpolatedSHcoefficentOnTheLocation();
                Try n times:
                       IF Worker threads:
                           CalculateFODamplitudesInParallel();
                   Switch(lmax value):
                       4: CalculateFODampFor4();
                       6: CalculateFODampFor6();
                       8: CalculateFODampFor8();
                       10: CalculateFODampFor10();
                       Default: CalculateFODampForOtherLmaxs();

                        IF Master thread:
                        CollectResultsFromWorkers();
                        IF > threshold:
                                PickThePointAsNewLocation();

        3: Grow a track in the reverse direction:
            //Same As stage 2

      IF buffer is full:
          Stop the kernel
```

A similar structure of kernel can be seen in the pseudocode in listing 3.2, except that it involves thread branchings where master thread and working threads are executed internally. Like the coarse-grained mapping, the fine-grained model is also implemented like a state machine which uses unique state IDs to represent different steps in the tracking process. Extra selection statements are added in the implementation to separate operations by master thread and by working threads. They are if-else statements, and are used to compare the thread local ID; if the thread ID is 0, the thread executes as a master, otherwise, the thread with non-zero IDs are working threads.

In the fine-grained mapping, explicit thread synchronization operations are required to guarantee the procedures work in the correct and expected manner. The synchronization before the worker threads start to sample the FOD space is necessary to make sure that the FOD coefficients are successfully loaded to the shared memory and updated (this is a standard operation in GPU programming). The second synchronization is needed before the reduction process in the master thread to ensure that all the threads finish computing the FOD amplitudes and storing the values in the shared memory. Although only a small number of thread synchronizations are needed, they are invoked very frequently, which is expected to affect performance, and the thread-branching pattern in GPU also potentially reduces overall performance. Furthermore, 16 thread per block leads to a low GPU occupancy and half of a warp (a warp is 32 threads) is idle at any time, which could be another issue in this mapping.

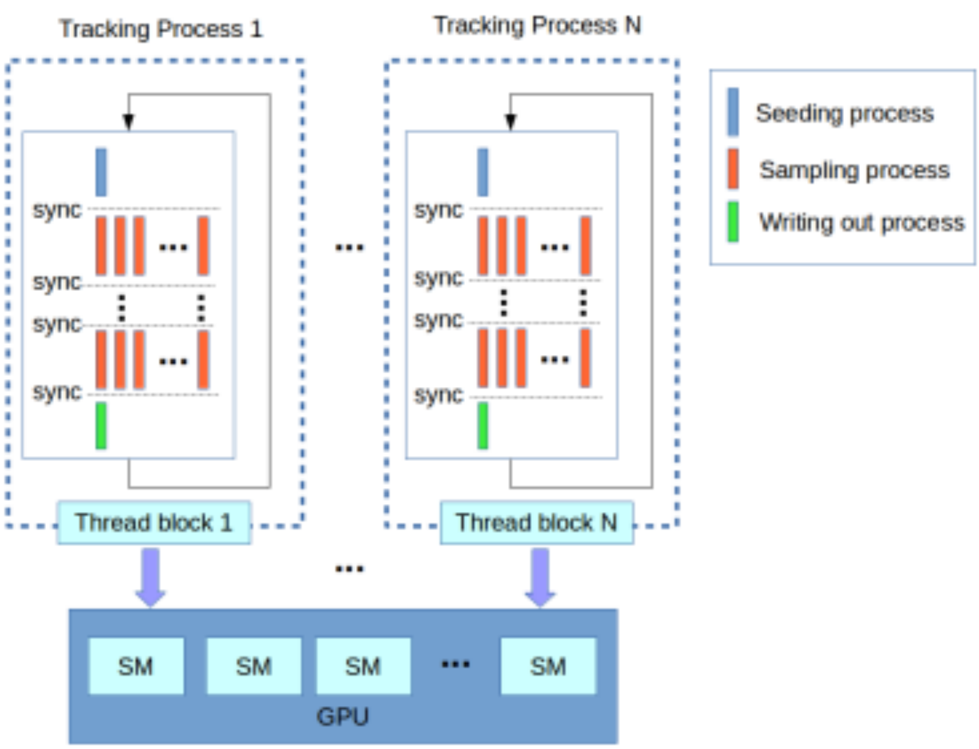


Figure 3.3: Workflow of the Fine-grained Voxel and Task Based Parallel Mapping

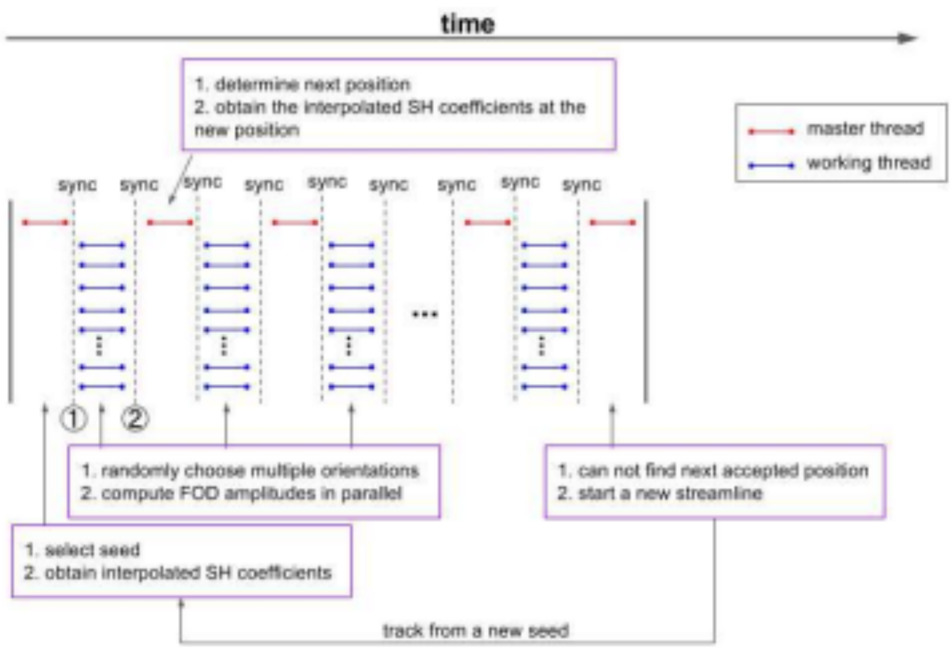


Figure 3.4: Illustration of MapReduce in Fine-grained Mapping

3.2.3 Other Optimizations

We implemented the coarse- and fine-grained algorithms for probabilistic tractography of CSD diffusion images, using driver functions written in C/C++ and NVIDIA CUDA framework. Standard CUDA methods for allocating, writing and reading GPU memory, and for invoking kernels, were used. The functions were built into command-line tools yielding a compatible drop-in replacement for tckgen (in Mrtrix 3) or streamtrack (in Mrtrix 2) to accelerate existing workflows to generate tractographs from the reconstructed streamlines.

In the implementation, four standard techniques were used to optimise the kernel and driver programs as follows:

Double buffering: both parallel mappings have been implemented with double buffers, which are used to ameliorate the effects of the relatively poor latency for data communication between CPU and GPU. The purpose of using double buffers is to divide tracking process into several runs to avoid the the situation where the application may exhaust the memory storage on GPU. The data transfers between the two storage areas are strongly limited by the bandwidth of the Peripheral Component Interconnect Express (PCIe) interface which connects the GPU card to the host system. By using double buffers, two separate buffers in the global memory are allocated on GPU. The tracking application generates and writes the streamlines into one of the buffer at one time, until the buffer is fully occupied. In figure 3.5, it illustrates the usage of double buffer. Under the condition that one of the buffers is full (②), the tracking kernel is terminated and relaunched immediately and asynchronously with another buffer(③); meanwhile the data in the full buffer is transferred to the host memory and saved to disk. The states of the previous tracking process are saved onto GPU global memory, then the next tracking kernel continues from the previously saved states.

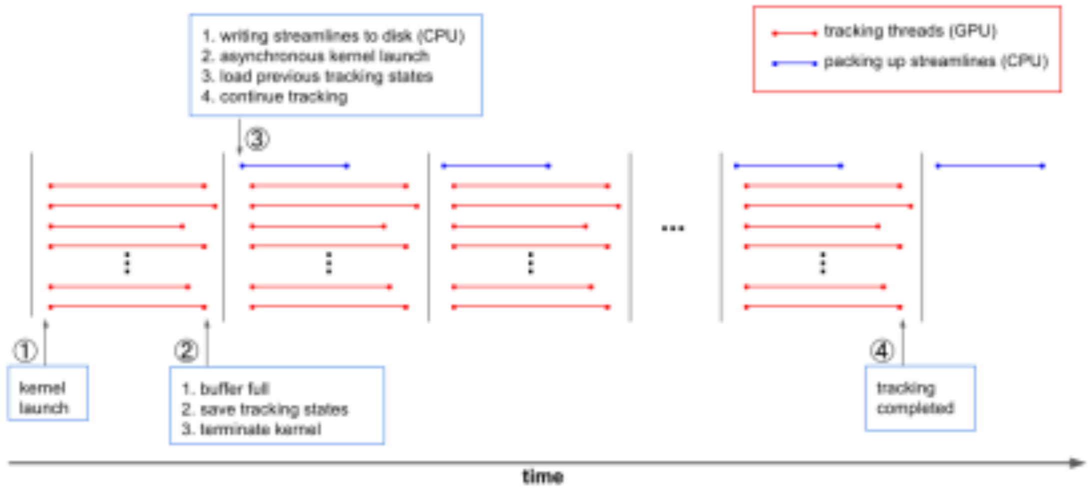


Figure 3.5: Asynchronous Kernel Launch with Double Buffers

Loop unrolling: in the both coarse- and fine-grained kernels, the operations of computing FOD amplitudes which are performed as a summation, are unrolled. A looping structure is the usual way of performing summation in common programming languages. The loops are explicitly unrolled to save the compute resource of function calls and variable updates during iterations. The unrolled structure includes the summation operations for each spherical harmonic order and degree. In our implementation, only functions that calculate the FOD amplitude values for SH orders of 4, 6, and 8, because these options are mostly used in practice. And a default function (without explicit unrolling) is provided for other options of SH orders, such as 10, 12 and etc.

Pre-computing values: pre-computed values for normalization factors and the associated Legendre polynomials are used as constant variables in the implementation, since these factors only depend on the SH order and degree. Using the pre-computed values can significantly reduce the overall time cost. While computing these constant values involves complex numeric operations, such as square root and exponential operations, particularly for the associated Legendre polynomials, the main reason for pre-computing them was to reduce the register load and the usual recursive implementations for computing these are not well suited to GPUs.

Optimized thread Configuration: The above optimisations are standard practice on most parallel architectures. But the most gain from a GPU arises when thread configuration is optimised. It is essential for the the overall performance, particularly when the issue of limited shared memory storage is known in the coarse-grained model. In that case, it needs to reduce the threads in each block to enable more threads to be executed at the same time. In the fine-grained parallel model, the hardware limitation of the number of thread blocks are considered to be the main issue, because this model defines a constant thread block size (i.e. 16 thread in one block in our case), which is not ideal for contemporary GPU architectures. Therefore, in both of the cases, careful experimentations of the thread configuration are critical for the overall performance.

3.2.4 Experimental Setup

Diffusion data: The HARDI diffusion-weighted data used in the experiments is from the HCP dataset, with ID of"100408" (Van Essen et al., 2013). The diffusion-weighted images are acquired using a 288 diffusion-weighted direction scheme, a b-value of 3000 s/mm^2 and a spatial resolution of 145x174x145. Diffusion weighted directions are randomly and evenly sampled on the sphere. The number of directions has to be equal or larger than the minimal requirement for spherical harmonics order of 8, which is 45 directions. Using this experimental setting, it results in a raw diffusion-weighted file of approximately 300 MB in size. And the size of files that store corresponding FOD coefficients for spherical harmonic order of 4, 6, and 8 are 210MB, 391MB and 628MB respectively.

Seeding mechanism: seeds from which streamlines are tracked are randomly sampled from the data space by randomly choosing a voxel and position within that voxel. A large number of streamlines are needed in most quantitative studies for the purpose of statistical reliability, typically 100,000 to 1,000,000. The seeding process is repeated until sufficient tracks are generated. The seeds are stored in the format of (x,y,z) and saved in a text file with each row representing a seed position.

Hardware: performance was measured on three generations of NVIDIA GPUs: M2070 (Fermi), K20m (Kepler) and GTX970 (Maxwell) and compared with 6-core and 12-core Intel X5650 CPUs. The overall time costs are measured for the GPU implementations on both of the GPU architectures. For comparison, we run performance measurements on the CPU systems, using MRtrix (Tournier et al., 2012). All the experiments are run on the local computing cluster MASSIVE (Goscinski, 2012).

In our work, we compare the performance between CPUs (i.e. Intel Xeon X5650 CPU) and three GPUs (i.e. NVIDIA Tesla M2070, Tesla K20 and GTX970) which are listed in table 4.3 and table 3.2. Single Xeon X5650 CPU has 6 cores running at 2.6 GHz, with the peak float precision processing performance of around 124.8 GLOPs. It has a three-level cache architecture, including 32KB L1 cache, 256KB L2 cache and a 12MB L3 cache. On the other hand, the GPUs have a significantly different architectures. Tesla M2070 has 448 cores at clock frequency of around 1 GHz, which can achieve a peak floating point processing of 1288 GLOPs. Another GPU from Tesla series, K20, features 2496 CUDA cores, running at a relatively lower clock speed, 706 GHz. NVIDIA GTX 970 is a commodity GPU processor, mainly targeting gaming purposes, unlike M2070 and K20 GPUs that are categorized as industry-orientated chips. GTX 970 chip is based on the latest Maxwell architecture, with 13 Streaming Multiprocessors (SMMs) and each SMM has 128 lightweight CUDA cores, running at the clock speed of 1050 GHz. Also, the shared memory space is no longer unified with L1 cache in the previous generations, and the size is increased to 96 KB/SMM.

Table 3.1: CPU Hardware Information

CPU	Intel Xeon X5650
Architecture	X86
Clock Speed(MHz)	2670
Total Core	6 (12 threads)
Perforamnce Peak(GFLOPs)	124.8
L1 Cache	32 KB
L2 Cache	256 KB
L3 Cache	12 MB
Peak Performance Per Watt(GFLOPs/Watt)	1.31
MSRP(when released)	>$1000

Table 3.2: GPU Hardware Information

GPU		NVIDIA Telsa M2070	NVIDIA Telsa K20	NVIDIA GTX 970
Architecture		Fermi	Kepler	Maxwell
Clock Speed(MHz)		1150	706	1050
Total Cores		448 (32 * 16 SMs)	2496 (192 * 12 SMs)	1664 (128*13)
Performance Peak(GFLOPs)		1288	3524	3494
Peak Performance Per Watt		5.7	15.7	24.1
MSRP (when released)		>$3000	>$5000	$329
Cache Size	Register	32768 * 32-bit	65536 * 32-bit	65535 * 32-bit
	Shared Memory	16KB/48KB	16KB/48KB	96KB
	L1 Cache	16KB/48KB	16KB/48KB	24KB
	L2 Cache	768KB	1536KB	1792KB

Validation: streamlines generated by the GPU implementations are validated by comparing to native (CPU-based) MRtrix results. Due to the nature of probabilistic tractography, comparisons of the distributions of the track lengths generated from the same dataset are conducted, and also a Kolmogorov-Smirnov (K-S) statistic test.

Measurements: computation times were measured for multiple spherical harmonic order decompositions of the diffusion data (specifically l_{max} = 4,6,8) and multiple numbers of required streamlines (in the range 1,000 - 1,000,000). Times were measured using the time module in Python to measure the total application elapsed time. Each measurement was repeated twenty times to improve reliability and reduce variance in timings. For the GPU implementations, some overheads, such as copying data across system memory and GPU memory and kernel launching time are also considered as part of the total processing time.

Our principal performance metric is *throughput*, defined as the number of streamlines and number of streamline points that can be computed per unit time interval. In practice, the number of streamlines is a more useful metric than the total number of points on the streamlines. However, streamlines can vary in length significantly, so streamline point throughput is preferred in terms of understanding compute performance.

3.3 Results

In this section, we report the validation and performance measures of CSD-based probabilistic tractography as computed with the GPU implementations we have described, compared to parallel CPU computation with standard MRtrix software. We also discuss the optimization mechanisms applied to the the GPU versions that deliver the reported performance.

3.3.1 GPU Configuration

A GPU application requires careful calibration parameters to achieve the peak performance based on the usage of on-chip registers and shared memory needed by the application and offered on a specific hardware. In our case, the two parallel versions of tractography algorithms are configured on M2070, K20 and GTX970 GPUs. In table 3.3, it shows that 64 threads/block are allocated for the coarse-grained version on all the platforms, along with 56 thread blocks on M2070 and K20, and 104 blocks on GTX970. The coarse-grained mapping also requires a large shared memory space to store the temporary spherical harmonic coefficients, which are 4.096KB, 7.424KB and 11.776KB for lmax value of 4, 6 and 8 respectively. On the other hand, the fine-grained version allocates a small number of threads, 16, in each thread block, in order to achieve a small requirement of shared memory size: 509, 573 and 637 Bytes on each SM. Due to the limited size of shared memory for coarse-grained model and the hardware specified restricts (e.g. the number of thread blocks can be allocated on each SM), the thread occupancies (according to the official occupancy calculator provided by NVIDIA) (NVIDIA, 2010) in all the listed conditions are below 33%. The theoretical occupancies are calculated based on the actual memory usage (e.g. shared memory and registers), GPU thread configurations (e.g. configuration of the number of thread blocks and number of threads within a block), and the actual resources available on the particular architecture (e.g. Fermi, Kepler or Maxwell architectures) and GPU hardware (e.g. M2070, K20 or GTX970).

3.3.2 Validation

CSD-based probabilistic tractography produces a different result every time it is run. While a single random seed could be kept constant for comparisons using the same algorithm, this is not possible to compare CPU and GPU implementations that use different random number generators. Since long runs of order one million tracts are expected to be statistically representative of the connectivity structure of the subject brain, to validate performance it is necessary to undertake statistical comparison of the output of tractography processes against standard results. In this work, our objective is to produce a faster implementation of the MRtrix tractography process, and so we determined to cross-validate outputs of the new versions against output from the standard MRtrix software.

Figure 3.6 shows the fiber length distribution of 1 million tracts using the coarse-grained mapping. The length distribution is defined as the number of total streamlines for a particular fiber length. Most of the streamlines have length shorter than 600 points (i.e. 120mm) and the minority of the tracts are longer fibers. Figure 3.6 also presents the small residuals of coarse-grained model compared to MRtrix, which indicates that the tracts

Hardware	Model	lmax	Thread Per Block	Thread Block Size	Share Memory / Block	Register / Thread	Active Warps / SM	Active Threads / SM	Active Blocks / SM	Active Tracking Process / SM	GPU Occupancy
M2070	Coarse	4	64	56	4.096K	63	16/48	512	8	512	33%
		6	64	56	7.424K	63	12/48	384	6	384	25%
		8	64	56	11.776K	63	8/48	256	4	256	17%
	Fine	4	16	200	509B	56	8/48	256	8	8	17%
		6	16	200	573B	56	8/48	256	8	8	17%
		8	16	200	637B	56	8/48	256	8	8	17%
K20	Coarse	4	64	56	4.096K	124	16/64	512	8	512	25%
		6	64	56	7.424K	129	12/64	384	6	384	19%
		8	64	56	11.776K	124	8/64	256	4	256	13%
	Fine	4	16	200	509B	58	16/64	512	16	16	25 %
		6	16	200	573B	58	16/64	512	16	16	25 %
		8	16	200	637B	58	16/64	512	16	16	25 %
GTX970	Coarse	4	64	104	4.096K	120	16/64	512	8	512	25%
		6	64	104	7.424K	120	16/64	512	8	512	25 %
		8	64	104	11.776K	120	16/64	512	8	512	25 %
	Fine	4	16	150	523B	112	16/64	512	16	16	25 %
		6	16	150	571B	126	16/64	512	16	16	25 %
		8	16	150	651B	136	8/64	256	8	8	19 %

Table 3.3: GPU Configuration and Occupancy Information

produced by our parallel mapping has a similar distribution to MRtrix, but not identical. It is because the different random number generator implementations, and seeding mechanisms (e.g. MRtrix automatically drops the extreme short streamlines e.g. less than 2mm, but our implementations keep the short streamlines which are refined/filtered in the post-process steps in our design). The similar comparison between fine-grained parallel model and MRtrix can be seen in figure 3.7.

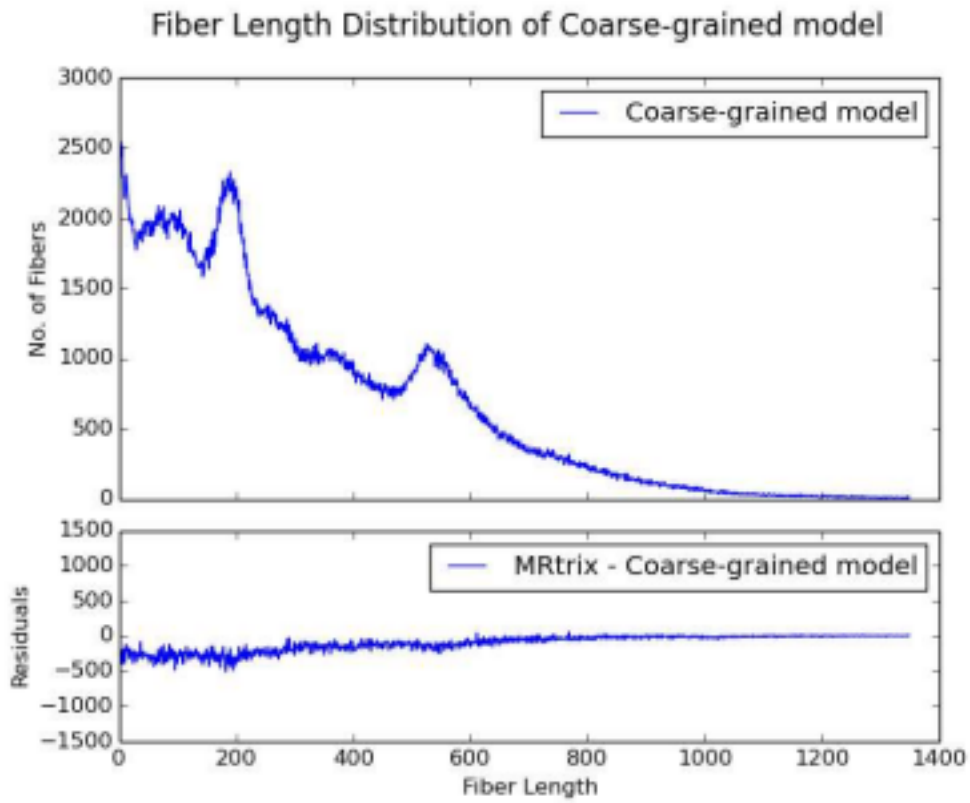


Figure 3.6: Validation by Comparing Tract Length Distribution of Coarse-grained Mapping with MRtrix

We also conduct a Kolmogorov-Smirnov test to see whether the distributions of the streamlines lengths generated from the three methods are from the same population. The basic K-S statistics indicate (nearly) sampling from the same population when it is small (i.e. close to 0). In table 3.4, a 0.03 K-S statistics can be observed between MRtrix and Coarse-grained (and Fine-grained) implementations, which indicates that our implementations perform in a similar way, but we also acknowledge that the streamlines distributions generated by the implementations are not identical. As mentioned above, the differences can mainly be caused by the weak tolerance of short streamlines in our designs and (maybe) insufficient number of samples.

3.3.3 Performance

Figure 3.8 illustrates the time costs of the CSD-based tractography algorithms for the spherical harmonics representation of order 8, on a 6-core CPU and two CPUs with 12 cores, and parallel mapping methods on the three target GPUs. As expected, time costs of

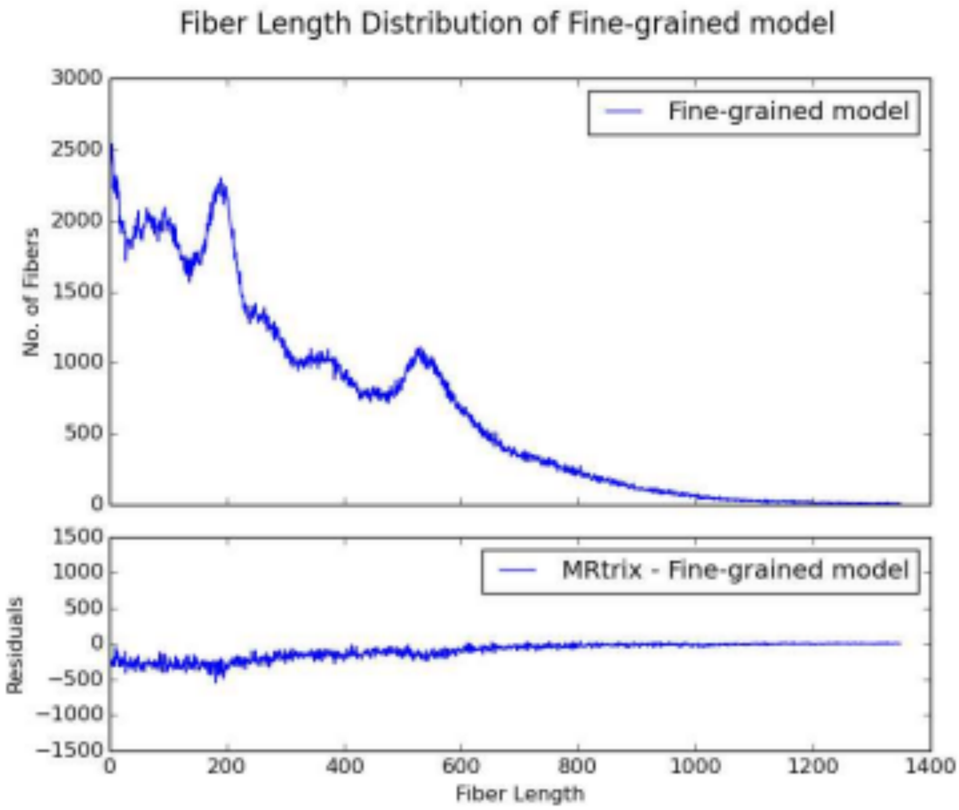


Figure 3.7: Validation by Comparing Tract Length Distribution of Fine-grained Mapping with MRtrix

Table 3.4: K-S Statistic Test for Validation

	MRtrix		Coarse-grained		Fine-grained	
	K-S stats	P-value	K-S stats	P-value	K-s stats	P-value
MRtrix	0	1	0.03	0.59	0.03	0.62
Coarse-grained	0.03	0.59	0	1	0.01	0.99
Fine-grained	0.03	0.62	0.01	0.99	0	1

all the implementations are linearly proportional to the number of tracts. This indicates that tractography problems are linear as discussed in the previous section. Secondly, the 12 core CPU doubles the performance compared to the 6 core CPU. Thirdly, using the parallel mapping methods and GPUs can achieve various levels of speedups than the CPU implementations. The best performance is achieved on coarse-grained parallel mapping on GTX970 GPU, which obtains an 10x+ speedup than the 6 core CPU, taking around 30 seconds to generate 1 million streamlines and its performance is similar on other tested GPUs: M2070 and K20.

More importantly, with newer generation of GPU, the fine-grained mapping and implementation automatically gains performance improvements. In figure 3.8, it shows that time costs on K20 and GTX970 are half that of the M2070 GPU. A roughly 150 seconds are required by running the fine-grained implementation on the M2070 GPU (Fermi architecture) to generate 1 millions probabilistic streamlines, while only around 75 seconds

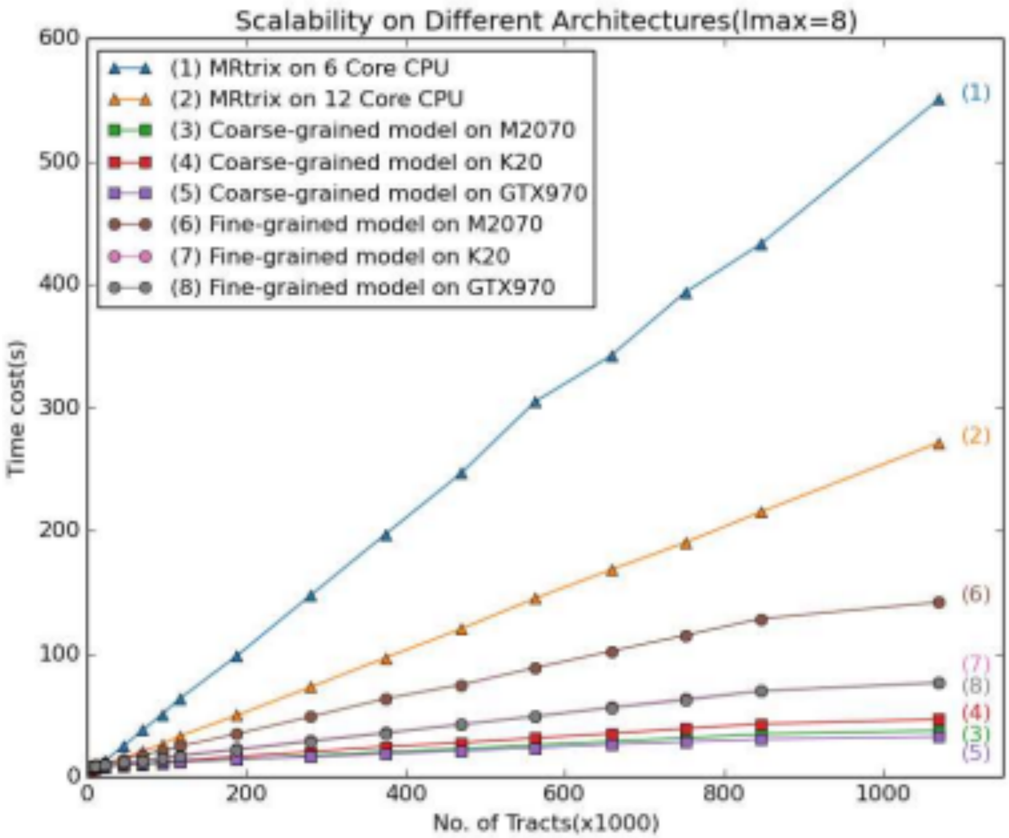


Figure 3.8: Time Costs Comparison on CPUs and GPUs

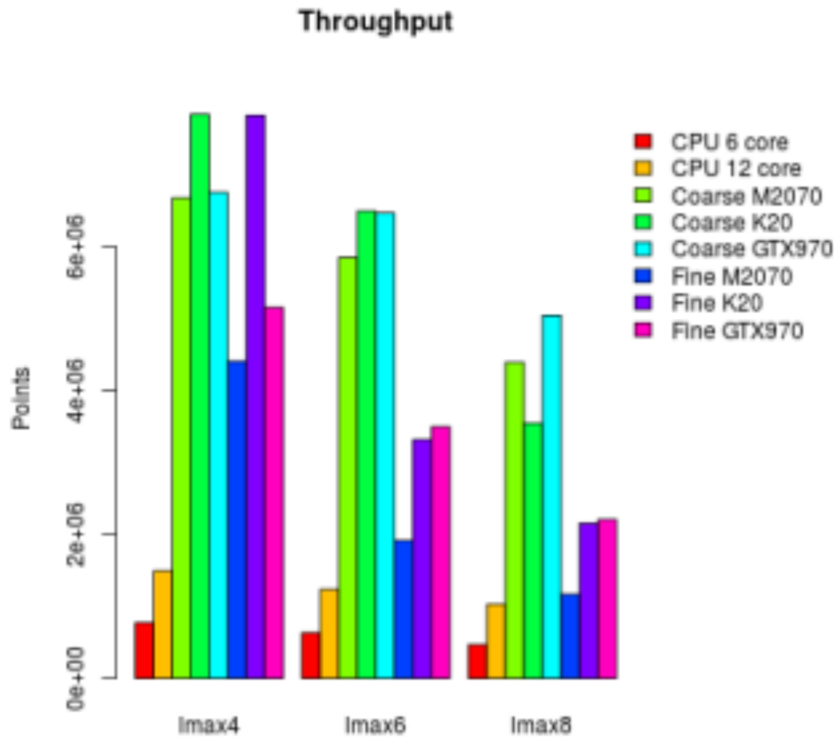


Figure 3.9: Throughputs Comparisons

are needed on K20 (Kepler architecture) and GTX970 (Maxwell architecture). Also it is worth of mentioning that the GTX970 GPU is an average commodity product with less number of CUDA cores, but still has a similar performance of solving the tractography problem in this case.

Figure 3.9 shows the throughput for the different implementations and hardware, under three distinct values of maximal harmonics order (4, 6 and 8). The throughput in question is defined as the number of track points that are generated in one second. A higher throughput corresponds to a higher performance achieved by the algorithm running on the particular hardware. As can be seen, GPU-based parallel algorithms outperformed the CPU version. Generally, linearly decreasing throughputs are observed as lmax values get bigger. In detail, the throughput of 12-core CPU is twice that of the 6-core CPU, which is confirmed with the results shown in Figure. Coarse-grained mapping achieves the highest throughputs on all lmax configurations, almost double the performance of fine-grained version in our measurements. Throughput gradually declines as the lmax value increases, which can be observed on all the implementations and hardware. For lmax=8, coarse-grained version can produce more than 10x speedup compared to the 6 core CPU, while the fine-grained mapping only achieves double throughput on K20 and GTX970 GPUs.

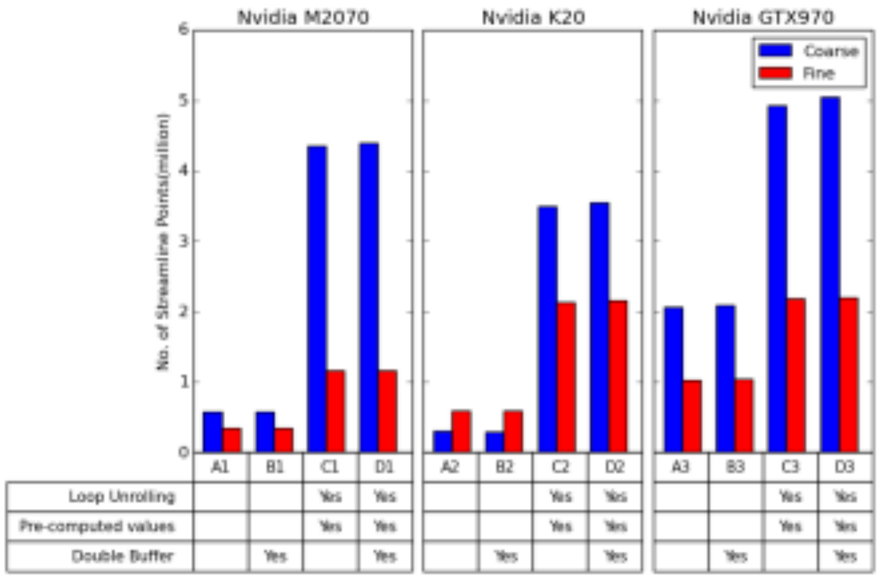


Figure 3.10: Influences of Double Buffer, Loop-unrolling and Pre-computed values

3.3.4 Other Optimizations

In both GPU implementations, we exploit a double buffer mechanism in the parallel mappings, which enables the communication between host and device to be overlapped with processing time on GPU hardware. We unroll the loops within the GPU kernels where practical, and use precomputed values instead of repeatedly computing the same for each iteration. These optimizations dramatically improve performance in the GPU environment. The results shown in figure 3.10, illustrate the throughput of the optimized implementations against the implementation without double buffering, unrolled loops or or pre-computed constants. No significant performance improvement can be observed by using double buffering, compared to the single-buffer implementation on both of the hardware platforms and the both parallel mappings. However, significant performance boosts can be observed in the cases that apply manually unrolling loops in the source code and pre-computed values in the functions of calculating the FOD amplitudes values, which can be seen on the GPUs.

3.4 Discussion

We have presented two parallel mappings of the CSD-based tractography tracking algorithm and implemented them on GPU hardware, realising significant performance improvements. Both the coarse-grained and fine-grained methods achieve higher performance on GPU compared to CPU (e.g. MRtrix).

Across all the algorithms, problem size (lmax value) and hardware, the highest performance is achieved by the coarse-grained mapping running on the K20 GPU for lmax=4. Globally, the coarse-grained solution is reliably faster than the fine-grained solution, with

Table 3.5: Limitation Factors of the Parallel Mappings

	Warps Per Multiprocessor					
Parallel Mapping	Coarse-grained Model			Fine-grained Model		
Hardware	M2070	K20	GTX970	M2070	K20	GTX970
Shared Memory	**8**	**8**	**16**	77	77	154
Register	16	16	16	16	**16**	**16**
Thread Block/SM	**8**	16	32	**8**	**16**	32

only minor variation between the GPU generations (M2070 versus K20 and GTX970). While the fine-grained solution is universally slower (save an impressive performance at lmax=4 on the K20; and despite being devised to increase GPU occupancy) than the coarse-grained version, it does show increased throughput for newer (K20, GTX970) versus older (M2070) generations of GPU. While the newer architectures do not bring further speed-ups for cache-bound parallel designs like the coarse-grained mapping, we tentatively suggest that parallel mappings like the fine-grained design that are designed to require less on-chip cache storage, but involve more frequent between-thread communications, may be well-positioned to achieve higher performance when new generation hardware is applied, because the newer GPU architectures possibly have more cores, lower clock rates and less communication latency.

The parallel mappings executed on the GPU have a higher performance than the CPU implementation, but they remain moderately limited by the hardware constraints, including the size of (fast) shared memory, register capacity and maximal allocatable thread blocks. In table 3.5, we list the known factors that constrain the overall performance, and highlight the most dominant factor(s), showing that the coarse-grained model is predominantly limited by the usage of the shared memory, which results in only 8 warps can be run in a single multiprocessor on M2070 and K20. Limitations are the least on the GTX970, as the shared memory on Maxwell architecture is distinct from the L1 cache and is expanded to 96KB on each SM. But the shared memory size is still insufficient to avoiding frequent global memory access. Conversely, the performance of fine-grained parallel mapping is mainly restricted by the physical hardware constraints, i.e. the number of thread blocks can be allocated to each multiprocessor which is 8 on NVIDIA M2070, 16 on K20 and 32 on GTX970. Low GPU occupancy means majority of GPU cores are idle and not fully used during runtime. This is the main factor that restricts the performance of this fine-grained model. It seems reasonable to expect that the thread management system of future GPUs will be able to schedule more thread blocks on a single multiprocessor, the fine-grain parallel mapping may have a better ability to adapt to newer hardware architecture. Increasing the on-chip shared memory is unlikely to deliver high performance for the implementations presented herein.

Besides hardware constraints, performance is also a strong function of optimizations applied in the implementation, such as using double buffering, pre-computed values and manually unrolling loops. The double-buffer model may not benefit the overall performance, however it is useful when performing real-time rendering in further application. The buffers are used in the parallel mappings is to eliminate the time cost of transferring generated streamlines from GPU back to CPU for storage purpose. A standard PCIe 2.0 interface can achieve a bandwidth of 8GB/s and the data transfer operation in our implementations only happens when the buffer is full. Therefore the ratio between the transfer time and the actual processing time on GPU is very low. However, this feature can definitely benefit the likely situation where the streamlines are needed for immediate display in real-time. For example, the full buffer could be used by an OpenGL shader to perform rendering and the second buffer can be used to continue tractography.

Using pre-computed values and unrolling loops manually in the process of computing FOD amplitudes for next possible streamline positions are proven to improve the performance of the application significantly. As shown in formula 2.7 and 2.8, the computing of FOD amplitudes requires computing the values of the normalization factor $N_{l,m}$ and the spherical harmonic basis functions $Y_{l,m}$. Since these values only depend on SH orders and degrees, i.e. l and m, the use of pre-computed values of these values dramatically save a large number of floating-point calculations. Figure 3.10 illustrates that this program-level optimization is the most effective optimization we have applied to the implementations.

Regarding energy efficiency, GPUs provide much higher performance with less energy consumption than CPUs. Energy efficiency is calculated using the number of floating-point calculations completed per one watt of electricity consumed, known as GFLOPs/watt. In table 4.3 and 3.2, theoretical GFLOPs/watt measures for GPUs (i.e. NVIDIA Tesla M2070, K20 and Geforce GTX970) are more than 4 and 10 times higher than the CPU(i.e. Intel Xeon X5650) respectively. Let us assume that the total number of floating-point operations are the same across MRtrix and the two GPU implementations, which is defined as "totalFLOP", TF. And the Thermal Design Power(TDP) are known for all the hardwares: 95 watt for Xeon X5650 CPU, 225 watt for both of Tesla GPUs and 145 watt for GTX970. Then the ratio of FLOP/watt between the CPU and GPUs can be expressed as:

$$\frac{TF}{95 * t_x} : \frac{TF}{225 * t_m} : \frac{TF}{225 * t_k} : \frac{TF}{145 * t_g} \tag{3.1}$$

The time costs of generating 1 million streamlines under spherical harmonic order of 8 are 640 seconds (CPU,t_x), 37 seconds (Tesla M2070, t_m), 47 seconds (Tesla K20, t_k), and 33 seconds (Geforce GTX970, t_g). Therefore, the ratios of energy efficiencies are about 1.6 : 12.0 : 9.5 : 20.9. Clearly, the GPU implementations can achieve higher energy efficiencies. In particular, the Geforce GTX970 GPU delivers more than 10 times the energy efficiency of the Intel X5650 CPU. Another way of looking into this study is through the performance/dollar. Using a similar formula as 3.1, the performance per dollar can be measured by $TF/(time_{hardware} * price_{hardware})$, which calculates the results as: 1.6 : 9.0 : 4.3 : 91.8. This shows that the latest generation of commodity GPU cards are the best solution of solving parallel tractography algorithms and similar problems, in terms of performance and financial budget. We cannot measure the total floating-point operations directly, due to the probabilistic nature of the tractography problem. Therefore, no actual measurements of peak performance of implementation compared to the hardware peak performance are conducted.

The acceleration of CSD-based probabilistic tractography algorithm for high angular resolution diffusion-weighted images may enable paradigm shifts in this area, such as replacing pre-computed streamline data with real-time tracking and visualization and moving off-line tractography to on-line, more importantly, supporting crossing fibres that the current console software of the MRI scanner system can not provide. In a typical pipeline of tractography analysis and visualization, a region of interest (ROI) is used to generate a number of streamlines and stored them into a file, which then is loaded into visualization tool to display. Once a new ROI is required, the process repeats. In studies that involve a large number of ROIs, this process become time-consuming and tedious. With the software package we implemented, it is capable to generate more than 40,000 streamlines within a second, shown in figure 3.9, which makes it possible to perform interactive, immediate tracking. More importantly, because the solution is implemented on the GPU, it becomes straight-forward to visualize the output on an attached display. Similarly, it is also possible to obtain a large number of streamlines based on probabilistic tractography algorithms immediately after the diffusion-weighted data are acquired in the MRI scanner. This enables more flexibility of adjusting data acquisition parameters, such as magnetic field strength, number of diffusion-weighted directions, etc. Furthermore, the parallel tractography software package can be integrated into more advanced visualization facilities, such as CAVE2 platform. These systems are usually powered by GPUs. By using the large 2D/3D display and their computation resource, the human connectome studies can be conducted more efficiently and cooperatively.

Last but not least, with the new generations and new hardwares announced by the GPU manufacturers (i.e. AMD and NVIDIA), the performance of our accelerated tractography implementation will be improved without any modifications need to be changed to the code itself. The next generation GPU architecture, "Pascal", is recently released by NVIDIA, with new GPU hardware like GTX1080 and Tesla P100, which both have more GPU cores built in (e.g. 2560 cores for GTX1080 and 3584 for P100) and larger and faster memory storage and cache space. AMD will unveil their next generation of GPU family in middle of 2016, namely "Polaris". More importantly, both companies design their product to focus on the energy efficiency (i.e. performance per watt), which definitely will make our accelerated solution more possible to integrate into clinical environment in the near future.

3.5 Conclusion

In the Chapter, we have extended the work of CSD-based probabilistic tracking algorithm (i.e. coarse-grained model) (Raniga et al., 2012), proposed a new mappings (i.e. fine-grained mapping) on GPUs, implemented and optimized the applications on three different generations of GPUs (i.e. Fermi, Kepler and Maxwell), and validated and evaluated the performance in terms of throughput of generating tracts. By using the parallel nature of voxels, the coarse-grained parallel mapping achieves a 10x speedup on GPU compared to a 6 core modern CPU. The time cost of reconstructing one million tracts is less than 30 seconds.. The fine-grained parallel mapping, while slower, appears to be well-positioned to offer improved performance when using future generation of GPUs without any modifications. By parallelizing the tractography algorithm, real-time CSD-based probabilistic tractography becomes possible, which may open new opportunities for brain structural studies, and clinical applications.

Chapter 4

Acceleration of FastICA in fMRI

4.1 Introduction

Functional magnetic resonance imaging (fMRI) allows non-invasive recording of brain activities from humans. Most commonly used fMRI techniques are based on blood-oxygen-level-dependent (BOLD) contrasts, which provides an indirect measure of neuronal activities of human brain. Among the different types of functional MRI techniques, studying neural activation patterns and functional connectivity in "real-time" is becoming popular. It can significantly extend the possibilities of experimental designs and open new ways to study functional components of the brain, like neurofeedback. Neurofeedback is becoming one of the most attractive experiment designs in nowaday neuroscience research, which is an experiment setup that feeds the brain activation information back to the subjects while they are being scanning and ask them to perform self-regulation of their brain functions. This technique has been applied in various types of studies like emotion network (Johnston et al., 2010), and proven to be an effective tool in neuro-therapy, such as pain management (Jensen et al., 2007), Autism treatment (Jarusiewicz, 2002) and treatment of depression (Hammond, 2005).

Conventional neurofeedback designs are mostly hypothesis-based, using predetermined region-of-interests (ROIs) as brain masks. These studies used fMRI activity as a feedback. A number of studies have also started using connectivity as a feedback. These studies also used two regions of interest and provided correlation between regions as a feedback. The main assumption underlying these studies is that functional connectivity is stable. However, it is reported that the hypothesized networks oscillate significantly over time, like network merging and branching often occur during task-based and even task-independent experiments (Chang and Glover, 2010). Even the default mode network (DMN) in the resting-state fMRI studies, which is believed to remain largely unchanged, exhibit dynamics due to other factors, such as memory for recent experience (Tambini et al., 2010). Furthermore, the "noise" and the network instability in the analysis are confirmed by some studies, that contain valuable information of the functional network (Ghosh et al., 2008). However, hypothesis-based approach is not capable of capturing those subtle neural dynamic information.

In contrast, data-driven or hypothesis-free approaches, such as using independent component analysis (ICA) over sliding-window can perform dynamic analysis of fMRI time-series data. Since 2003, many studies have proven that ICA analysis by sliding window approach can provide good accuracy of depicting targeted functional activities in the brain (Esposito et al., 2003), such as robust detection of default mode network in resting-state data (Kiviniemi et al., 2011; Long et al., 2008). And the advantages of using data-driven approach (e.g. ICA analysis by sliding window) are also highlighted to be capable of capturing information of strong network divergence from “stationary” connectivity patterns, and enhancing the understanding of behaviours shifts and adaptive processes, by tracking the functional dynamics of the whole brain (Allen et al., 2012). Hence, this type of algorithms can provide extremely useful information when applied to whole-brain functional data, particularly for real-time applications which is limited by the high computational complexity.

Attempts to achieve real-time ICA analysis by sliding window approach are mostly conducted by either applying to a reduced time window, or using spatial constraints. Early studies to make computational times become compatible with real-time settings, use relative smaller time window, such as 10 time points (Esposito et al., 2003) and 20 time points (Karvanen and Theis, 2004). Compared to that, more recent researches achieve ”real-time” ICA analysis with 60 time points, but focus on particular networks (Kiviniemi et al., 2011), which is similar to studies that use spatial constraints (Wang 2013). A comparison between using static spatial constraints from (off-line) functional localizer and dynamic exploitation of spatial components (e.g. functional networks or regions) from ICA proves that the performance with a priori information is better (Soldati et al., 2013). However, for the purpose of quality assurance and enabling immediate functional localization (Weiskopf et al., 2007), it is still very important to develop techniques that perform real-time whole-brain ICA analysis (which is still believed to be a very computationally challenging task), particularly in the applications of neurofeedback (Weiskopf, 2012).

ICA algorithms are known to be ideal of mapping on parallel computing architectures, as most of the computing time are taken by linear algebra operations: vector-matrix and matrix-matrix operations. Most of the existing software packages that implement ICA algorithms (e.g. Informax-ICA and FastICA), are based on libraries like Basic Linear Algebra Subprograms (BLAS) (Lawson et al., 1979) and Linear Algebra PACKage (LAPACK) (Anderson et al., 1999), using C/C++ programming language. Also, to utilizing the nature of the parallelism in ICA algorithms, a tool known as High Performance Signal Analysis Tools (HiPerSAT) implements informax-ICA in OpenMP (i.e. Open MultiProcessing) for parallel processing, and FastICA using the well-known distributed computing framework: Message Passing Interface (MPI) (Keith et al., 2006). HiPerSAT is now integrated as a built-in tool in the EEGLab software which is a widely used neuroscience applications for analyzing and visualizing EEG brain data (Delorme and Makeig, 2004).

Recently, GPU-based accelerations of ICA algorithms are also being explored, to take advantage of the high performance computing power provided by the GPU parallel architecture. The first attempt is conducted in 2012 and an open source implementation (i.e. CUDAICA) is released (Raimondo et al., 2012). They focus on the informax-ICA, achieve a 25x and 4.5x speedup compared to Libraries: Automatically Tuned Linear Algebra Software (ATLAS) (Whaley and Dongarra, 1998) and Intel Math Kernel Library (MKL) (Intel, 2011), and apply the GPU implementation in EEG analysis. Similar work is published regarding GPU acceleration of another type of ICA algorithm: Joint Approximate Diagonalization of Eigen-matrices (JADE).

However, among the ICA algorithms, FastICA is proven to be converging faster and robust to the type of distributions (Haykin et al., 2009; Langlois et al., 2010). It is supported by a study of comparing 14 ICA algorithms with different window sizes, model orders (e.g. number of extracted components), and priori conditions (Soldati et al., 2013). They highlight that FastICA is one of the algorithms that achieve best performance in terms of robustness to errors in parameters, computational time, and flexibility of using priori information. Therefore, we choose FastICA as the main algorithm to use in our project, and investigate into accelerating FastICA algorithm on GPU, along with several optimization, to achieve nearly real-time sliding-window-based ICA analysis and set up a framework for neurofeedback studies.

4.2 Methods

The idea of functional connectivity is constructed by computing the correlations between voxel-based or region-based BOLD signal patterns acquired using fMRI. It is a way to represent the relationships between the voxels and/or functional regions. Usually, functional connectivity analysis focuses on functional (cortical) regions or ROIs. Conventional approaches of defining regions are via the biological properties of the tissues (e.g. their functional properties, topographic organization, anatomical connection); or some data-driven methods, e.g. region-growing algorithm (Lu et al., 2003). However, both of these methods are categorized as stationary approaches in the way that analyses of this kind are based on the entire time-series data, and more likely to provide summary information regarding the functional data.

Another type of functional connectivity analysis is known to be dynamic via window sliding, which is widely used to observe and explore the functional changes over time. It is recently becoming one of the major frameworks to study brain dynamics both online and offline, especially effective in the context of (online) neurofeedback studies. Along with ICA algorithms, it enables dynamic exploration of functional connectivity and network variations from the online fMRI time-series data. In the context of fMRI, the independent components (ICs) represent either spatial changes over time (Spatial ICA) or temporal activation patterns (temporal ICA). Commonly, studies on fMRI focus on spatial changes, as it is clear to be interpreted.

However, algorithms in fMRI connectivity analysis are normally time consuming, which limits the possibility of real-time applications. Therefore, in the following section, we present a framework based on the idea of sliding window and accelerated ICA (i.e. FastICA) algorithm.

4.2.1 Framework Overview

In our project, we focus on the spatial information changes along the sliding windows to investigate brain activated region patterns, in other words, brain networks or brain states. A brain network is defined as a spatial activation map that indicates the correlations of ROIs (i.e. collections of voxels), which can be either regarded as a brain region or an existing connection of multiple regions (i.e. brain networks). The framework mainly involve two major steps: (i) use ICA to compute the independent components (ICs) on the fly, instead of using pre-defined brain masks (ROIs; and (ii) track the changes of those spatial components, i.e. tracking the brain network changes.

Independent components are computed separately for every sliding windows, given the fMRI time-series data of the whole brain. Figure 4.1 shows an illustration of ICA-based sliding window analysis process. The size of the sliding window and the step size are the parameters can be defined for different experiment settings. For each time window (e.g. with 40 time points), all the voxels (in the brain) are included for the the IC extraction. Before performing ICA, the 4D time-series data need to be realigned into 2D space (i.e. matrix form), in the order of x-, y- and then z-axis shown in figure 4.2. The result of this realignment can be seen as a 2D matrix which includes voxel index in a axis and voxel-wise time-series data in another axis. Then, a full or reduced ICA is operated on the matrix, where a full ICA means the number of extracted ICs equal to the time points of the original signal; and a reduced ICA can extract fewer ICs from the signal. After finish ICA for one time frame, the time window slides forward by a step size, and compute the ICs again. In consequence, ICs for each consecutive time windows, representing functional activation regions or network are extracted.

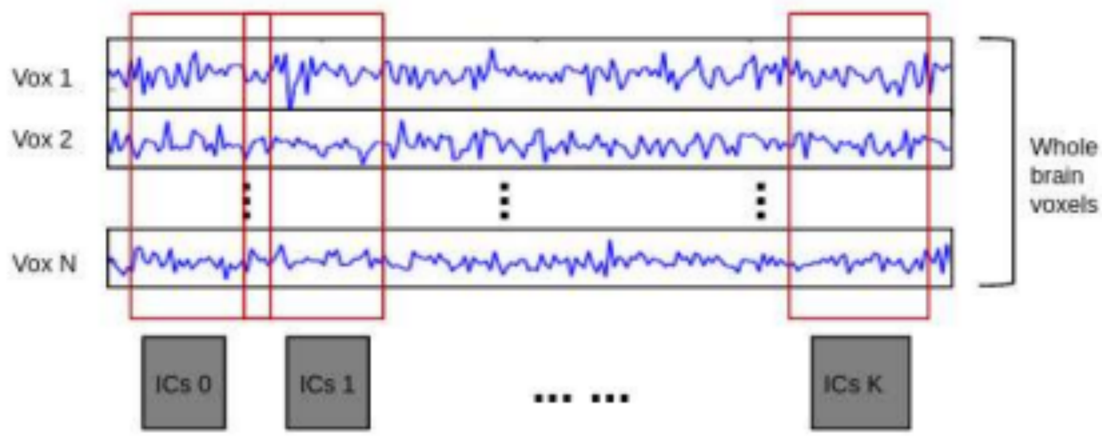


Figure 4.1: Illustration of Sliding Window Functional MRI ICA Analysis

Provided the window-wise ICs that represents functional networks or regions, it is necessary to perform a “tracking” process of those spatial components, due to the unpredictable order of the components extracted from the ICA algorithms. To track the components, the correlations between the ICs in the consecutive time windows can be computed and are used to match the ICs that have high correlation coefficients (and

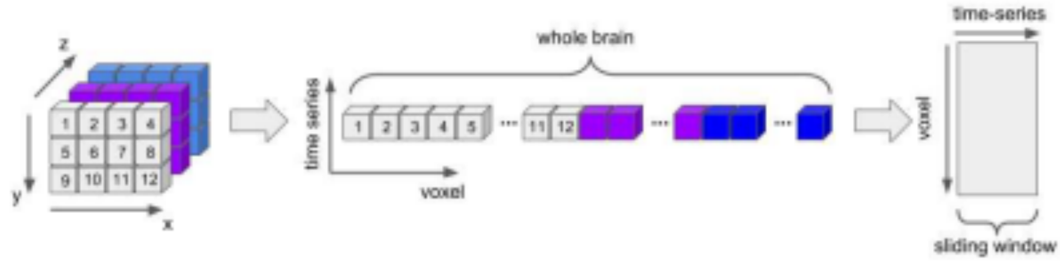


Figure 4.2: Realignment of FMRI Data to 2D Matrix

above a certain threshold), which indicates that they are more likely to be the same ICs across the time window. In this way, the brain activation networks (i.e. ICs) can be tracked and the network dynamics (like network merging and branching) can also potentially be monitored. The process can be seen in figure 4.3, the ICs randomly ordered are matched across time windows during the process of sliding window.

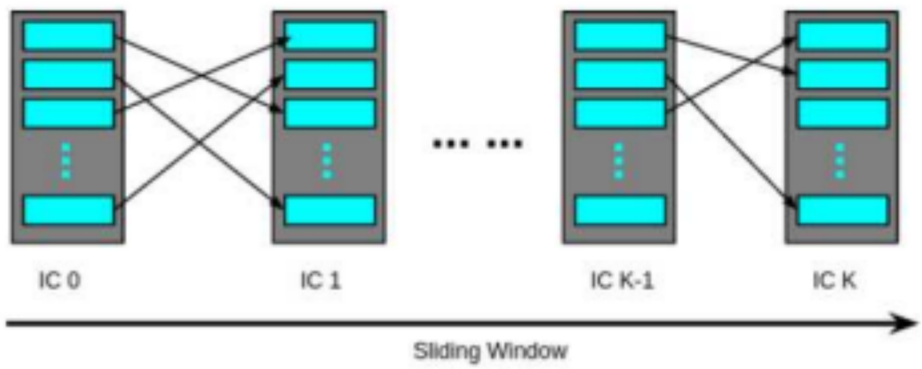


Figure 4.3: Illustration of ICA Components Tracking in Slide Window Analysis

In our project, we are focusing on online ICA component extraction and matching, in other words, aiming for real-time tracking of the spatial independent components, to enable studies such as neurofeedback. However, it is clear that computing iCA is the most computing intensive process, then the problem becomes finding a solution of efficiently solving the computational bottleneck of ICA.

4.2.2 ICA

Independent Components Analysis (ICA) is a type of methods that are developed to extract statistically independent structure information from a given data (Hyvärinen et al., 2004). It is widely used in many applications, such as blind source separation (BSS) (Lee, 2005), image feature extraction (Hoyer and Hyvärinen, 2000), and etc. A well-known scenario of the problem is the “Cocktail Party” (figure 4.4), in which the blue dots represent individual sound sources (e.g. speakers, instruments, and etc), and the red dots are recorders that acquire the sound information from the air. In this case, the recordings from each recorders are a combination of the sources and more specially a linear combination (that can be expressed as a linear mixing function). Given the mixed sound recordings from all the recorders, the purpose of ICA is to estimate a linear un-mixing function that can separate the original sources.

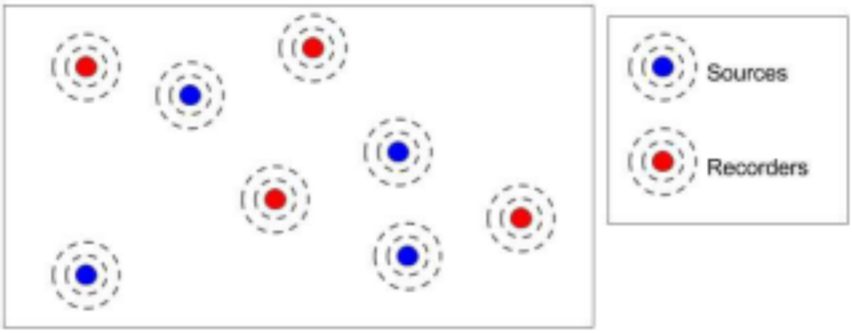


Figure 4.4: Illustration of A Cocktail Party Problem

Mathematically, a typical ICA problem can be defined as finding the components $s_1, s_2, ..., s_n$, given the mixed data $x_j(j = 0, 1, 2, ..., m)$, which can be described as:

$$x_j = a_j^1 s_1 + a_j^2 s_2 + \cdots + a_j^m s_m \quad for\ all\ j \tag{4.1}$$

where a_j^k(k =(1,n)) are the unknown mixing coefficients. It can also be represented in vector-matrix notation as:

$$X = As \tag{4.2}$$

where A is the mixing matrix containing all the mixing coefficients, X is the mixed data and s is the independent components need to estimate. Then, solving ICA problem becomes finding a proper mixing matrix A. Correspondingly, the independent components can be obtained by inverting matrix A (known as the unmixing matrix) and multiplying with the observed data, X, as:

$$s = inv(A)X = W^T X \tag{4.3}$$

where W is the unmixing matrix that can be obtained by a direct or approximate inversion of matrix A.

The basic idea of solving ICA problem is to find a linear combination of components that are regarded as being “most” statistically independent. It is widely accepted that the measurement of 'independence' can be qualified by the concept of non-gaussianity. In other words, the problem of ICA is to find the representation of non-gaussian (or least gaussian) data, i.e. to maximize the non-gaussianity of $W^T X$ in the equation, which seems to capture the essential structure of the data in many applications (Hyvärinen and Oja, 2000).

Then, the problem becomes how to quantitatively measure the non-gaussianity of a variable. One of the classical ways is using the fourth-order cumulant (i.e. Kurtosis). Kurtosis describes whether the distribution of a data is peaked or flat, relative to a normal distribution (Joanes and Gill, 1998). And it is widely used as the measurement of non-gaussianity in ICA. More often, its absolute value represents a greater/smaller non-gaussianity with a large/small value, and zero kurtosis value for a gaussian distribution.

However, Kurtosis is not robust to outliers, which is a significant constraint of using kurtosis to measure non-gaussianity in some applications (Huber, 1985). Alternative, the concept of differential entropy in information theory can also be used to estimate the non-gaussianity. By definition, among all the random variables that have the same variance, a gaussian variable has the largest entropy; and a higher entropy means the variable has a more random distribution (e.g. gaussian distribution) (Papoulis and Pillai, 2002). In practice, a term called "negentropy" (i.e. negative entropy) is used, which is defined as the difference against a gaussian random variables that has the same covariance (Hyvärinen, 1999). However, direct estimation of negentropy is very computationally intensive, which is one of the major issues for real-time and online applications. Therefore, some form of approximation are necessary for computing the negentropy. One of the methods that are proven to be robust to noise and have good performance in ICA is as the following:

$$G(u) = 1/a_1 logcosh(a_1 u) \quad or \quad G(u) = -e^{-u^2/2} \tag{4.4}$$

where a_1 is some constant in the range of 1 and 2. Given the contrast function, a fast fixed-point algorithm for ICA is developed as FastICA to iteratively maximize the non-gaussianity of a distribution (Hyvärinen and Oja, 1997).

4.2.3 FastICA

Given the selected contrast function, FastICA is an algorithm introduced in 1999, to iteratively and rapidly maximize the contrast function (i.e. reach the convergence of the unmixing matrix and obtain the ICs) (Hyvärinen, 1999). This algorithm involves two stages: pre-processing which includes centering and whitening of the data, and the FastICA itself. In the following section, those main steps will be recapped in detail.

The pre-processing techniques: zero-mean centering and whitening the observed variables, can make the estimation of ICA model simpler and better conditioned (Hyviirinen et al., 2001). Centering is usually through subtracting the original data with its mean vector. Whitening is a process that linearly transforms the data so that its components are uncorrelated and have equal unit variances. For example, a common way of performing whitening is via Principle Component Analysis (PCA) or eigenvalue decomposition of the covariance matrix of the observed data (Jolliffe, 2002; Saad, 1992). During the whitening process, dimension reduction on the original data is usually performed, by drop of the eigen-spaces that have relatively small corresponding eigenvalues that are mostly donated by noise. With data centering and whitening, solution can be converged with less iterations, i.e. lower computing time cost.

Assume that the data is preprocessed with centering and whitening techniques, FastICA algorithm can be described as:

- **1**: Choose a random initial vector w_0 of norm 1
- **2**: $w_1 = E\{xg(w^T x)\} - E\{g'(w^T x)\}w$
- **3**: Divide w_k by its norm
- **4**: If not converged(i.e. $|w_k^T w_{k-1}|! \approx 1$), go back to step 2

where the update rule is based on negentropy; the contrast function is a hyperbolic function: tanh, which can be replaced by other options (e.g. cubic and exponential functions); and the criteria of convergence means that the new and old vector point to (almost) the same directions, in other words, their dot production is close enough to 1. This is the procedure of estimating one of the independent components.

In order to estimate several independent components, the above procedure needs to be repeated. To prevent different vectors from converging into a same direction, some technique known as 'decorrelation' operation is essential after each iteration. For whitened data, unmixing matrices become orthogonal, which enables orthogonality-based decomposition like the Gram-Schmidt decomposition (Björck, 1994). After each iteration of w_k, the projections of w_k on the previously vectors, $w_0, w_1, ..., w_{k-1}$ are subtracted from w_k to guarantee the orthogonality and avoid converging into the existing components. Alternatively, a symmetric decorrelation can be used to operate on the entire unmixing matrix after each iteration (Hyvärinen, 1999). One of the methods can be used is the matrix square roots:

$$W = (WW^T)^{-1/2}W \tag{4.5}$$

where W is the unmixing matrix. Normally, the matrix inverse square root can be computed using eigenvalue decomposition.

The pseudocode the FastICA algorithm can be seen in code 1. The inputs to the algorithm are (i) the matrix, X, containing the original observed mixed-data where each row of the matrix is the time-series signals in fMRI data; (ii) number of components extracted from ICA; (iii) choice of contrast function (logcosh function by default); (iv) maximal number of iterations. In pre-processing step, centering is performed by subtracting each element by the mean value of the column it belongs to; and whitening includes eigen-decomposition, followed by normalization (i.e. division of eigenvector by the corresponding eigenvalue, then project the centered data onto the normalized eigen-space with optional dimension reduction. The main iteration starts with a random unmixing matrix followed by symmetric decorrelation; and updates the unmixing matrix using the selected contrast function until it gets converged or fail to converge within a certain number of iterations.

Algorithm 1 Pseudo Code of the FastICA Algorithm

//centering
$X' \Leftarrow X - meancolumn(X)$

//whitening
$eigenvalue, eigenvector \Leftarrow svd(X')$
$norm\_eigenvector \Leftarrow (eigenvector/eigenvalue)$
$X' \Leftarrow dot(norm\_eigenvector'[0 : numcomponents], X')$

//iteration solver
$Wk0 \Leftarrow$ random vector of length numcomponents
$symmetricDecorrelation(wk0)$
while $min[0 : maxiteration]$ **do**
  $wkm + 1 \Leftarrow EX'g(wkmX') - Eg'(wkmX')wkm$
  $symmetricDecorrelation(Wkm + 1)$
  check $Wkm \Leftarrow (Wkm\,Wkm) - 1/2\,Wkm$:
  if $true$ **then**
    reach convergent & return Wkm
  else
    repeat
  end if
end while

4.2.4 Parallel Mapping of FastICA

We analyze and decompose the process of FastICA into a series of individual operations which can be categorized into several types: scalar-vector, scalar-matrix, vector-vector, vector-matrix and matrix-matrix. These linear algebra operations are ideal to be mapped onto parallel computing architectures, like GPU(s). In this section, we present the detailed CUDA implementation of FastICA and related optimizations, with the utilization of several GPU libraries: CUBLAS (Nvidia, 2008), CuSolver (Nvidia, 2008) and CULA (Humphrey et al., 2010), and custom GPU kernels.

In detail, table 4.1 presents all the operations in FastICA algorithm. It is clear that the FastICA is implemented as a hybrid way in which operations on both CPU (C++) and GPU (CUDA) are seamlessly and cooperatively executed. All the computing intensive operations are executed by GPU in a parallel way, which mainly include the linear algebra operations. Especially, we use device routines: Sgesvd and Ssyev from CULA which is an implementation of the Linear Algebra Package (LAPACK) interface for CUDA-enabled GPU(s), to compute the SVD and eigendecomposition which are regarded as the most time costing part in the program. All other matrix/vector operations use the GPU routines from CUBLAS and CUSOLVER libraries provided by NVIDIA. Furthermore, several custom GPU kernels are written.

To take advantage of the CUBLAS and CULA libraries, all the data need to be loaded onto GPU memory. It is important to mention that the memory in CUDA framework is aligned column-major due to historical reason, while the serial part of the application (i.e. C++ code) is row-major. Since CUDA 5.0, unified memory is supported, which means memory will be automatically managed between host and device, i.e. allocated unified memory space can be accessed by both host code and device code efficiently, by declaring the allocated memory using keyword: cudaMallocManaged.

4.3 Results

In this section, the way of validating our parallel implementation is firstly presented with both known sources (i.e. the cocktail party dataset) and unknown source (e.g. fMRI dataset). Then, the performance comparison of the GPU parallel implementation with the CPU version (e.g. numpy implementation) is reported.

4.3.1 Validation

The implementation is validated with the cocktail party dataset, which contains 9 different known sound sources (http://research.ics.aalto.fi/ica/cocktail/cocktail_en.cgi), such as (male/female) speech with different languages, instruments and singing. Each source sound has a length of 6 second with the sample rate of 8000 Hz. In figure 4.5, it shows the sound waves of those sources. Some of them have similar waveforms and share some common features of the sound wave, like source 2 and 4, which are the sounds that a male and female English speakers read news; source 3 and 6 are the same person speak different sentences in Italian; and source 5, 7 and 9 are either singing or music.

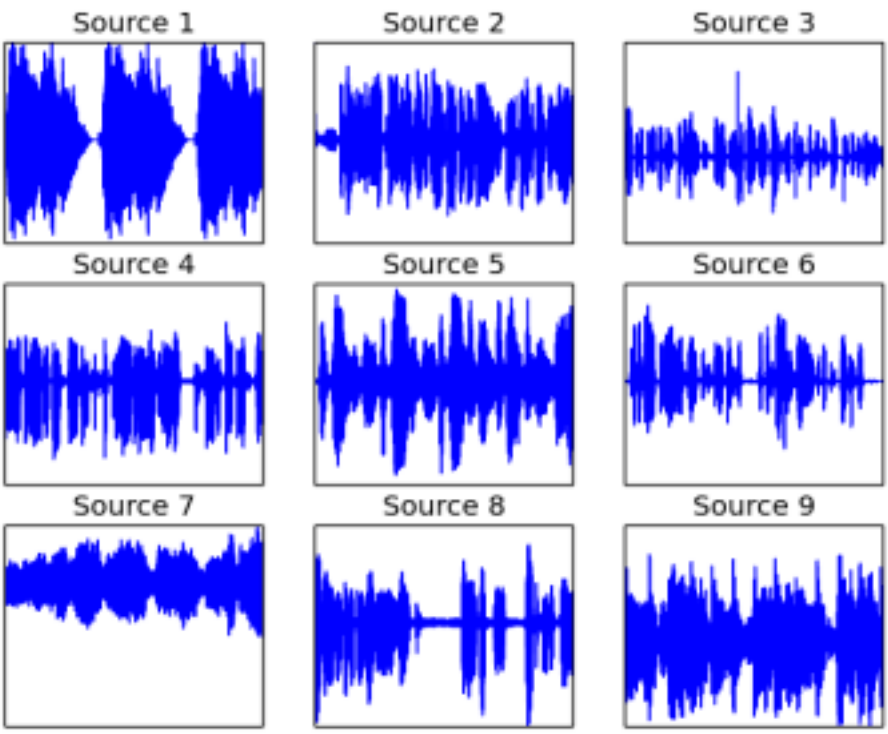


Figure 4.5: Source Signals of The Cocktail Party

The sources are linearly combined with a random mixing matrix, i.e. in the case of the cocktail party, the sources are recorded from 9 different location in the room. The random

Table 4.1: Operations and Their Types in the FastICA Algorithm, and the Corresponding Implementations

Procedure		**Operation**	**Type**	**Function Call**
Preprocess	Data Loading	Read input matrix	General	Custom CPU function
	Centering	Sum of each column	matrix-vector	cublasSgemv
		Get mean of each column	scalar-vector	cublasSscal
		Duplicate mean vector	Custom kernel	DuplicateVector
		Perform centering	vector-vector	cublasSaxpy
	Whitening	SVD of the centered data	SVD	culaDeviceSgesvd ('S', 'S') - reduced SVD
		EigenVector / EigenValue (u/d)	Vector-vector	Custom CPU function
		Projection (Reduce Dimension)	Matrix-Matrix	cublasSgemm
		Normalization / Scale by sqrt(n_com)	scalar-vector	cublasSscal
FastICA	Initialization	Initialize a random unmixing matrix, W	Custom kernel	curand_init, curand_uniform
	Symmetric Decorrelation of Random W	Covariance of W	Matrix-Matrix	cublasSgemm
		Eigen Decomposition of its covariance	Eigen Decomposition	culaDeviceSsyev
		Normalization	scalar-vector	Custom CPU function
		Correlation of new and old eigenvector	Matrix-Matrix	cublasSgemm
		Decorrelation	Matrix-Matrix	cublasSgemm
	Iteration to Convergence	Compute the initialized ICs	Matrix-Matrix	cublasSgemm
		Contrast function: (1) tanh function	Custom GPU function	kernelSquare
		Contrast function: (3) Compute S0 *X^T	Matrix-Matrix	cublasSgemm
		Contrast function: (4) Compute subtraction	Vector-Vector	KernelAminusB (Custom kernel)
		Symm Decorr of the updated W'	Symmetric Decorrelation	same as (iii-1)
		Check convergence: W'*W.T	Marix-Matrix	cublasSgemm
		Check convergence: get max diagonal value	General	Custom CPU function
		Compute ICs: (1) unwhitening W	Matrix-Matrix	cublasSgemm
		ComputeICs: (2) get ICs	Matrix-Matrix	cublasSgemm

mixed data is shown in figure 4.6, each soundwave is mixed with all the sources, and no clear shapes of the original sound waves can be identified easily by visually inspecting the waveform of the mixed data. Furthermore, it is important to mention that to solve an ICA problem, it requires equal to or more mixed data than the number of original sources. For example, in the this case, 9 mixed sound data is acquired.

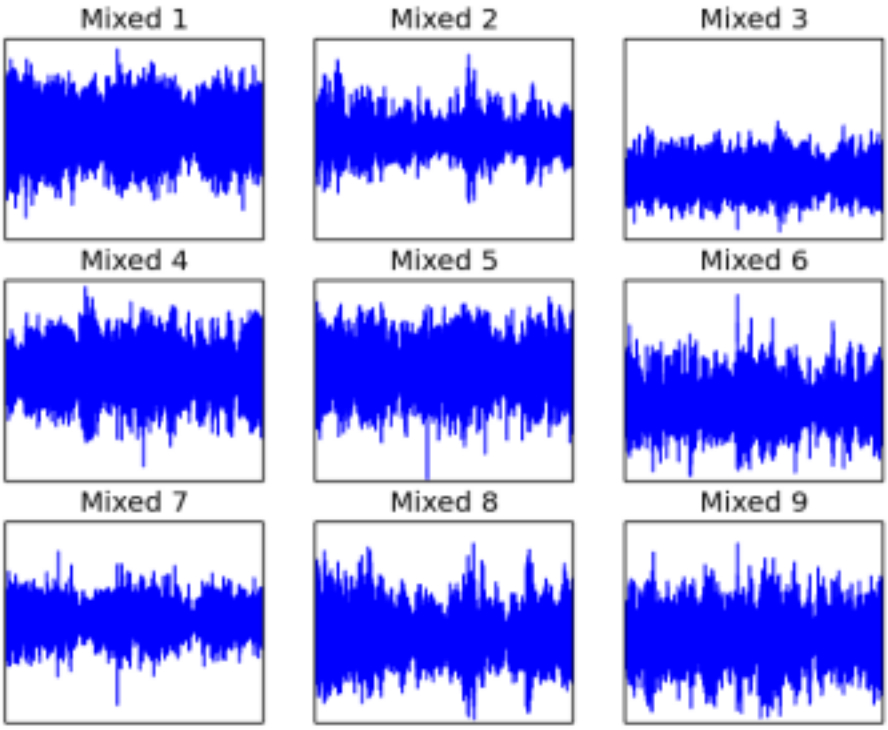


Figure 4.6: Mixed Signals of The Cocktail Party

With our parallel GPU implementation, the ICs are extracted in a random order, due to the nature of the ICA algorithm. In figure 4.7, it presents all the 9 ICs, and they can be clearly matched with the original signals. IC 2, 3, 5 and 8 are almost identical to original source 1, 3, 9 and 5 respectively, and the differences can not be recognized in playbacks. Besides, ICs 1, 4, 6, 7 and 9 are very close to original source 1, 3, 9, 8 and 5, but their signs are reversed which is one of the properties of ICA. Therefore, the implementation is capable of efficiently extracting the targeted independent components in the basic cocktail party dataset.

4.3.2 Performance Measurements

4.3.2.1 Hardware Specifications

A low-end GPU, NVIDIA GTX 750 Ti (table 4.2) is used in the experiments. It is built on the latest Maxwell architecture, with 640 CUDA cores running at the clock speed of 1020 MHz (i.e. 40 cores per SM and 16 SMs in total), which can provides a peak performance of 1305.6 GFLOPs. And it has a separate 24KB L1 cache and 1792 KB L2 cache. It is worth of mention that its market place is around 149 dollars when released. To compare

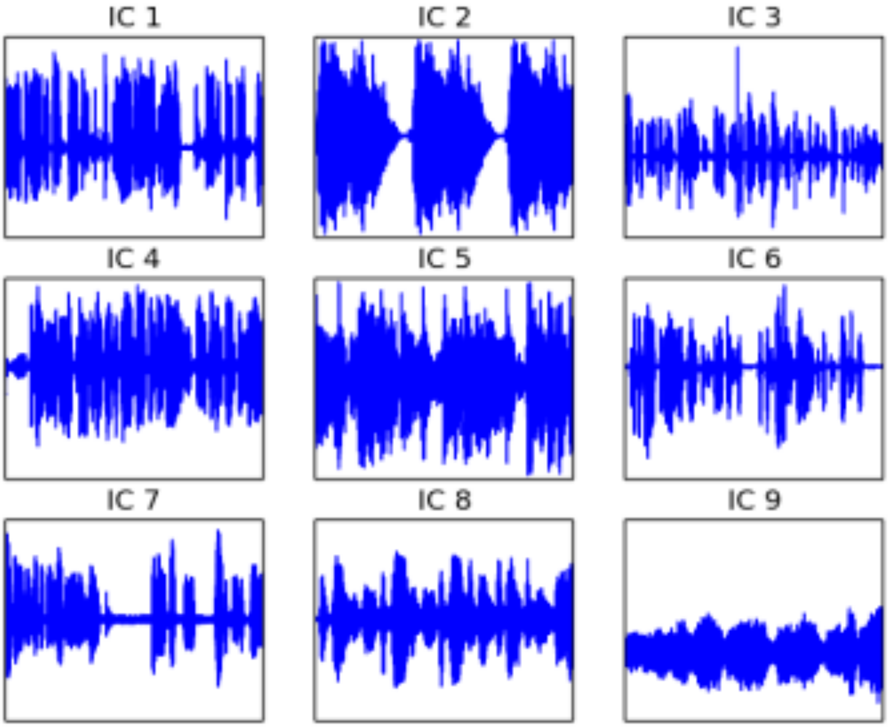


Figure 4.7: Extracted Components from the CockTail Party and Linked with Their Original Sources

the performance, we use a single Xeon X5650 CPU (table 4.3) has 6 cores running at 2.6 GHz, with the peak float precision processing performance of around 124.8 GLOPs. It has a three-level cache architecture, including 32KB L1 cache, 256KB L2 cache and a 12MB L3 cache.

4.3.2.2 Time Cost Distribution

In the cocktail party, after measuring each steps in FastICA implementation, the distribution of the time costs can be seen in figure 4.8. The total processing time of extracting 9 components from the data is around 678.75 ms, which is mostly taken by data loading (i.e. 559.54 ms), 82.4%; while whitening process occupies 12.9% of the time consumption, due to the SVD operation. Other operations, such as centering, unmixing matrix initialization and iterative solver (with an average 6 iterations to convergence) take the remaining 5% of the total time. Within each iteration, roughly 95% of time cost is evenly shared by computing the contrast function (i.e. logcosh in this case) and performing symmetric decorrelation.

The fMRI time-series data used in the experiments are acquired using multi-band sequence, with a spatial resolution of 64x64*42, and a total length of 400 time points with the temporal resolution of 0.754 seconds. The testing data is random selected from one of the time window that is defined to have 40 time points, which results in a matrix in

Table 4.2: GPU Specification

GPU		NVIDIA GTX 750 Ti
Architecture		Maxwell
Clock Speed (MHz)		1020
Total Cores		640 (16 SM x 40 Cores)
Performance Peak (GFLOPs)		1305.6
Peak Performance Per Watt (GFLOPs/Watt)		21.76
MSRP (when released)		$149
Cache Size	Register Per SM	65535*32-bit
	Shared Memory Per SM	96 KB
	L1 Cache	24 KB
	L2 Cache	1792 KB

Table 4.3: CPU Hardware Information

CPU	Intel Xeon X5650
Architecture	X86
Clock Speed(MHz)	2670
Total Core	6 (12 threads)
Perforamnce Peak(GFLOPs)	124.8
L1 Cache	32 KB
L2 Cache	256 KB
L3 Cache	12 MB
Peak Performance Per Watt(GFLOPs/Watt)	1.31
MSRP(when released)	>$1000

the size of 30412x40 (for the voxels in the entire brain). Then, this matrix is fed into the GPU FastICA implementation and perform the IC extraction.

As shown in figure 4.9, extracting 40 components from a time window (whose size is 30412x40) of the fMRI data takes roughly 5.154143 seconds in total. More than 60% time is spent by the iterative solver (e.g. with 72 iterations), and each iteration takes about 40.13 ms in average with a standard deviation of 2.36 ms. Loading the input matrix (whose dimension is 30412x40 and file size is 30MB) from file system to CPU memory and then transferring it to GPU memory take 21.1% of the entire time, around 1.2087 seconds. And 752 ms is occupied by the whitening operation, which is approximately 13%. In each iteration (i.e. 40.13 ms in average), majority of the time, 66.8%, is spent by the operation of symmetric decorrelation (and mainly by the eigen-decomposition operation for about 24.9 ms); the contrast function takes 23.9% of time and the remaining time is shared by other operations, like convergence checking (8.9%).

4.3.2.3 Performance Comparison

Compared to the FastICA routine implemented in Scikit-Learn library (implemented with Numpy which is written in C++ with LAPACK), our GPU implementation shows a reasonable speedup. In the first comparison, the time costs of extracting ICs (i.e. 10, 20, 30 and 40 components) from a fixed time window (i.e. 40 time points) are measured, as well as the number of iterations taken to reach convergence. The used time windows are randomly selected from the fMRI data which contains 30412 voxels over 400 time points. In

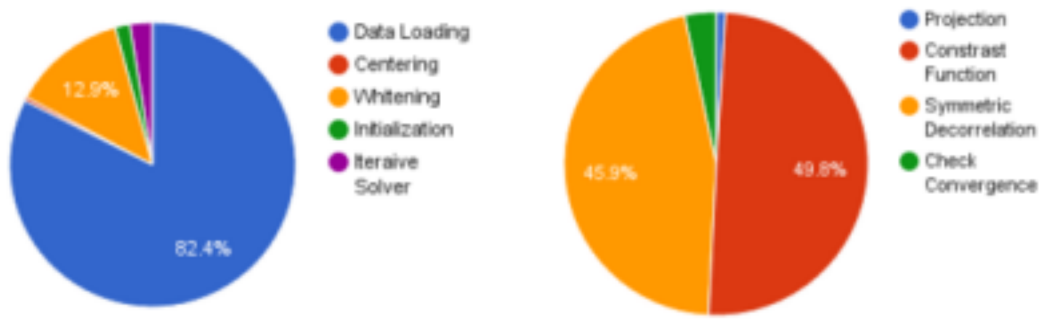


Figure 4.8: The Time Distribution of the GPU Implementation with the Cocktail Party Data

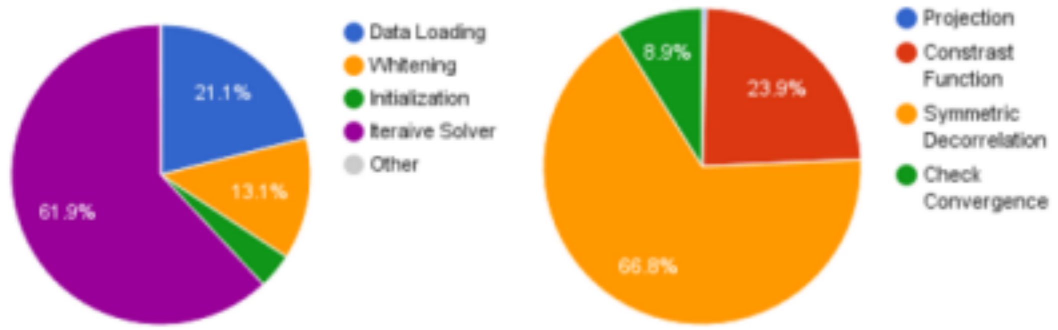


Figure 4.9: The Time Distribution of the Implementation Tested with the FMRI Data

figure 4.10, it shows the comparison results are that (i) the number of iterations increases with the number of extracted components and has a relatively large variation (i.e. the standard deviation of the number of iterations in the figure); (ii) both implementations show a similar convergence stabilities, i.e. they use similar number of iterations to reach convergence; (iii) the GPU implementation outperforms the numpy-based implementation when extracting relatively large number of ICs (i.e. 20, 30 and 40 components). Although it takes longer time of extracting 10 ICs from a window of 40 time points using the GPU implementation (i.e. 1. 32 seconds) than the conventional CPU version (i.e. 0.87 seconds), the GPU implementation is faster by approximately 53.9%, 129.6% and 135.0% respectively, in terms of the total time cost of the entire FastICA process, when extracting larger number of components.

Due to the variety of the number of the iterations, the time costs relative to the iteration numbers are also compared. Figure 4.11 illustrates the average time cost of each iteration for different number of extracted ICs, which is rather stable with small deviation. Similarly, performing ICA extractions with large components on GPU implementation achieves a reasonable speedups: 1.45x, 2.36x and 2.4x, while a 42.5% slower performance is also observed for the extraction of 10 ICs from a 40-points time window. While the time costs of each iteration show a clearly linear increase in the FastICA implemented in the

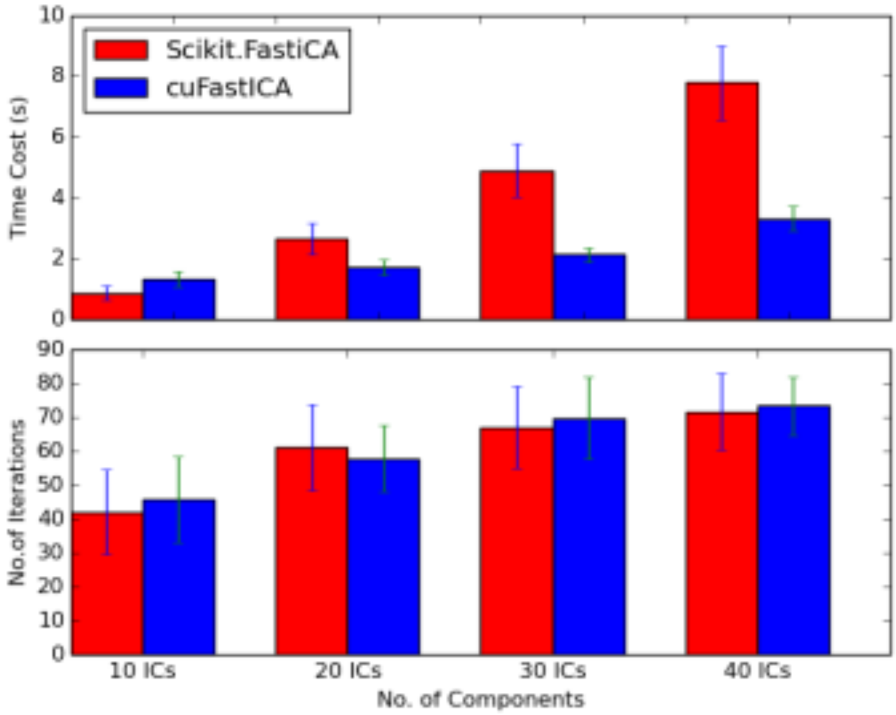


Figure 4.10: Time Cost Comparison of the Two Implementation

Scikit-learn library, i.e. 21.0 ms (10 ICs), 43.7 ms (20 ICs), 73.1 ms (30 ICs) and 108.6 ms (40 ICs), there is no significant time difference for 10, 20 and 30 ICs extraction on GPU, and only an extra 50% time is costed for computing 40 ICs.

Besides, the experiments to compare the scalability of the two implementations are designed to measure the time cost of different size of time windows (i.e. from 10 to 100 time points), which results in the matrix size of 30412x10 up to 30412x100. In this case, equal number of ICs are extracted, as the widths of the time window. For example, 10 ICs are computed from a window of 10 time points; 20 ICs are computed from a 20-time-point window, and etc. For each window size, the time are measured across 10 different sliding windows by a step size of 20. And an averaged time is calculated to represent the time cost of the window with the particular size. Results in figure 4.12 shows (i) a consistently linear increase of time costs of FastICA in Scikit-learn; (ii) extremely efficient computing of full ICA for small time windows, such as widths of 10, 20 and 30 (3.32 ms, 4.12ms and 5.08 ms respectively); (iii) a relatively small increase of time costs on cuFastICA implementation. A significant time increase can be seen, when computing the full ICA analysis with larger window than 30 time points, especially it takes more than 10 times longer to process the full ICA with a window width of 40, compared to 30-time-point window. And slight rises of time consumptions can be seen for full ICA of 40-, 50- and 60-time-point windows, similarly for the window widths of 70, 80 and 90; and 44.5% more time is costed by computing full ICA (with 70 window width), compared to 60, and 29.8% longer time is taken by computing 100 ICs from a full ICA, as to 90 ICs.

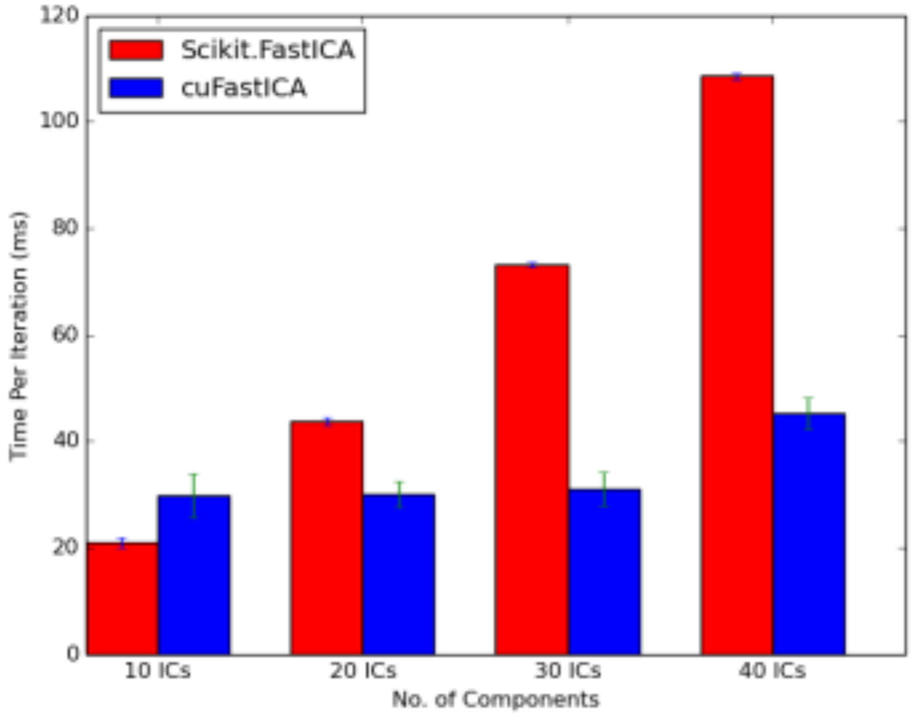


Figure 4.11: Performance Compared to Numpy (Time Cost Per FastICA Iteration)

4.4 Discussion

Overall time consumption of FastICA algorithm and its implementation is significantly affected by the number of iterations towards unmixing convergence. The main factors related to the speed of convergence are: (a) the size of the selected time frame; (b) Number of ICs need to extract; (c) choice of contrast function; (d) selection of initial unmixing matrix. In figure 4.13, it shows that the number of iterations in full ICA analysis on both implementations increases gradually, with the width of the time window getting larger (from 10- to 100-time points). For example, roughly 80 iterations are required by algorithm to generate more than 70 ICs. Both of the implementations have a very stable trend of convergence, i.e. given the same time window, they reach convergence after the similar number of iterations, even though each iteration starts with a random unmixing matrix. And in the following discussion, we will also have an investigation of the influences of the selection of the initialized unmixing matrix on the convergence speed in future work.

A simulation is conducted as a proof of concept to the framework proposed in this chapter. It depicts the process of extracting ICA components on consecutive sliding windows and tracking the components across the time windows. The time-series data (of 400 temporal samples) used in this simulation is from 10 resting-state fMRI scans, with the temporal resolution of 0.75 seconds (i.e. TR is 0.75 seconds) with multi-band acquisition. and the spatial resolution of the dataset is 64x64x42. The size of time window is 40 time points and step size is 1. Within each time window, the ICA analysis is performed independently on each of the scans and extract 10 components. These components are tracked

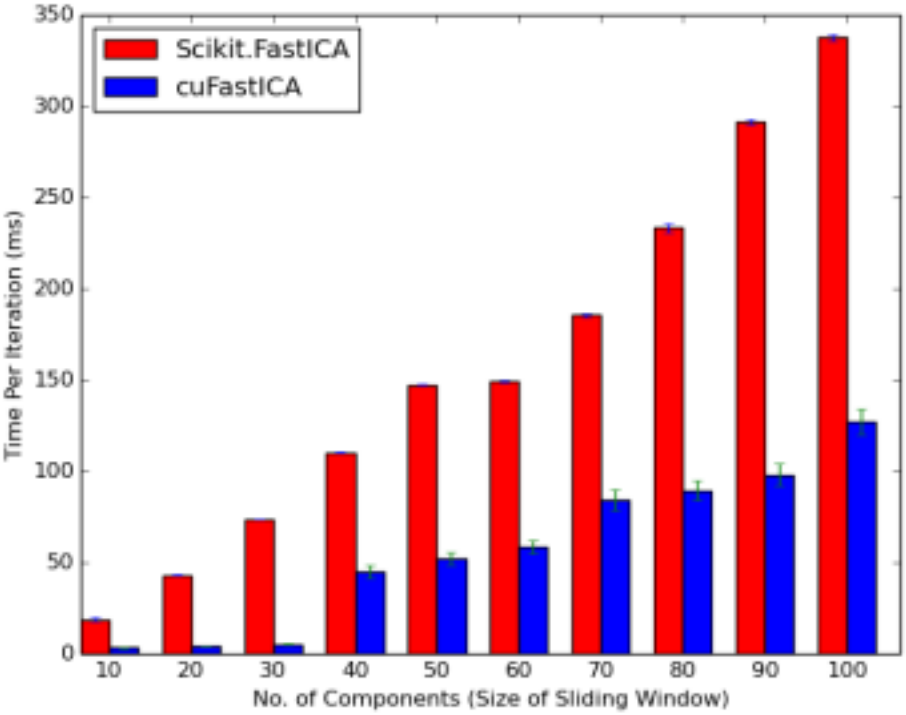


Figure 4.12: Illustration of Scalability Comparisons between the Two Implementations

across sliding windows by computing the correlations between the ICs on two consecutive windows. In figure 4.14, it shows a screenshot of the simulation. The selected ICs are overlapped onto 10 slices of the scan, in which the activation areas are highlighted with positive (red color) and negative (blue color) correlations. Although it is a demonstration of the sliding process processed off-line, the similar pipeline can be applied in the future on-line studies by integrating the accelerated GPU-based FastICA implementation.

Beyond the demonstration of the framework in the simulation discussed above, the main purpose of this work is to achieve real-time on-line ICA-based functional studies. To accomplish that, the entire process time including data acquisition, transferring, pre-processing (e.g. smoothing, denoising) and the ICA computing, is no longer than 1 TR which is typically 2 seconds for conventional MRI acquisition. For example, in the case of extracting 10 ICs from a time window of 30 time points, the time cost of performing ICA is only 0.11 seconds, which then can be applied in real-time applications. However, for multiple band sequence (0.75 second for one TR), it is still a very challenging task. Besides, the GPU used in the tests are a low-end commodity GPU: NVIDIA GTX 970. When more advanced GPUs are used, such as the NVIDIA Tesla series that contains more GPU processing units and is designed for floating-point scientific computing, the performance of the parallel FastICA implementation can be improved accordingly.

Also, the acceleration of FastICA can facilitate the data-driven fMRI studies of network dynamics in a faster way. Most of the current functional network studies are hypothesis-driven, i.e. use pre-defined network regions or ROIs. However, it limits the exploration of new functional network patterns and its dynamics, such as network branching and

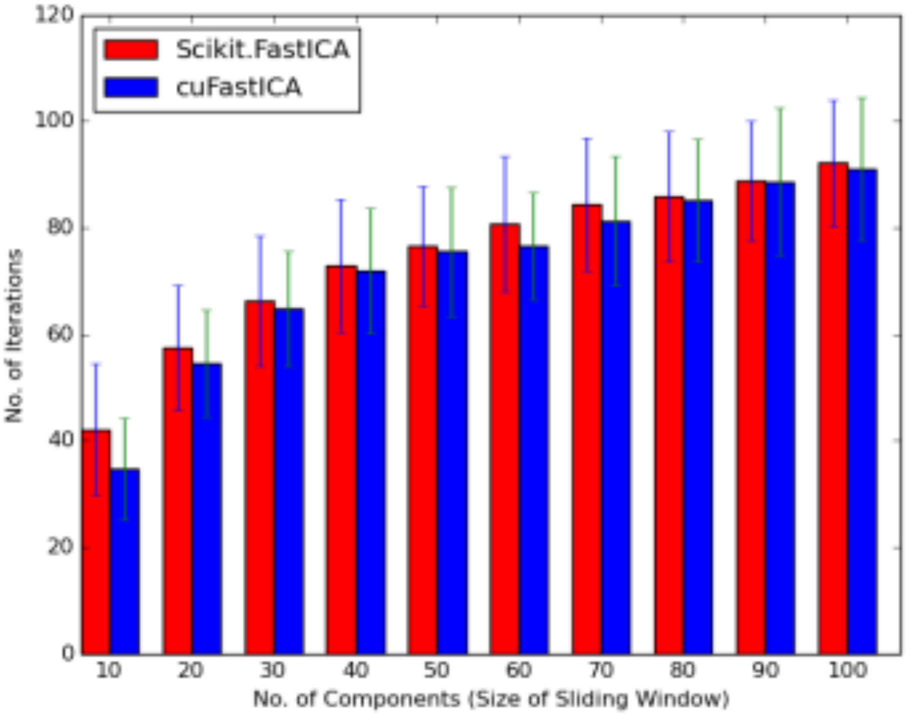


Figure 4.13: Stability of Convergence in FastICA

merging. Using the fast GPU implementation, it is applicable to extract a large number of ICs from relatively wide time window within a reasonably short period of time, which is not achievable for conventional processing pipeline. Moreover, in the applications like neurofeedback, there has no real-time ICA-based studies, where the work proposed in the chapter aims to tackle.

4.5 Conclusion

In this chapter, we have presented a framework of using ICA-based sliding window approach (i.e. FastICA) to exploring functional network dynamics. We have accelerated the main component in the pipeline, i.e. the FastICA algorithm, on GPU and achieved a reasonable speedup, compared to conventional CPU-based software packages. The main contribution is that the work presented here has the potential of achieving real-time data-driven exploration of fMRI data and applying in areas like neurofeedback.

Integrate the ICA-based approach in real-time with MRI acquisition techniques like multi-band is still a challenging task. Therefore, we are planning to further reduce the processing time of the FastICA algorithm by minimizing the number of iterations to reach convergence. This may be done by exploring the overlapped area of the consecutive windows in fMRI data (i.e. more than 90% areas are overlapped in the sliding window approach), and adjust the initial unmixing matrix of the second window, according to the final unmixing matrix from the previous window. In this way, the iterations of reaching

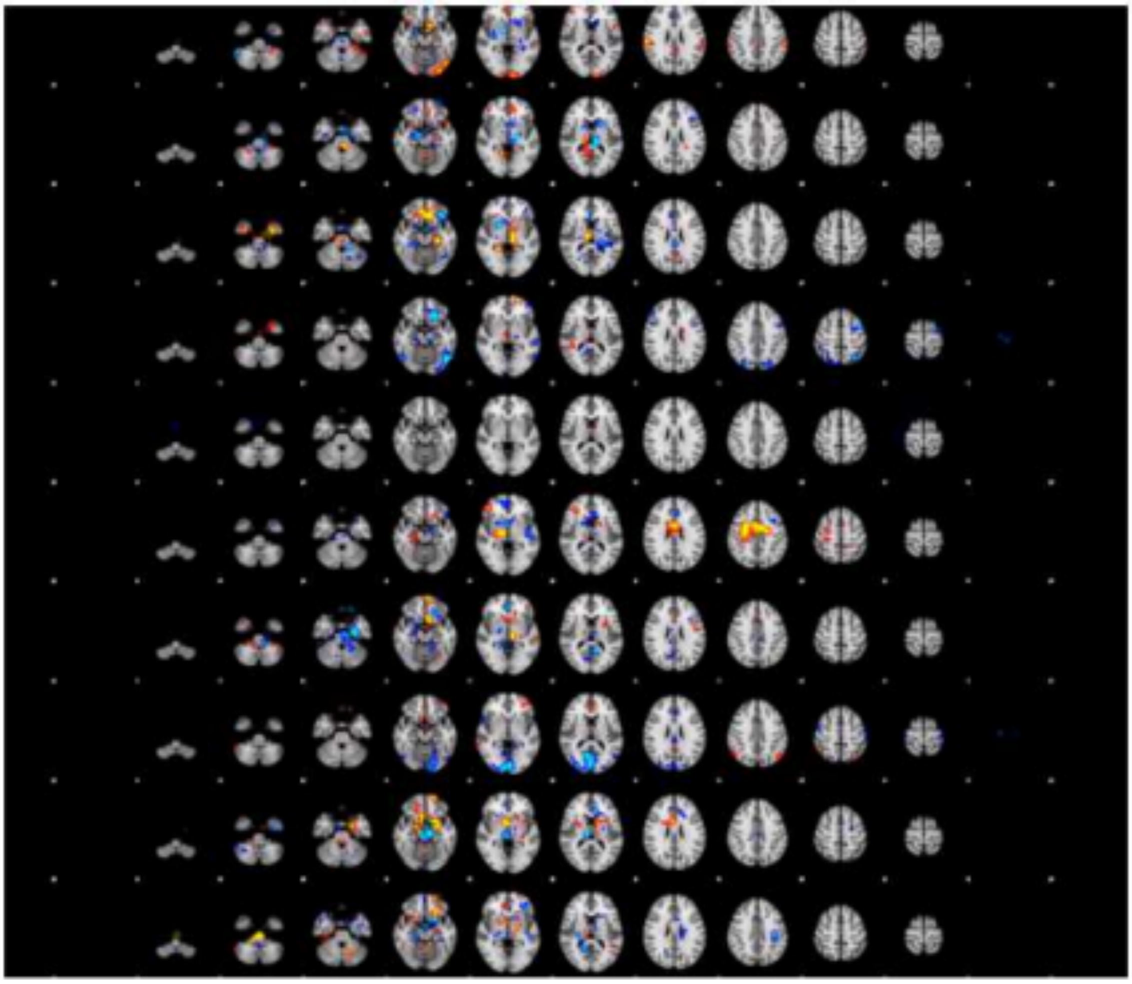


Figure 4.14: A Screenshot of ICA-based Sliding-window Approach in fMRI Time-series Data

convergence can potentially be reduced significantly. Therefore, this is the next step in our work, towards achieving "real-time" ICA-based functional network studies.

Chapter 5

Conclusion and Future Work

In this work, we accelerated the algorithms in CSD-based diffusion tractography (with high spherical harmonic representations) and functional MRI connectivity analysis (i.e. mainly ICA analysis of fMRI time series data), by mapping and implementing them on the GPU architectures. Two parallel mappings (i.e. coarse- and fine-grained mappings) are presented, which have different properties and therefore are applicable for different GPU architectures. The coarse-grained mapping is designed for old architectures like Fermi, while the fine-grained mapping is for current (Kepler and Maxwell architectures) and future architectures e.g. Pascal, and self-tuned for performance improvement without changing the implementations. A maximal 60x speedup is achieved in our implementation compared to one of the standard software packages, MRtrix. And a reconstruction of 1 million streamlines (based on CSD model) can be completed within 20 seconds using our implementation on a commodity GPU hardware, i.e. a throughput of approximately 40,000-50,000 streamlines can be generated within 1 second. This may make the real-time (high order model-based) interactive tractography feasible to use in research and clinical environment e.g. intraoperative MRI image-guided neurosurgery.

Also, the work on accelerating sliding-window-based spatial ICA tracking on functional MRI time-series data can be regarded as a replacement of the traditional (ROI-based) approach to study brain functional dynamics. We map the FastICA algorithm on the GPU platform and achieve less than 2 seconds of completing the ICA process for a 40 time-point time window whole-brain spatial independent components extraction, which is below the normal acquisition speed of fMRI sequences. This potential enables a paradigm shift in fMRI experiment design, and open new possibilities of performing real-time neurofeedback studies and clinical therapies.

It is worth noting that, to our knowledge it was the first few attempts to accelerate the CSD-based probabilistic tractography and FastICA algorithm using GPU(s), and achieve a reasonable speedup. On the basis of the results reported so far, this thesis opens new interesting perspectives for real-time interactive tractography and experimental neurofeedback studies. At the same time, it becomes feasible to combine these two techniques and integrate into intraoperative image-guided neurosurgery. Further research will be carried out as the follows:

- **1**: The implementation of the GPU accelerated CSD-based probabilistic tracking algorithms will be refined to support GPU on-chip random number generator, as the RNG function was not supported by CUDA framework at the time we implemented the methods. We will finalize the codes and push the code back to MRtrix as part of the MRtrix project or a separate GPU version, under General Public (GNU) licence.
- **2**: The accelerated tractography codes (both coarse- and fine-grained mappings) will be integrated into the ImageHD project (mainly for the Huntington's Disease) which provides real-time fiber construction and visualization in Monash CAVE2 facility.
- **3**: The refined version of the GPU-accelerated FastICA implementation will be donated to the scikit-learn (one of the most widely used open source machine learning library in Python), CULA libraries (i.e. a GPU-accelerated libraries for linear algebra) and/or NVIDIA's CUBLAS library where FastICA has not been implemented.
- **4**: With the availability of the GPU version of the FastICA algorithm, we are planning to set up and integrate the code into the neurofeedback pipeline in Monash Biomedical Imaging (MBI), Monash University. Besides, a pilot neurofeedback study is under consideration, using the real-time ICA-based fMRI methods.

Appendix A

Glossary

WM	White Matter
DW	Diffusion Weighted
DTI	Diffusion Tensor Imaging
GPU	Graphic Processing Unit
MR	Magnetic Resonance
ROI	Region of Interest
fMRI	Functional Magnetic Resonance Imaging
ICA	Indepedent Component Analysis
CT	Computed Tomography
MRI	Magnetic Resonance Imaging
dMRI	Diffusion Magnetic Resonance Imaging
NF	Neurofeedback
GPGPU	General Purpose Graphic Processing Unit
3D	Three-dimensional
CUDA	Compute Unified Device Architecture
OpenCL	Open Computing Language
CPU	Central Processing Unit
API	Application Programming Interface
OS	Operating System
ALU	Arithmetic Processing Unit
SM	Streaming Multi-processor
DPU	Double-precision Processing Unit
SFU	Special Function Unit
DRAM	Dynamic Random Acess Memory
SIMD	Single Instruction Multiple Data
SP	Streaming Processor
GM	Grey Matter
CSF	Cerebrospinal Fluid
SNR	Single Noise Ratio
RF	Radio Frequency

NMR	Nuclear Magnetic Resonance
FID	Free Induction Decay
TE	Echo Time
TR	Time of Repetition
HARDI	High Angular Resolution Diffusion Imaging
ADC	Apparent Diffusion Coefficient
SD	Spherical Deconvolution
CSD	Constrained Spherical Deconvolution
FOD	Fiber Orientation Distribution
PDF	Probablisty Density Function
SH	Spherical Harmonics
BOLD	Blood-Oxygen-Level-Dependent
EPI	Echo-planar Imaging
GLM	General Linear Model
DMN	Default Mode Network
RSN	Resting State Network
IC	Independent Component
EEG	Electroencephaslogram
LCRNG	Linear Congruential Random Number Generator
GLOPs	Gigibype of FLoating-point per Second
MPI	Message Passing Interface
BLAS	Basic Linear Algebra Subprograms
LAPACK	Linear Algebra Package
BSS	Blind Source Seperation
SVD	Singlar Value Decomposition
MBI	Monash Biomedical Imaging
GNU	General Public Licence

References

Alexander, A. L. (2010). Deterministic white matter tractography, *Diffusion MRI: Theory, methods, and applications* pp. 383–395.

Allen, E. A., Damaraju, E., Plis, S. M., Erhardt, E. B., Eichele, T. and Calhoun, V. D. (2012). Tracking whole-brain connectivity dynamics in the resting state, *Cerebral cortex* p. bhs352.

Amdahl, G. M. (2013). Computer Architecture and Amdahls Law, *Computer* **46**(12): 38–46.
URL: *http://dx.doi.org/10.1109/mc.2013.418*

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D. (1999). *LAPACK Users' Guide*, third edn, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Arns, M. and Kenemans, J. L. (2014). Neurofeedback in ADHD and insomnia: Vigilance stabilization through sleep spindles and circadian networks, *Neuroscience & Biobehavioral Reviews* **44**: 183–194.

Assaf, Y. and Pasternak, O. (2008). Diffusion tensor imaging (DTI)-based white matter mapping in brain research: a review, *Journal of molecular neuroscience* **34**(1): 51–61.

Atkinson, K. and Han, W. (2012). *Spherical harmonics and approximations on the unit sphere: an introduction*, Vol. 2044, Springer Science & Business Media.

Bargmann, C. I. and Newsome, W. T. (2014). The Brain Research Through Advancing Innovative Neurotechnologies (BRAIN) Initiative and Neurology, *JAMA Neurology* **71**(6): 675.
URL: *http://dx.doi.org/10.1001/jamaneurol.2014.411*

Basser, P. J. and Jones, D. K. (2002). Diffusion-tensor MRI: theory, experimental design and data analysis–a technical review, *NMR in Biomedicine* **15**(7-8): 456–467.

Beaulieu, C., Plewes, C., Paulson, L. A., Roy, D., Snook, L., Concha, L. and Phillips, L. (2005). Imaging brain connectivity in children with diverse reading ability, *Neuroimage* **25**(4): 1266–1271.

Beckmann, C. F., Jenkinson, M. and Smith, S. M. (2003). General multilevel linear modeling for group analysis in FMRI, *Neuroimage* **20**(2): 1052–1063.

Behrens, T., Berg, H. J., Jbabdi, S., Rushworth, M. and Woolrich, M. (2007). Probabilistic diffusion tractography with multiple fibre orientations: What can we gain?, *Neuroimage* **34**(1): 144–155.

Behrens, T., Woolrich, M., Jenkinson, M., Johansen-Berg, H., Nunes, R., Clare, S., Matthews, P., Brady, J. and Smith, S. (2003). Characterization and propagation of uncertainty in diffusion-weighted MR imaging, *Magnetic resonance in medicine* **50**(5): 1077–1088.

Bello, L., Castellano, A., Fava, E., Casaceli, G., Riva, M., Scotti, G., Gaini, S. M. and Falini, A. (2010). Intraoperative use of diffusion tensor imaging fiber tractography and subcortical mapping for resection of gliomas: technical considerations, *Neurosurgical focus* **28**(2): E6.

Bengio, Y. (2009). Learning deep architectures for AI, *Foundations and trends® in Machine Learning* **2**(1): 1–127.

Biswal, B., Zerrin Yetkin, F., Haughton, V. M. and Hyde, J. S. (1995). Functional connectivity in the motor cortex of resting human brain using echo-planar mri, *Magnetic resonance in medicine* **34**(4): 537–541.

Björck, Å. (1994). Numerics of gram-schmidt orthogonalization, *Linear Algebra and Its Applications* **197**: 297–316.

Brenner, D. J. and Hall, E. J. (2007). Computed tomographyan increasing source of radiation exposure, *New England Journal of Medicine* **357**(22): 2277–2284.

Bullmore, E. and Sporns, O. (2009). Complex brain networks: graph theoretical analysis of structural and functional systems, *Nature Reviews Neuroscience* **10**(3): 186–198.

Calhoun, V. D., Adali, T., Hansen, L. K., Larsen, J. and Pekar, J. J. (2003). ICA of functional MRI data: an overview.

Calhoun, V. D., Kiehl, K. A. and Pearlson, G. D. (2008). Modulation of temporally coherent brain networks estimated using ICA at rest and during cognitive tasks, *Human brain mapping* **29**(7): 828–838.

Calhoun, V. D., Liu, J. and Adalı, T. (2009). A review of group ICA for fMRI data and ICA for joint inference of imaging, genetic, and ERP data, *Neuroimage* **45**(1): S163–S172.

Chang, C. and Glover, G. H. (2010). Time–frequency dynamics of resting-state brain connectivity measured with fMRI, *Neuroimage* **50**(1): 81–98.

Clatworthy, P., Williams, G. B., Acosta-Cabronero, J., Jones, S., Harding, S., Johansen-Berg, H. and Baron, J.-C. (2010). Probabilistic tractography of the optic radiationsan automated method and anatomical validation, *Neuroimage* **49**(3): 2001–2012.

Cook, E. H. (1990). Autism: review of neurochemical investigation, *Synapse* **6**(3): 292–308.

Daga, P., Winston, G., Modat, M., Cardoso, M. J., White, M., Mancini, L., McEvoy, A. W., Thornton, J., Micallef, C., Yousry, T. et al. (2011). Improved neuronavigation through integration of intraoperative anatomical and diffusion images in an interventional MRI suite, *Information Processing in Computer-Assisted Interventions*, Springer, pp. 168–178.

Damoiseaux, J. S. and Greicius, M. D. (2009). Greater than the sum of its parts: a review of studies combining structural connectivity and resting-state functional connectivity, *Brain Structure and Function* **213**(6): 525–533.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters, *Communications of the ACM* **51**(1): 107–113.

Delorme, A. and Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis, *Journal of neuroscience methods* **134**(1): 9–21.

Egner, T. and Gruzelier, J. H. (2003). Ecological validity of neurofeedback: Modulation of slow wave EEG enhances musical performance, *Neuroreport* **14**(9): 1221–1224.

Elam, J. S. and Essen, D. V. (2013). Human Connectome Project, *Encyclopedia of Computational Neuroscience*, Springer Science Business Media, pp. 1–4.
URL: *http://dx.doi.org/10.1007/978-1-4614-7320-6*$_5 92 - 1$

Enislidis, G., Wagner, A., Ploder, O. and Ewers, R. (1997). Computed intraoperative navigation guidancea preliminary report on a new technique, *British Journal of Oral and Maxillofacial Surgery* **35**(4): 271–274.

Enquobahrie, A., Cheng, P., Gary, K., Ibanez, L., Gobbi, D., Lindseth, F., Yaniv, Z., Aylward, S., Jomier, J. and Cleary, K. (2007). The image-guided surgery toolkit IGSTK: an open source C++ software toolkit, *Journal of Digital Imaging* **20**(1): 21–33.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P. and Bengio, S. (2010). Why does unsupervised pre-training help deep learning?, *The Journal of Machine Learning Research* **11**: 625–660.

Esposito, F., Seifritz, E., Formisano, E., Morrone, R., Scarabino, T., Tedeschi, G., Cirillo, S., Goebel, R. and Di Salle, F. (2003). Real-time independent component analysis of fMRI time-series, *Neuroimage* **20**(4): 2209–2224.

Fahlbusch, R., Ganslandt, O. and Nimsky, C. (2000). Intraoperative imaging with open magnetic resonance imaging and neuronavigation, *Child's Nervous System* **16**(10-11): 829–831.

Farquharson, S., Tournier, J.-D., Calamante, F., Fabinyi, G., Schneider-Kolsky, M., Jackson, G. D. and Connelly, A. (2013). White matter fiber tractography: why we need to move beyond DTI: clinical article, *Journal of neurosurgery* **118**(6): 1367–1377.

Fillard, P., Descoteaux, M., Goh, A., Gouttard, S., Jeurissen, B., Malcolm, J., Ramirez-Manzanares, A., Reisert, M., Sakaie, K., Tensaouti, F. et al. (2011). Quantitative evaluation of 10 tractography algorithms on a realistic diffusion MR phantom, *Neuroimage* **56**(1): 220–234.

Fischl, B. (2012). FreeSurfer, *Neuroimage* **62**(2): 774–781.

Friston, K. J., Ashburner, J., Heather, J. et al. (2003). Statistical parametric mapping, *Neuroscience databases: a practical guide* **1**(237): 5.

Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J.-P., Frith, C. D. and Frackowiak, R. S. (1994). Statistical parametric maps in functional imaging: a general linear approach, *Human brain mapping* **2**(4): 189–210.

Galloway, R. L., Maciunas Jr, R. J., Edwards, C. A. et al. (1992). Interactive image-guided neurosurgery, *Biomedical Engineering, IEEE Transactions on* **39**(12): 1226–1231.

Gasser, T., Ganslandt, O., Sandalcioglu, E., Stolke, D., Fahlbusch, R. and Nimsky, C. (2005). Intraoperative functional MRI: implementation and preliminary experience, *Neuroimage* **26**(3): 685–693.

Gering, D. T., Nabavi, A., Kikinis, R., Hata, N., O'Donnell, L. J., Grimson, W. E. L., Jolesz, F. A., Black, P. M. and Wells, W. M. (2001). An integrated visualization system for surgical planning and guidance using image fusion and an open MR, *Journal of Magnetic Resonance Imaging* **13**(6): 967–975.

Gerlach, R., de Rochemont, R. d. M., Gasser, T., Marquardt, G., Reusch, J., Imoehl, L. and Seifert, V. (2008). FEASIBILITY OF POLESTAR N20, AN ULTRA-LOW-FIELD INTRAOPERATIVE MAGNETIC RESONANCE IMAGING SYSTEM IN RESECTION CONTROL OF PITUITARY MACROADENOMAS: LESSONS LEARNED FROM THE FIRST 40 CASES, *Neurosurgery* **63**(2): 272–285.

Ghosh, A., Rho, Y., McIntosh, A. R., Kötter, R. and Jirsa, V. K. (2008). Noise during rest enables the exploration of the brain's dynamic repertoire, *PLoS Comput Biol* **4**(10): e1000196.

Goebel, R. and Linden, D. (2014). Neurofeedback with real-time functional MRI, *MRI in Psychiatry*, Springer, pp. 35–46.

Goldman, A. I. (2006). *Simulating minds: The philosophy, psychology, and neuroscience of mindreading*, Oxford University Press.

Goscinski, W. J. (2012). Multi-modal Australian sciences imaging and visualisation environment (MASSIVE), *Proceedings of the ATIP/A* CRC Workshop on Accelerator Technologies for High-Performance Computing: Does Asia Lead the Way?*, A* STAR Computational Resource Centre, p. 14.

Greicius, M. D., Krasnow, B., Reiss, A. L. and Menon, V. (2003). Functional connectivity in the resting brain: a network analysis of the default mode hypothesis, *Proceedings of the National Academy of Sciences* **100**(1): 253–258.

Greicius, M. D., Srivastava, G., Reiss, A. L. and Menon, V. (2004). Default-mode network activity distinguishes Alzheimer's disease from healthy aging: evidence from functional MRI, *Proceedings of the National Academy of Sciences of the United States of America* **101**(13): 4637–4642.

Grönemeyer, D., Seibel, R., Erbel, R., Schmidt, A., Melzer, A., Plaßmann, J., Deli, M., Schmermund, A., Baumgart, D., Welsch, R. et al. (1996). Equipment configuration and procedures: preferences for interventional micro therapy, *Journal of digital imaging* **9**(2): 81–96.

Gunkelman, J. D. and Johnstone, J. (2005). Neurofeedback and the brain, *Journal of Adult Development* **12**(2-3): 93–98.

Hadani, M., Spiegelman, R., Feldman, Z., Berkenstadt, H. and Ram, Z. (2001). Novel, compact, intraoperative magnetic resonance imaging-guided system for conventional neurosurgical operating rooms, *Neurosurgery* **48**(4): 799–809.

Hagmann, P., Reese, T., Tseng, W., Meuli, R., Thiran, J. and Wedeen, V. (2004). Diffusion spectrum imaging tractography in complex cerebral white matter: an investigation of the centrum semiovale, *International Society for Magnetic Resonance in Medicine, ISMRM TWELFTH SCIENTIFIC MEETING, Kyoto, Japan, 15-21 May 2004*, Vol. 12, International Society for Magnetic Resonance in Medicine, p. 623.

Hammond, D. C. (2005). Neurofeedback treatment of depression and anxiety, *Journal of Adult Development* **12**(2-3): 131–137.

Harris, M. (2008). Many-core GPU computing with NVIDIA CUDA, *Proceedings of the 22nd annual international conference on Supercomputing - ICS 08*, Association for Computing Machinery (ACM).
URL: *http://dx.doi.org/10.1145/1375527.1375528*

Hashemi, R. H., Bradley, W. G. and Lisanti, C. J. (2012). *MRI: the basics*, Lippincott Williams & Wilkins.

Haykin, S. and Chen, Z. (2005). The cocktail party problem, *Neural computation* **17**(9): 1875–1902.

Haykin, S. S., Haykin, S. S., Haykin, S. S. and Haykin, S. S. (2009). *Neural networks and learning machines*, Vol. 3, Pearson Education Upper Saddle River.

He, Y. and Evans, A. (2010). Graph theoretical modeling of brain connectivity, *Current opinion in neurology* **23**(4): 341–350.

Hollmann, M., Mönch, T., Mulla-Osman, S., Tempelmann, C., Stadler, J. and Bernarding, J. (2008). A new concept of a unified parameter management, experiment control, and data analysis in fMRI: application to real-time fMRI at 3T and 7T, *Journal of neuroscience methods* **175**(1): 154–162.

Hornak, J. P. (1997). Basics of NMR.

Hoyer, P. O. and Hyvärinen, A. (2000). Independent component analysis applied to feature extraction from colour and stereo images, *Network: Computation in Neural Systems* **11**(3): 191–210.

Hsieh, J. (2009). Computed tomography: principles, design, artifacts, and recent advances, SPIE Bellingham, WA.

Huber, P. J. (1985). Projection pursuit, *The annals of Statistics* pp. 435–475.

Hughes, A. J., Daniel, S. E., Kilford, L. and Lees, A. J. (1992). Accuracy of clinical diagnosis of idiopathic Parkinson's disease: a clinico-pathological study of 100 cases., *Journal of Neurology, Neurosurgery & Psychiatry* **55**(3): 181–184.

Humphrey, J. R., Price, D. K., Spagnoli, K. E., Paolini, A. L. and Kelmelis, E. J. (2010). CULA: hybrid GPU accelerated linear algebra routines, *SPIE defense, security, and sensing*, International Society for Optics and Photonics, pp. 770502–770502.

Hushek, S. G., Martin, A. J., Steckner, M., Bosak, E., Debbins, J. and Kucharzyk, W. (2008). MR systems for MRI-guided interventions, *Journal of Magnetic Resonance Imaging* **27**(2): 253–266.

Hyman, S. E. (2002). Neuroscience, genetics, and the future of psychiatric diagnosis, *Psychopathology* **35**(2-3): 139–144.

Hyvärinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis, *Neural Networks, IEEE Transactions on* **10**(3): 626–634.

Hyvärinen, A., Karhunen, J. and Oja, E. (2004). *Independent component analysis*, Vol. 46, John Wiley & Sons.

Hyvärinen, A. and Oja, E. (1997). A fast fixed-point algorithm for independent component analysis, *Neural computation* **9**(7): 1483–1492.

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications, *Neural networks* **13**(4): 411–430.

Hyviirinen, A., Karhunen, J. and Oja, E. (2001). Independent component analysis, *Wileyand Sons* .

Intel, R. (2011). Intel math kernel library reference manual, *Technical report*, Tech. Rep. 630813-051US, 2012.[Online]. Available: http://software. intel. com/sites/products/-documentation/hpc/mkl/mklman/mklman. pdf.

Jarusiewicz, B. (2002). Efficacy of neurofeedback for children in the autistic spectrum: A pilot study, *Journal of Neurotherapy* **6**(4): 39–49.

Jenkinson, M., Beckmann, C. F., Behrens, T. E., Woolrich, M. W. and Smith, S. M. (2012). Fsl, *Neuroimage* **62**(2): 782–790.

Jensen, M. P., Grierson, C., Tracy-Smith, V., Bacigalupi, S. C. and Othmer, S. (2007). Neurofeedback treatment for pain associated with complex regional pain syndrome type I, *Journal of Neurotherapy* **11**(1): 45–53.

Joanes, D. and Gill, C. (1998). Comparing measures of sample skewness and kurtosis, *Journal of the Royal Statistical Society: Series D (The Statistician)* **47**(1): 183–189.

Johnson, W. M. (1989). *Super-scalar processor design*, Stanford University.

Johnston, S. J., Boehm, S. G., Healy, D., Goebel, R. and Linden, D. E. (2010). Neurofeedback: A promising tool for the self-regulation of emotion networks, *Neuroimage* **49**(1): 1066–1072.

Jolliffe, I. (2002). *Principal component analysis*, Wiley Online Library.

Karvanen, J. and Theis, F. J. (2004). Spatial ICA of fMRI data in time windows, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering* **735**: 312–319.

Kazhdan, M., Funkhouser, T. and Rusinkiewicz, S. (2003). Rotation invariant spherical harmonic representation of 3 d shape descriptors, *Symposium on geometry processing*, Vol. 6, pp. 156–164.

Keith, D., Hoge, C., Frank, R., Malony, A. D. and Center, N. (2006). HiPerSAT Technical Report.

Kiviniemi, V., Starck, T., Remes, J., Long, X., Nikkinen, J., Haapea, M., Veijola, J., Moilanen, I., Isohanni, M., Zang, Y.-F. et al. (2009). Functional segmentation of the brain cortex using high model order group PICA, *Human brain mapping* **30**(12): 3865–3886.

Kiviniemi, V., Vire, T., Remes, J., Elseoud, A. A., Starck, T., Tervonen, O. and Nikkinen, J. (2011). A sliding time-window ICA reveals spatial variability of the default mode network in time, *Brain connectivity* **1**(4): 339–347.

Koay, C. G., Özarslan, E., Johnson, K. M. and Meyerand, M. E. (2012). Sparse and optimal acquisition design for diffusion MRI and beyond, *Medical physics* **39**(5): 2499–2511.

LaBar, K. S. and Cabeza, R. (2006). Cognitive neuroscience of emotional memory, *Nature Reviews Neuroscience* **7**(1): 54–64.

Lacroix, M., Abi-Said, D., Fourney, D. R., Gokaslan, Z. L., Shi, W., DeMonte, F., Lang, F. F., McCutcheon, I. E., Hassenbusch, S. J., Holland, E. et al. (2001). A multivariate analysis of 416 patients with glioblastoma multiforme: prognosis, extent of resection, and survival, *Journal of neurosurgery* **95**(2): 190–198.

Landaverde, R., Zhang, T., Coskun, A. K. and Herbordt, M. (2014). An investigation of Unified Memory Access performance in CUDA, *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, Institute of Electrical & Electronics Engineers (IEEE). **URL:** *http://dx.doi.org/10.1109/hpec.2014.7040988*

Lange, T., Eulenstein, S., Hünerbein, M., Lamecker, H. and Schlag, P.-M. (2004). Augmenting intraoperative 3D ultrasound with preoperative models for navigation in liver surgery, *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2004* pp. 534–541.

Langlois, D., Chartier, S. and Gosselin, D. (2010). An introduction to independent component analysis: InfoMax and FastICA algorithms, *Tutorials in Quantitative Methods for Psychology* **6**(1): 31–38.

Lawson, C. L. and Hanson, R. J. (1974). *Solving least squares problems*, Vol. 161, SIAM.

Lawson, C. L., Hanson, R. J., Kincaid, D. R. and Krogh, F. T. (1979). Basic linear algebra subprograms for Fortran usage, *ACM Transactions on Mathematical Software (TOMS)* **5**(3): 308–323.

Le Bihan, D. (2003). Looking into the functional architecture of the brain with diffusion MRI, *Nature Reviews Neuroscience* **4**(6): 469–480.

Lee, S. (2005). Blind source separation and independent component analysis: A review.

Lindholm, E., Nickolls, J., Oberman, S. and Montrym, J. (2008). NVIDIA Tesla: A unified graphics and computing architecture, *IEEE micro* (2): 39–55.

Liu, N. (2008). *Theory and Applications and Legendre Polynomials and Wavelets*, ProQuest.

Long, X.-Y., Zuo, X.-N., Kiviniemi, V., Yang, Y., Zou, Q.-H., Zhu, C.-Z., Jiang, T.-Z., Yang, H., Gong, Q.-Y., Wang, L. et al. (2008). Default mode network as revealed with multiple methods for resting-state functional MRI analysis, *Journal of neuroscience methods* **171**(2): 349–355.

Lu, Y., Jiang, T. and Zang, Y. (2003). Region growing method for the analysis of functional MRI data, *NeuroImage* **20**(1): 455–465.

Luebke, D. (2008). CUDA: Scalable parallel programming for high-performance scientific computing, *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, Institute of Electrical & Electronics Engineers (IEEE). **URL:** *http://dx.doi.org/10.1109/isbi.2008.4541126*

Lumenta, C. B., Gumprecht, H. and Krammer, M. J. (2010). Image-guided neurosurgery, *Neurosurgery*, Springer, pp. 579–590.

Lustmann, J. (2005). Intraoperative computerized navigation for flapless implant surgery and immediate loading in the edentulous mandible.

Lyon, G. R. and Rumsey, J. M. (1996). *Neuroimaging: A Window to the Neurological Foundations of Learning and Behavior in Children.*, ERIC.

Mäkelä, J. P., Kirveskari, E., Seppä, M., Hämäläinen, M., Forss, N., Avikainen, S., Salonen, O., Salenius, S., Kovala, T., Randell, T. et al. (2001). Three-dimensional integration of brain anatomy and function to facilitate intraoperative navigation around the sensorimotor strip, *Human brain mapping* **12**(3): 180–192.

Maldjian, J. A., Schulder, M., Liu, W.-C., Mun, I.-K., Hirschorn, D., Murthy, R., Carmel, P. and Kalnin, A. (1997). Intraoperative functional MRI using a real-time neurosurgical navigation system, *Journal of computer assisted tomography* **21**(6): 910–912.

Malinen, S., Hlushchuk, Y. and Hari, R. (2007). Towards natural stimulation in fMRIissues of data analysis, *Neuroimage* **35**(1): 131–139.

Markram, H. (2012). The Human Brain Project, *Sci Am* **306**(6): 50–55.
URL: *http://dx.doi.org/10.1038/scientificamerican0612-50*

Mataric, M. J. (2002). Action and Bioiogy to Robotics, *Imitation in animals and artifacts* p. 391.

McGeer, P. L., Schulzer, M. and McGeer, E. G. (1996). Arthritis and anti-inflammatory agents as possible protective factors for Alzheimer's disease A review of 17 epidemiologic studies, *Neurology* **47**(2): 425–432.

McGirt, M. J., Chaichana, K. L., Gathinji, M., Attenello, F. J., Than, K., Olivi, A., Weingart, J. D., Brem, H. and Quiñones-Hinojosa, A. r. (2009). Independent association of extent of resection with survival in patients with malignant brain astrocytoma: clinical article, *Journal of neurosurgery* **110**(1): 156–162.

McKeown, M. J., Sejnowski, T. J. et al. (1998). Independent component analysis of fMRI data: examining the assumptions, *Human brain mapping* **6**(5-6): 368–372.

Mischel, P. S., Nguyen, L. P. and Vinters, H. V. (1995). Cerebral cortical dysplasia associated with pediatric epilepsy. Review of neuropathologic features and proposal for a grading system., *Journal of Neuropathology & Experimental Neurology* **54**(2): 137–153.

Mislow, J. M., Golby, A. J. and Black, P. M. (2010). Origins of intraoperative MRI, *Magnetic resonance imaging clinics of North America* **18**(1): 1–10.

Monti, M. M. (2011). Statistical analysis of fMRI time-series: a critical review of the GLM approach, *Frontiers in human neuroscience* **5**(article 28).

Moritz, C. H., Haughton, V. M., Cordes, D., Quigley, M. and Meyerand, M. E. (2000). Whole-brain functional MR imaging activation from a finger-tapping task examined with independent component analysis, *American Journal of Neuroradiology* **21**(9): 1629–1635.

Mousazadeh, H., Marami, B., Sirouspour, S. and Patriciu, A. (2011). GPU implementation of a deformable 3D image registration algorithm, *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Institute of Electrical & Electronics Engineers (IEEE).
URL: *http://dx.doi.org/10.1109/iembs.2011.6091213*

Nabavi, A., Black, P. M., Gering, D. T., Westin, C.-F., Mehta, V., Pergolizzi Jr, R. S., Ferrant, M., Warfield, S. K., Hata, N., Schwartz, R. B. et al. (2001). Serial intraoperative magnetic resonance imaging of brain shift, *Neurosurgery* **48**(4): 787–798.

Nan, W., Rodrigues, J. P., Ma, J., Qu, X., Wan, F., Mak, P.-I., Mak, P. U., Vai, M. I. and Rosa, A. (2012). Individual alpha neurofeedback training effect on short term memory, *International journal of psychophysiology* **86**(1): 83–87.

Nimsky, C., Ganslandt, O., Cerny, S., Hastreiter, P., Greiner, G. and Fahlbusch, R. (2000). Quantification of, visualization of, and compensation for brain shift using intraoperative magnetic resonance imaging, *Neurosurgery* **47**(5): 1070–1080.

Nimsky, C., Ganslandt, O. and Fahlbusch, R. (2004). Functional neuronavigation and intraoperative MRI, *Advances and technical standards in neurosurgery*, Springer, pp. 229–263.

Nimsky, C., Ganslandt, O., Hastreiter, P. and Fahlbusch, R. (2001). Intraoperative compensation for brain shift, *Surgical neurology* **56**(6): 357–364.

Nvidia, C. (2008). CuSolver library, *NVIDIA Corporation, Santa Clara, California* **15**: 27.

NVIDIA, C. (2010). GPU Occupancy Calculator, *CUDA SDK* .

Nvidia, C. (2011). Nvidia cuda c programming guide, *Nvidia Corporation* **120**(18): 8.

Ogawa, S., Lee, T.-M., Kay, A. R. and Tank, D. W. (1990). Brain magnetic resonance imaging with contrast dependent on blood oxygenation, *Proceedings of the National Academy of Sciences* **87**(24): 9868–9872.

ONeil, R. et al. (1963). Convolution operators and $L(p, q)$ spaces, *Duke Mathematical Journal* **30**(1): 129–142.

Papoulis, A. and Pillai, S. U. (2002). *Probability, random variables, and stochastic processes*, Tata McGraw-Hill Education.

Park, S. K. and Miller, K. W. (1988). Random number generators: good ones are hard to find, *Communications of the ACM* **31**(10): 1192–1201.

Pieper, S., Halle, M. and Kikinis, R. (2004). 3D Slicer, *Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on*, IEEE, pp. 632–635.

Poline, J.-B. and Brett, M. (2012). The general linear model and fMRI: does love last forever?, *Neuroimage* **62**(2): 871–880.

Raimondo, F., Kamienkowski, J. E., Sigman, M. and Slezak, D. F. (2012). CUDAICA: GPU optimization of infomax-ICA EEG analysis, *Computational intelligence and neuroscience* **2012**: 2.

Raniga, P., Wong, K., Tournier, J.-D., Connelly, A. and Salvado, O. (2012). Gpu accelerated csd-based probabilistic tractography, *ISMRM Annual Meeting* p. 1911.

Roberts, D. W., Hartov, A., Kennedy, F. E., Miga, M. I. and Paulsen, K. D. (1998). Intraoperative brain shift and deformation: a quantitative analysis of cortical displacement in 28 cases, *Neurosurgery* **43**(4): 749–758.

Rolls, E. T. and Deco, G. (2002). *Computational neuroscience of vision*, Oxford university press Oxford.

Rota, G., Sitaram, R., Veit, R., Erb, M., Weiskopf, N., Dogil, G. and Birbaumer, N. (2009). Self-regulation of regional cortical activity using real-time fMRI: The right inferior frontal gyrus and linguistic processing, *Human brain mapping* **30**(5): 1605–1614.

Roux, F., Boulanouar, K., Ranjeva, J., Manelfe, C., Tremoulet, M., Sabatier, J. and Berry, I. (1999). Cortical intraoperative stimulation in brain tumors as a tool to evaluate spatial data from motor functional MRI, *Investigative radiology* **34**(3): 225–229.

Saad, Y. (1992). *Numerical methods for large eigenvalue problems*, Vol. 158, SIAM.

Sanai, N., Polley, M.-Y., McDermott, M. W., Parsa, A. T. and Berger, M. S. (2011). An extent of resection threshold for newly diagnosed glioblastomas: clinical article, *Journal of neurosurgery* **115**(1): 3–8.

Sato, J. R., Hoexter, M. Q., Fujita, A. and Rohde, L. A. (2012). Evaluation of pattern recognition and feature extraction methods in ADHD prediction, *Frontiers in systems neuroscience* **6**(68): 23015782.

Savadjiev, P., Campbell, J. S., Descoteaux, M., Deriche, R., Pike, G. B. and Siddiqi, K. (2008). Labeling of ambiguous subvoxel fibre bundle configurations in high angular resolution diffusion MRI, *NeuroImage* **41**(1): 58–68.

Schnupp, J., Nelken, I. and King, A. (2011). *Auditory neuroscience: Making sense of sound*, MIT Press.

Semrud-Clikeman, M. (2016). Research in Brain Function and Learning.
URL: *http://www.apa.org/education/k12/brain-function.aspx*

Sherlin, L. H., Arns, M., Lubar, J., Heinrich, H., Kerson, C., Strehl, U. and Sterman, M. B. (2011). Neurofeedback and basic learning theory: implications for research and practice, *Journal of Neurotherapy* **15**(4): 292–304.

Smith, S. M., Beckmann, C. F., Andersson, J., Auerbach, E. J., Bijsterbosch, J., Douaud, G., Duff, E., Feinberg, D. A., Griffanti, L., Harms, M. P. et al. (2013). Resting-state fMRI in the human connectome project, *Neuroimage* **80**: 144–168.

Soldati, N., Calhoun, V. D., Bruzzone, L. and Jovicich, J. (2013). ICA analysis of fMRI with real-time constraints: an evaluation of fast detection performance as function of algorithms, parameters and a priori conditions, *Front. Hum. Neurosci* **7**(19.10): 3389.

Stone, J. E., Gohara, D. and Shi, G. (2010). OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems, *Comput. Sci. Eng.* **12**(3): 66–73.
URL: *http://dx.doi.org/10.1109/mcse.2010.69*

Sullivan, E. V. and Pfefferbaum, A. (2006). Diffusion tensor imaging and aging, *Neuroscience & Biobehavioral Reviews* **30**(6): 749–761.

Sulzer, J., Haller, S., Scharnowski, F., Weiskopf, N., Birbaumer, N., Blefari, M. L., Bruehl, A., Cohen, L., Gassert, R., Goebel, R. et al. (2013). Real-time fMRI neurofeedback: progress and challenges, *Neuroimage* **76**: 386–399.

Tambini, A., Ketz, N. and Davachi, L. (2010). Enhanced brain correlations during rest are related to memory for recent experiences, *Neuron* **65**(2): 280–290.

Tanner, L. (1979). The spreading of silicone oil drops on horizontal surfaces, *Journal of Physics D: Applied Physics* **12**(9): 1473.

Tournier, J., Calamante, F., Connelly, A. et al. (2012). MRtrix: diffusion tractography in crossing fiber regions, *International Journal of Imaging Systems and Technology* **22**(1): 53–66.

Tournier, J.-D., Calamante, F. and Connelly, A. (2007). Robust determination of the fibre orientation distribution in diffusion MRI: non-negativity constrained super-resolved spherical deconvolution, *NeuroImage* **35**(4): 1459–1472.

Tournier, J.-D., Calamante, F., Gadian, D. G. and Connelly, A. (2004). Direct estimation of the fiber orientation density function from diffusion-weighted MRI data using spherical deconvolution, *NeuroImage* **23**(3): 1176–1185.

Tournier, J.-D., Mori, S. and Leemans, A. (2011). Diffusion tensor imaging and beyond, *Magnetic Resonance in Medicine* **65**(6): 1532–1556.

Treisman, A. M. and Gelade, G. (1980). A feature-integration theory of attention, *Cognitive psychology* **12**(1): 97–136.

Tuch, D. S. (2004). Q-ball imaging, *Magnetic resonance in medicine* **52**(6): 1358–1372.

Unsgaard, G., Ommedal, S., Muller, T., Gronningsaeter, A. and Hernes, T. A. N. (2002). Neuronavigation by intraoperative three-dimensional ultrasound: initial experience during brain tumor resection, *Neurosurgery* **50**(4): 804–812.

van Eimeren, T., Monchi, O., Ballanger, B. and Strafella, A. P. (2009). Dysfunction of the default mode network in Parkinson disease: a functional magnetic resonance imaging study, *Archives of neurology* **66**(7): 877–883.

Van Essen, D. C., Smith, S. M., Barch, D. M., Behrens, T. E., Yacoub, E., Ugurbil, K., Consortium, W.-M. H. et al. (2013). The WU-Minn human connectome project: an overview, *Neuroimage* **80**: 62–79.

Van Essen, D. C., Ugurbil, K., Auerbach, E., Barch, D., Behrens, T., Bucholz, R., Chang, A., Chen, L., Corbetta, M., Curtiss, S. W. et al. (2012). The Human Connectome Project: a data acquisition perspective, *Neuroimage* **62**(4): 2222–2231.

Wang, Z., Zhang, H., Wu, X., Yao, L. and Long, Z. (2013). Real-time independent component analysis of fMRI using spatial constraint, *SPIE Medical Imaging*, International Society for Optics and Photonics, pp. 86720P–86720P.

Wayth, R. B., Greenhill, L. J. and Briggs, F. H. (2009). A GPU-based Real-time Software Correlation System for the Murchison Widefield Array Prototype, *PUBL ASTRON SOC PAC* **121**(882): 857–865.
URL: *http://dx.doi.org/10.1086/605334*

Wee, C.-Y., Yap, P.-T., Li, W., Denny, K., Browndyke, J. N., Potter, G. G., Welsh-Bohmer, K. A., Wang, L. and Shen, D. (2011). Enriched white matter connectivity networks for accurate identification of MCI patients, *Neuroimage* **54**(3): 1812–1822.

Weishaupt, D., Köchli, V. D. and Marincek, B. (2008). *How does MRI work?: an introduction to the physics and function of magnetic resonance imaging*, Springer Science & Business Media.

Weiskopf, N. (2012). Real-time fMRI and its application to neurofeedback, *Neuroimage* **62**(2): 682–692.

Weiskopf, N., Klose, U., Birbaumer, N. and Mathiak, K. (2005). Single-shot compensation of image distortions and BOLD contrast optimization using multi-echo EPI for real-time fMRI, *Neuroimage* **24**(4): 1068–1079.

Weiskopf, N., Sitaram, R., Josephs, O., Veit, R., Scharnowski, F., Goebel, R., Birbaumer, N., Deichmann, R. and Mathiak, K. (2007). Real-time functional magnetic resonance imaging: methods and applications, *Magnetic resonance imaging* **25**(6): 989–1003.

Whaley, R. C. and Dongarra, J. J. (1998). Automatically tuned linear algebra software, *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, pp. 1–27.

Wilkinson, I., Romanowski, C., Jellinek, D., Morris, J. and Griffiths, P. (2014). Motor functional MRI for pre-operative and intraoperative neurosurgical guidance, *The British journal of radiology* .

Wilt, N. (2013). *The cuda handbook: A comprehensive guide to gpu programming*, Pearson Education.

Wittenbrink, C. M., Kilgariff, E. and Prabhu, A. (2011). Fermi GF100 GPU architecture, *IEEE Micro* (2): 50–59.

Wolf, I., Vetter, M., Wegner, I., Böttger, T., Nolden, M., Schöbinger, M., Hastenteufel, M., Kunert, T. and Meinzer, H.-P. (2005). The medical imaging interaction toolkit, *Medical image analysis* **9**(6): 594–604.

Yoo, S.-S., Guttmann, C. R. and Panych, L. P. (2001). Multiresolution data acquisition and detection in functional MRI, *Neuroimage* **14**(6): 1476–1485.

Zang, Y., Jiang, T., Lu, Y., He, Y. and Tian, L. (2004). Regional homogeneity approach to fmri data analysis, *Neuroimage* **22**(1): 394–400.

Zhang, J., Ji, H., Kang, N. and Cao, N. (2005). Fiber tractography in diffusion tensor magnetic resonance imaging: A survey and beyond, *Proceedings of the 2005 International Symposium on Medical Imaging and Computing*, pp. 39–56.

Zibulevsky, M. and Pearlmutter, B. A. (2001). Blind source separation by sparse decomposition in a signal dictionary, *Neural computation* **13**(4): 863–882.

Zou, Q.-H., Zhu, C.-Z., Yang, Y., Zuo, X.-N., Long, X.-Y., Cao, Q.-J., Wang, Y.-F. and Zang, Y.-F. (2008). An improved approach to detection of amplitude of low-frequency fluctuation (alff) for resting-state fmri: fractional alff, *Journal of neuroscience methods* **172**(1): 137–141.

Zwart, S. F. P., Belleman, R. G. and Geldof, P. M. (2007). High-performance direct gravitational N-body simulations on graphics processing units, *New Astronomy* **12**(8): 641–650.
URL: *http://dx.doi.org/10.1016/j.newast.2007.05.004*