UNIVERSITY OF CALIFORNIA, SAN DIEGO

# GPU Acceleration of Optical Mapping Algorithm for Cardiac Electro-Physiology

A Thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Electrical Engineering

by

Pingfan Meng

Committee in charge:

Professor Ryan Kastner, Chair
Professor Pamela Cosman, Co-Chair
Professor Shaya Fainman
Professor Kenneth Zeger

2011

The thesis of Pingfan Meng is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2011

DEDICATION

*To my parents (Jianping and Fangang).*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Professor Kastner for his advice that guides me in my academic research. I would also like to thank Dr. Ali Irturk, Sam Wood and Alex Indaco for the discussions on the optical mapping algorithm. Finally, I would like to thank Helen Wang who helped my written English.

ABSTRACT OF THE THESIS

**GPU Acceleration of Optical Mapping Algorithm for Cardiac Electro-Physiology**

by

Pingfan Meng
Master of Science in Electrical Engineering

University of California San Diego, 2011

Professor Ryan Kastner, Chair
Professor Pamela Cosman, Co-Chair

Hundreds of thousands of people die from heart attack in the U.S. every year. Cardiac dysrhythmia is known to be one of the most important causes for heart attack. Optical mapping is a tool for researching cardiac dysrhythmia and preventing heart attack. However, the optical mapping algorithm is computationally intensive and consumes a considerable amount of time. For example, 1 second of data requires 3.7 hours of computation. Therefore, a GPU implementation of the optical mapping algorithm is used in cardiac electrophysiology. The efficiency of different input sizes and memory access are studied to improve the GPU peformance. The result shows that GPU implementation of the optical mapping algorithm has a 33X speedup over the equivalent CPU implementation.

# Chapter 1

# Introduction

## 1.1 Background

An abnormal heart beat rhythm is often referred to as heart arrhythmia. A normal heart rate is 50 to 100 beats per minute. Cardiac arrhythmia is a term to describe a large and heterogeneous group of conditions in which there is abnormal electrical activity in the heart, some of which are life threatening. Arrhythmia is always accompanied by irregularities in the electrical impulse in the heart.

One serious risk for life-threatening ventricular arrhythmia and sudden cardiac death is prolongation of the cardiac electrical impulses (action potentials) and the resultant prolongation of the QT interval on the electrocardiogram. There are inherited versions of long QT syndrome (LQTS) associated with defects in no fewer than seven genes. Humans with these gene mutations are at risk of arrhythmia and sudden death that often strikes without warning in young adults.

An invasive method in cardiac physiology for studying arrhythmias, optical mapping, measures the heart's electrical impulses using a voltage sensitive fluorescent dye injected into the heart. Electrical phenomena can be visualized by measuring the various intensities of the light using one or more high-speed CMOS cameras to

provide a high temporal resolution signal. Optical mapping also encompasses the technique of applying filters and statistical resolution signals of the entire heart's electro-physiology.

Traditional methods of recording the transmembrane voltage and spread of activation in intact myocardium are to use contact electrodes such as microelectrodes, monophasic action potential (MAP) electrodes, or extracellular electrodes. Microelectrodes are useful for directly recording action potentials from a single location. MAP recordings provide accurate measures of activation time and action potential morphology[1], however they are typically used to record MAPs at a single location. Attempts have been made to build MAP electrode arrays, but at a very low spatial resolution[2]. Extracellular electrode arrays measure electrograms from a number of points on the surface of the heart with fairly high spatial resolution and high temporal resolution. These are useful to look at the spread of electrical activation, but are unable to record transmembrane voltage so one cannot get a true measure of the action potential shape[3]. On the other hand, optical mapping employs a potentiometric fluorescent dye whose emission intensity varies linearly with transmembrane voltage. Optical action potentials are recorded with a high speed imaging sensor[4]. These dyes have a very fast response time, on the order of microseconds, therefore the temporal resolution is only limited by the recording technology instead of the response time of the dye[12]. Recent advances in imaging technology allow researchers to optically monitor transmembrane voltage transients with high spatial and temporal resolution. Additionally, applying mechanical forces to the myocardium with electrodes can alter the electrical activity through a phenomenon called mechano-electrode feedback (MEF). Optical mapping is a non-contact method, which is very important in many studies, especially those investigating MEF[13].

Cardiac optical mapping has been used in a variety of preparations ranging from the single cell to the intact organ, while optical mapping in general has potential applications to other types of electrically excitable cells including brain and nerve

cells.

Despite these advantages, optical mapping is not currently employed in high throughput drug screening experiments that require a high resolution analysis of cardiac electrophysiology (long QT syndrome in particular). Conventional implementations of optical mapping cannot perform the computationally intensive noise reduction and analysis in an acceptable time frame (days of processing for seconds worth of data). This fact severely limits the types of experiments that can use optical mapping. Therefore, there is a demand for real-time analysis with the resolution that optical mapping provides.

## 1.2 Motivation and Research Goal

The motivation of our graphic process unit (GPU) acceleration research on optical mapping technology is from its variety of potential applications in biomedical experimental studies. In the following, some of these scenarios are described. Then, we provide a discussion of high throughput drug screening, as this is the specific application that we targeted in this research.

Real-time, closed loop feedback: Real-time, closed loop feedback is a promising method for cardiac electrophysiology. This includes dynamic clamp[14][15][16], and the usage of tissue-level electrophysiological activity to prevent the onset of arrhythmia[17][18]. Optical mapping has been used to provide feedback control to understand cardiac electrophysiology [1, 2]. These closed loop control systems offer the unique ability to understand the heart dynamics by observing real-time stimulus/response mechanisms over a large area.

Arrhythmia Production: Optical mapping is often used to study arrhythmia production in the heart. In many knockout models (i.e., animals in which a specific gene is inserted or deleted) for cardiac disorders, researchers are interested in identifying whether certain pathological conditions make the heart more susceptible to

arrhythmia. These arrhythmias can occur at certain foci in the heart. With real time visualization one may be able to identify these foci immediately and study them more closely in the same heart. This would require multiple animals without the real-time capability, and with inter heart variability this could present additional challenges. Also, though one can identify the presence of arrhythmias using the electrocardiogram, a deeper understanding of the arrhythmia can be studied using optical mapping by identifying the action potential duration (APD), action potential (AP) amplitude, recurrent loops in the AP, conduction velocity, etc.

Enhanced Field of View: The exact area of the heart recorded by any given camera is determined by the desired resolution and the field of view (FOV). That is, we could modify the optical lenses to increase the field of view. This, however, would reduce the spatial resolution, i.e., each individual pixel would correspond to a larger area of the heart. While the precise tradeoff between FOV and resolution largely depends on the application, the use of multiple cameras can be used to increase both the FOV and resolution. This increases the amount of incoming data, which in turn increases the required computation time.

Immediate Experimental Feedback: The ability to see the optical mapping results during the experimental procedure should significantly reduce both the duration of the experiment and the required number of animals. For example, if one was interested in looking at the effect of an injectable therapy, it would be possible to see if there was a difference immediately and tailor the injection site, volume, number of injections, pacing method, pacing site, etc. This would be helpful for both whole heart and cell monolayer experiments as it would be feasible to adjust the pacing location to study electrical stimulation in the healthy tissue, versus the border zone and infarct locations. Drug Screening: Drug-induced polymorphic ventricular tachycardia (PMVT) or torsades de pointes is a significant and highly-visible clinical problem that has led to routine QT interval screening for all newly developed pharmacologic agents. However, it is increasingly appreciated that QT interval pro-

longation is neither specific nor reliable for the occurrence of PMVT. In addition to the problems of consistent measurement and interpretation of QT interval, some drugs cause substantial QT prolongation without increasing the risk of malignant reentrant ventricular arrhythmias, and vice versa.

High throughput screening would enable large screens of drugs that induce long-QT syndrome and torsades de pointes. High throughput screening would allow us to identify, understand and model the conditions under which these arrhythmias occur. High-speed optical mapping is an important tool in understanding this behavior. It would provide information not available in the ECG alone that could help discriminate and understand the underlying mechanisms responsible for drug induced arrhythmia.

All of the applications described above are computationally intensive. Usually, these experiments take considerable amounts of time which make the biomedical research unproductive. For example, 1 second of data requires 3.7 hours of computation on a quad-core CPU computer. The conventional multi-core CPU architecture cannot process such a high throughput algorithm with a high-performance. Therefore, our research goal is to deploy many-core architecture processors such as GPUs to accelerate the optical mapping algorithm. With the GPU acceleration, the cardiac researchers can get the analysis results in experiments more quickly and easily. We focused on developing and implementing programming models for optical mapping algorithm on GPUs. We also focused on optimizing the GPU program to gain a higher performance.

## 1.3 Major Contributions

The major contributions of this thesis are:

- Detailed analysis of the optical mapping algorithm and determination of the bottlenecks;

- Detailed analysis of GPU memory access optimization in the optical mapping CUDA program;
- Detailed analysis of GPU implementation of optical mapping algorithm in terms of different input sizes;
- High performance implementation of optical mapping algorithm using GPU GT230M that provides 33X speedup over an equivalent CPU implementation.

# Chapter 2

# Related Research

GPU acceleration has been applied to many biomedical imaging researches. CT reconstruction algorithms have been accelerated on GPUs[5][6]. These studies on CT reconstruction algorithms were focused on accelerating the Simultaneous Algebraic Reconstruction Technique (SART). MRI reconstructions have been studied in recent years. The research in this area has mainly focused on accelerating the fast Fourier transform (FFT)[7][8].

The GPU acceleration in cardiac electro-physiology has rarely been studied. Lionetti presented a study of GPU accelerated cardiac electro-physiology[9] which focuses on waveform differential equations solving for cardiac electro-physiology simulation. The research presented in this thesis focuses on optical mapping technology which studies the actual electro-physiology of a real animal heart by processing the captured video. Therefore, this thesis focuses on biomedical image processing instead of numerical differential equation analysis presented in Lionetti's thesis.

There are many other research articles that use GPUs for biomedical image processing. The unique feature of the optical mapping is the substantial temporal nature of the algorithm. For example, the phase shift function for a 1024 frames video requires operating over 32,768 interpolated frames. Most other related works [10][11]

operated over a much smaller temporal scale (e.g., only one or a few frames).

# Chapter 3

# Optical Mapping Algorithm

## 3.1 Algorithm Overview

The optical mapping algorithm consists of two major stages: (1) signal conditioning; (2) analysis. The raw image intensity signals are first conditioned using a variety of techniques in an attempt to reduce the noise. This includes spatial phase-shifted filtering and temporal median filtering[12]. Additionally, some experiments require the removal of motion artifacts due to cardiac contraction from the signal using a combination of ratiometry and motion tracking. This is followed by an analysis stage that performs actions like determining the activation time and subsequently creating an activation map over the ventricular epicardium showing the rate and path of action potential propagation. Other analyses include determining the spatial gradients of activation time to calculate local apparent conduction velocity vectors at each pixel[19]. The action potential duration (APD) is then defined as the time difference between repolarization (when the action potential has recovered back to the baseline) and activation. The APD dispersion is determined as the standard deviation of APD over the surface of the heart. Alternatively, to track organizing centers of meandering spiral waves, the phase is calculated over the surface of the

heart and phase singularities are located and followed.

The signal conditioning stage is the major computationally intensive part of the optical mapping algorithm. Therefore, we focused our acceleration strategy on studying the signal conditioning stage. In the following, we describe the signal conditioning stage in detail.

Signal conditioning begins with normalization and signal inversion. This translates the change in fluorescence into corresponding voltage by calculating the change in florescence from the baseline background florescence. Note that the fluorescence and voltage are inversely proportional. Further signal processing determines whether a particular pixel is interesting using a minimum range and a minimum value. This reduces the computational complexity by eliminating background (i.e., non-heart) pixels. More specifically, a pixel is marked valid if it meets two conditions: (1) the range of its values is greater than a given range (the pixel is in action), (2) the maximum of its values is greater than the given value (the pixel is active enough to be a heart pixel). Normalization only occurs on these "interesting" pixels.

Figure 3.1 shows the steps of optical mapping algorithm which includes a phase shift spatial finite impulse response (FIR) filter and a temporal median filter to enhance the electrical signal. More specifically, a phase shift spatial filter is applied on a kernel window that first passes through low pass interpolation by a factor of 10. This function iterates across the raw data in the kernel window, applying both spatial and temporal filters. The phase shifting function takes a window across time, a mask that defines valid pixels and a dimension that specifies the center of the window and phase shifts the entire window in time by using the center pixel as a reference. The shifted window is returned along with the mean shift amount. It should be noted that the pixel's entire duration through time is used when comparing the phase for each pixel against the reference pixel. Also another function calculates a phase shift difference between a reference pixel, a pixel to be shifted, and a power that is used to determine the level of accuracy. A discrete fast Fourier transform (FFT) is used

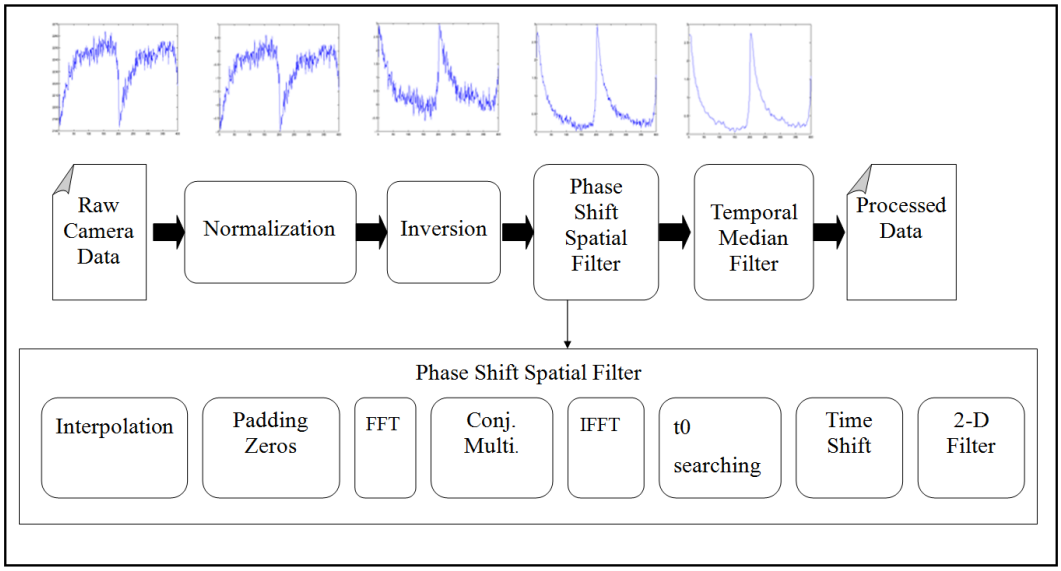


Figure 3.1: Signal conditioning stage flow chart and the correspondent image signals.

to calculate the phase shift amount. Since the length of the input array of FFT must be an integer power of 2, a padding zeros function is used on the input array before FFT. After performing the spatial phase shift filtering, the signal undergoes temporal median filtering. Median filtering replaces a pixel with the median value of the set of itself and a certain number of its adjacent pixels along the time axis. Median filtering is used because it preserves the steep upstroke of the optical action potential.

The phase-shift spatial filtering and the temporal filter are the two most complex modules in the signal conditioning stage shown in Figure 3.1. Also, described in section 3.4, these two modules take more than 95% of the total run-time of the optical mapping algorithm. Therefore, we describe these two modules in section 3.2 and 3.3 showing the algorithms they use and their necessity for the image quality enhancement.

## 3.2 Phase-Shift Spatial Filtering

As shown in Figure 3.1, the phase-shift spatial filtering module can be broken into sub-modules. They are interpolation, padding zeros, FFT, conjugated multiplication, IFFT, $t_0$ searching, time shifting and 2-D filter. In subsection 3.2.1-3.2.3 sub-modules, we describe these sub-modules.

### 3.2.1 Interpolation and Padding Zeros

The camera speed is 1,000 frames per second, referring to the appendix. Although it is high speed video capturing, it is still far from representing the real continuous-time potential fields. Therefore, in order to reconstruct a continuous description of the potential fields for optical mapping analysis, an interpolation is used to fill the regions between frames to enhance the sampling density[20].

The interpolation algorithm is implemented by using a low-pass finite impulse response (FIR) filter. Firstly, the algorithm expands the input array to the length increased by the interpolation factor. Secondly, it generates a coefficient array of a symmetric filter. The expanded array passes through this filter to produce the interpolated output array[21]. In the optical mapping algorithm, in order to enhance the sampling rate along the time axis (frames), a video data with $10\times$ more frames is produced by the interpolation algorithm, as shown in Figure 3.2.

The number of frames of the interpolated video data is usually not a number of power of two which is necessary for the FFT operation in the next stage. Therefore, the interpolated array needs to be padded with zeros to make the number of frames a power of two. For example, an original video with 1,024 frames will have 10,240 frames after interpolation. This interpolated data will be expanded to $32,768 = 2 \times 10^{15}$ frames by padding zeros.

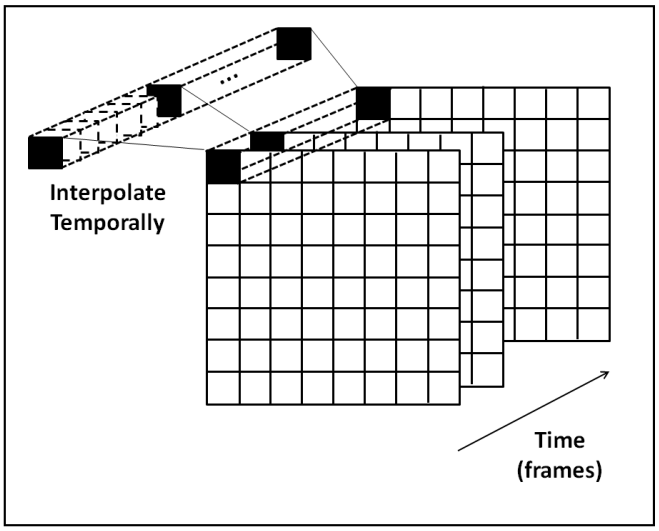


Figure 3.2: Demonstration of temporal interpolation.

### 3.2.2 FFT, Conjugated Multiplication, IFFT, $t_0$ searching and Time (Phase) Shifting

A 2-D spatial filter (described in subsection 3.2.3) is applied on individual frames using the information about the neighboring pixels to enhance the signal. However, due to the fact that the signal is moving as a waveform, the neighboring pixels all have phase differences with the pixel which is to be filtered, as shown in Figure 3.3. These phase differences will be noise on the filtered image if we apply the spatial filter directly. Therefore, it is necessary to eliminate these phase differences by shifting the pixels along the time axis, as shown in Figure 3.4.

The method to find the phase difference between two pixels is described as following:

For two signals $s_1$ and $s_2$, assume signal $s_2$ is the time-shifted version with delay $t_0$ of signal $s_1$:

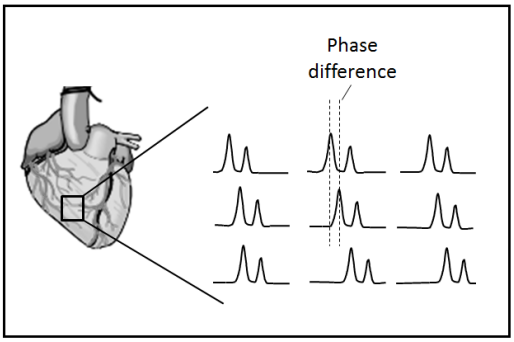


Figure 3.3: Waveform on heart surface and their phase differences demonstration.

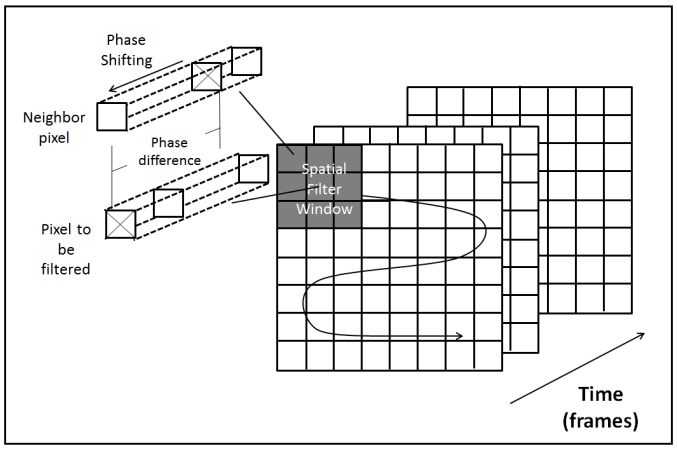


Figure 3.4: Demonstration of phase shifting.

$$s_2(t) = s_1(t - t_0) \tag{3.1}$$

From equation 3.1, the relation between the Fourier transforms of s1 and s2 is:

$$S_2(f) = e^{-j2\pi f t_0} S_1(f) \tag{3.2}$$

Therefore, t0 can be found:

$$\frac{S_2(f)S_1^*(f)}{|S_2(f)S_1^*(f)|} = e^{-j2\pi f t_0} \tag{3.3}$$

Taking the inverse Fourier transforms on both sides of equation 3.3, there will be:

$$F^{-1}(\frac{S_2(f)S_1^*(f)}{|S_2(f)S_1^*(f)|}) = \delta(t - t_0) \tag{3.4}$$

Thus, from equation 3.1 to 3.4, the time shift operation is: (1) take FFT's of the reference array and an array to be shifted; (2) take the conjugation of the reference FFT array and multiply it with the other FFT array; (3) take the IFFT of the output array from step 2; (4) in the output array of step 4, find the maximum value (due to the $\delta$ function). The index of the maximum value is the time difference $t_0$ that we require.

After the $t_0$'s are found, in each filter window shown in Figure 3.4, the neighbor pixels are shifted with the correspondent time differences. Due to the fact that each pixel has its own filter window, the output data of phase shifting is expanded with a factor which is the window size. This output data is filtered by a 2-D FIR filter which is described in subsection 3.2.3.

### 3.2.3 2-D (Spatial) Filter

A 2-D spatial filter is used to reduce the noise on the image data. In our case, the 2-D filter is a $5 \times 5$ Gaussian filter ($\delta = 1.179$) [12]. Therefore, each pixel is replaced

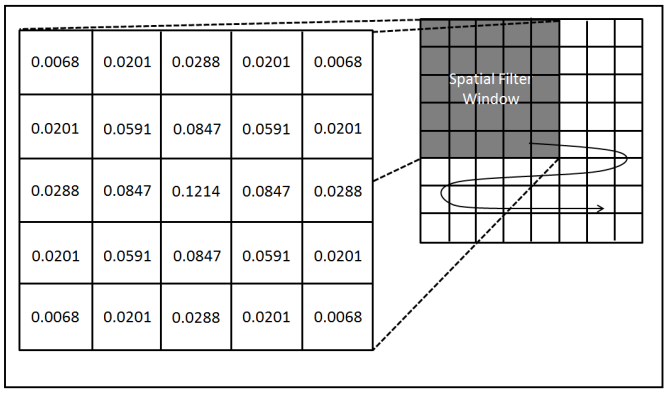


Figure 3.5: Demonstration of the 2-D spatial filter.

with the convolution of the $5 \times 5$ Gaussian filter and 24 phase shifted neighboring pixels. The 2-D filter operation and its coefficients are shown in Figure 3.5

## 3.3 Temporal Filtering

A temporal filter is applied on the output of the phase-shift filter to further enhance the image quality. We used a median filter of length 5 because it best preserves the feature of the cardiac actions. The median filter is demonstrated in Figure 3.6.

## 3.4 Profiling and Bottlenecks Study

In order to guide our GPU acceleration implementation, we profiled and studied the bottlenecks of the original Matlab serial implementation of the optical mapping algorithm.

Figure 3.7 shows the total run-time of the Matlab optical mapping program on a 2.66Ghz Quad-Core Intel Xeon machine with Matlab 2009a (256KB L2 cache per

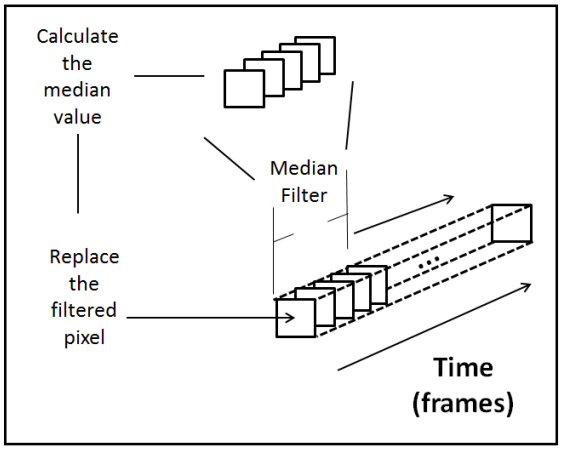


Figure 3.6: Demonstration of the temporal median filter.

core, 8 MB shared L3 cache and 3GB of main memory). As can be seen from the Figure, 256 frames require roughly an hour of processing time (51 min). Increasing the amount of input data and the number of runs significantly increases the computation time. 1024 frames with 6 runs roughly requires 1 day of processing time (22 hours). The different numbers of runs are tested because the optical mapping experiment is usually done repeatedly in a medicine testing process due to the possibility of failure. This particular experiment used a sampling rate of 1000 frames/second, which corresponds to approximately 6 seconds of data (6 runs each of which are a little over 1 second). In other words, 1 second of data requires 3.7 hours of computation.

Additionally, in Figure 3.7 (b), over 95% of the total run-time is taken by the phase shift spatial filer and the temporal median filter. Therefore, our design focuses on parallelizing these two parts of the optical mapping algorithm.

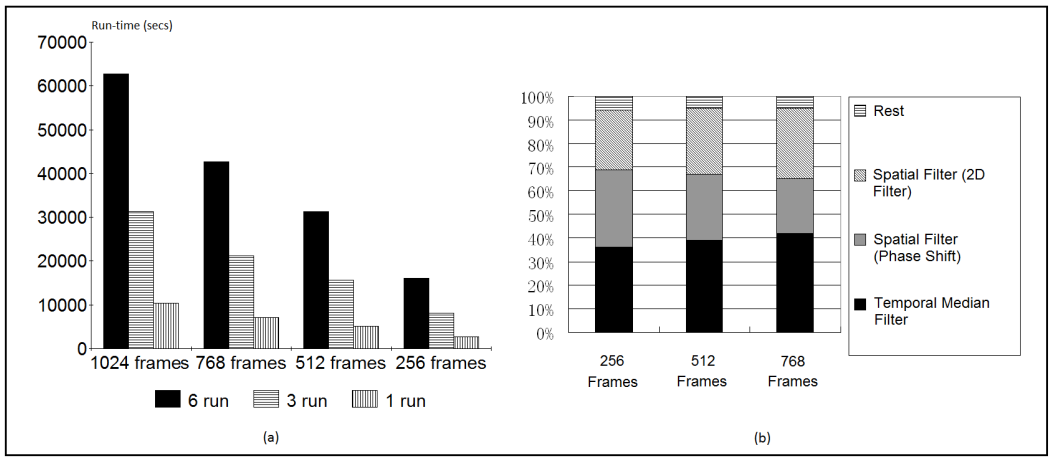


Figure 3.7: Bottleneck analysis. (a) The run-times for different size of video and numbers of "runs". (b) The time consumption of each step of optical mapping algorithm, which indicates the spatial filtering part and temporal filtering part take more than 95% of the total run-time.

# Chapter 4

# GPU Platform

## 4.1 GPUs

### 4.1.1 Overview

GPUs are known as a type of many-core processors that focus more on the execution throughput of parallel applications. This attribute makes GPUs have a very high performance in throughput computations. In recent years, the ratio between GPUs and multicore CPUs for throughput peak floating-point calculation is about $10 : 1$. Also, GPUs provide a low cost platform for hardware-assisted throughput computation. They tend to perform well on highly parallel applications. Due to the pixels can be processed independently in most of the stages of the optical mapping algorithm, the algorithm is possible to be highly parallelized. Therefore, the GPUs are suited for accelerating the optical mapping algorithm.

### 4.1.2 Architecture of GPUs

Figure 4.1 shows a simplified view of the GPU architecture. The GPU hardware provides a set of multiprocessors and it dynamically assigns each thread block to

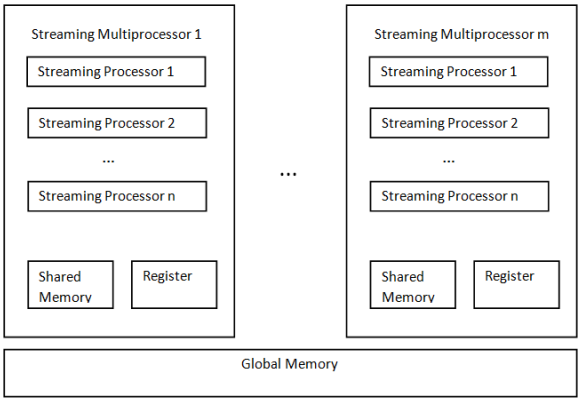


Figure 4.1: Simplified GPU architecture.

a stream multiprocessor (SM). In each block, the threads are scheduled as a set of warps, and each warp has 32 threads that execute in Single Instruction, Multiple Threads (SIMT) model. There can be certain amount of warps residing in each SM at any point in time. One of these warps will be executed by the hardware at a point in time. But SM can efficiently arrange the warps during execution. When a warp needs to wait for the result of a previously initiated long-latency operation, it is placed into the waiting list until the data is ready. One of the other warps which are no longer waiting for results is selected for execution.

Due to the SIMT model, the principle of GPU parallel computing is the idea that there is a significant amount of data which must be operated by a unified set of instructions. Also, due to the automatical thread scheduling mechanism, another important idea in GPU parallel computing is to launch enough threads at a time so that the stream processors can mask the memory access latency time by processing other threads when some threads are waiting for memory response.

The storage arrangement on GPUs is described as following. The threads in a thread block share local storages which are known as shared memory and register.

Another type of storage is called global memory, which any thread on the GPU may access. This global memory has higher access latency (about 20 times higher than that of shared memory or registers). However, the sizes of the local storages are very limited e.g. about 48KB per multiprocessor. The host (CPU) can write data from main memory (RAM) to the GPU global memory, or read data from the GPU global memory to the main memory.

## 4.2 CUDA Programming

### 4.2.1 Overview

Compute unified device architecture (CUDA) is an extension of the C programming language developed by NVIDIA for programming their GPUs. CUDA achieves high scalability which means simple, low-overhead thread management and no cache coherence hardware requirements. A CUDA program executes sequences of kernels, as demonstrated in Figure 4.2. Each of these kernels revokes a set of threads which are organized into two-dimensional thread blocks. Each thread block has a certain number of threads. This programming model is shown in Figure 4.3. The dimension of grids can be 1-D or 2-D, while the dimension of blocks can be 1-D, 2-D or 3-D. For example, we can arrange blocks as "block(0) - block(N)" or "block(0,0) - block(M,N)", while we can arrange threads as "thread(0) - thread(N)", "threads(0,0) - threads(M,N)" or "threads(0,0,0) - threads(M,N,K)".

### 4.2.2 CUDA Programming with Matlab

A Matlab program can utilize a GPU to parallelize a part of the program by calling a "mex" file[22] compiled from CUDA code. The process of compiling the CUDA file and applying the "mex" file is shown in Figure 4.4. While the parallelized parts are translated into CUDA code, the rest of the code is still written in Matlab

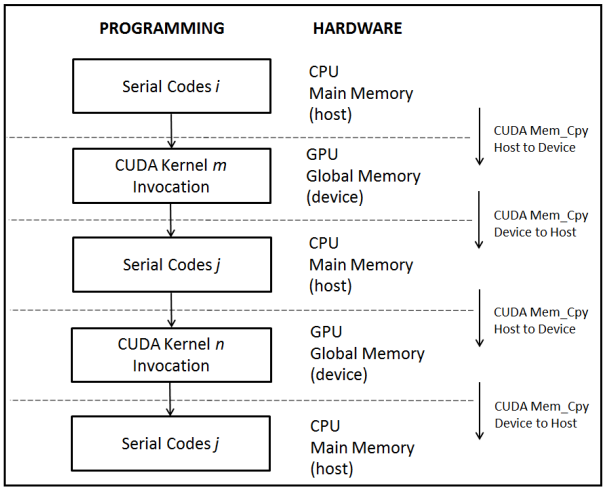


Figure 4.2: An example of CUDA program.

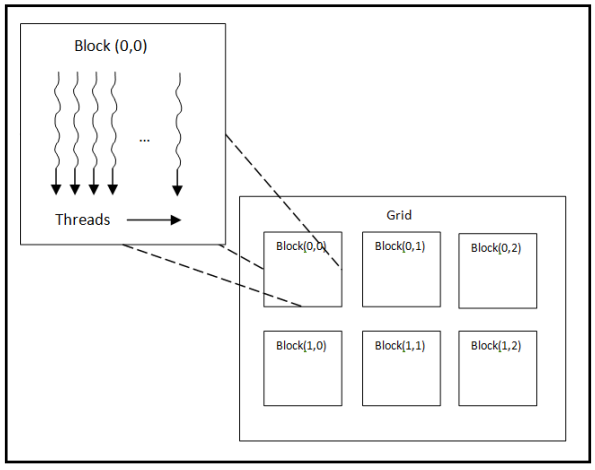


Figure 4.3: CUDA programming model (blocks and threads).

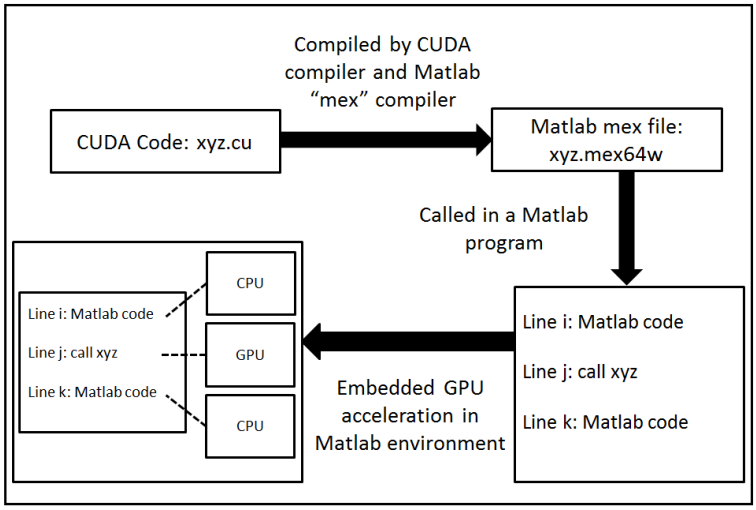


Figure 4.4: The process to embed a CUDA program in Matlab environment by using “mex” file.

script language. Similar to the optical mapping algorithm, before being transplanted in CUDA, most scientific programs are developed by non-professional programmers using high-level languages such as Matlab. The combination of Matlab and CUDA enables a high-performance and easy-developing solution for those GPU accelerating projects. We deployed this solution to accelerate the optical mapping algorithm from the original Matlab optical mapping program.

## 4.3 GPU vs. FPGA

An alternative platform is Field-programmable Gate Array (FPGA). An FPGA can be programmed using a hardware description language (HDL) such as “Verilog”. It also can be programmed in a high level language such as C and synthesized by tools such as Catapult C.

FPGAs and GPUs occupy different points in the spectrum of possible acceler-

ators, both of which are known to achieve substantially better performance than CPUs on certain workloads. Applications typically exhibit vastly different performance characteristics depending on the accelerator. This is an inherent problem attributable to architectural design, middleware support and programming style of the target platform. For the best application-to-accelerator mapping, factors such as programmability, performance, programming cost and sources of overhead in the design flows must be all taken into consideration. In general, FPGAs provide the best expectation of performance and low overhead, while the flexibility of GPU programming allows an easier developing and debugging process.

Our initial implementation presented in this thesis was done on GPUs due to the development complexity. Although the GPU implementation has gained a significant speedup against the original CPU implementation, it is still not fast enough to perform real-time optical mapping algorithm due to the software and driver overhead. Therefore, we will move to an FPGA implementation as the future work in the research of optical mapping algorithm acceleration.

# Chapter 5

# GPU Implementation

## 5.1 Parallelization Overview

In the original Matlab code, the temporal operations and the spatial operations were implemented by using nested iteration loops, as shown in the pseudo code in Figure 5.1. Since the CUDA kernels can only run as the SIMT model, one kernel is not able to evoke several kernels to parallelize the iterations in its kernel function body. Thus, parallelization of the optical mapping algorithm cannot be implemented by directly converting iterations into CUDA kernels.

We re-designed a hierarchy which is more suitable for GPU. In this new hierarchy, CUDA kernels are assigned to pixels of the video with a temporal index and a spatial index. Then, the kernels can execute with SIMT fashion with the input of each pixel located by the temporal index and the spatial index. The temporal index and spatial index can be projected to thread index and thread block index of a thread to locate the correlated output pixel in an array of output data. In this design, each thread works on one pixel. This design has two improvements with respect to efficiency. First, it maximizes the GPU threads occupancy which helps the processor effectively pipeline global memory accesses and thereby masking their cost [23]. Second, it

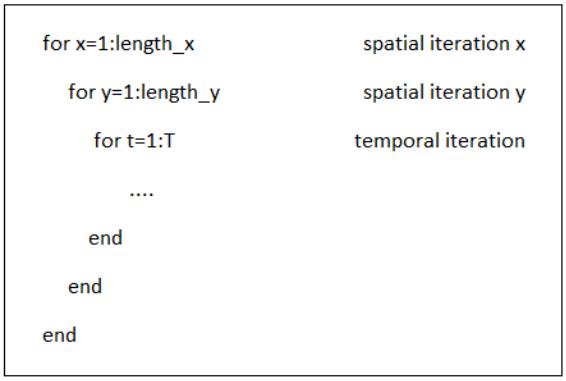


Figure 5.1: Pseudo code of the hierarchy of the original serial code of optical mapping algorithm. In the original serial CPU implementation, the temporal loop is iterated by the outer spatial loops. This hierarchy is not suitable for GPU implementation.

simplifies the operation within each thread to one pixel instead of iterations of many pixels.

However, the description above is only considered in the point of view of the algorithm-level parallelization without considering the GPU hardware attributes and limitations [24]. In order to make the CUDA program reach the GPU peak computing as closely as possible, GPU hardware attributes need to be considered in the design. More optimized designs are discussed in Chapter 6. Before the further discussion of more optimized designs, the implementation of the optical mapping algorithm in CUDA is described in the following sections.

## 5.2 GPU Accelerated Optical Mapping Program Overview

In this section, we present an overview of the GPU accelerated optical mapping program. As described in section 3.4, over 95% of the total run-time is taken by

the phase shift spatial filter and the temporal median filter. Therefore, the program is designed with that the phase shift filter and the temporal median filter parts are parallelized on the GPU while the other parts of the program are still executed on the CPU in serial. As shown in Figure 5.2, for the GPU parallel part, we developed interpolation, padding zeros, FFT, conjugated multiplication, IFFT, "$t_0$ searching", time shift, 2-D filter and temporal median filter CUDA kernels. Before the first CUDA kernel - interpolation, the data are processed on the CPU and stored on the main memory. At this point, the data are copied from the main memory (host memory) to the graphic memory (device memory). After the last CUDA kernel - temporal median filter, the data are copied from the graphic memory back to the main memory. In the following sections of this chapter, the CUDA parallel programming structures for implementing the kernels shown in Figure 5.2 are described.

## 5.3 Phase-Shift Spatial Filtering Implementation

### 5.3.1 Interpolation

The interpolation process consists of three steps: expanding the array by inserting zeros, filtering by a coefficient array, and selecting the right segment of the filtered array. All of these steps can be parallelized. The CUDA program structure of the first step (array expanding) is shown in Figure 5.3. The operation is parallelized both spatially and temporally. The GPU thread is organized in two levels. The top level is a grid which contains some thread blocks, and in each thread block, there are a certain amount of threads that form the second level.

The second step is 1-D filtering, as shown in Figure 5.4. Since this 1-D filtering does not have a feedback loop structure, the output data -y are independent from each other. Each element of the output array can be calculated in a GPU thread. In each thread, there is an iteration of addition computation which is adding the

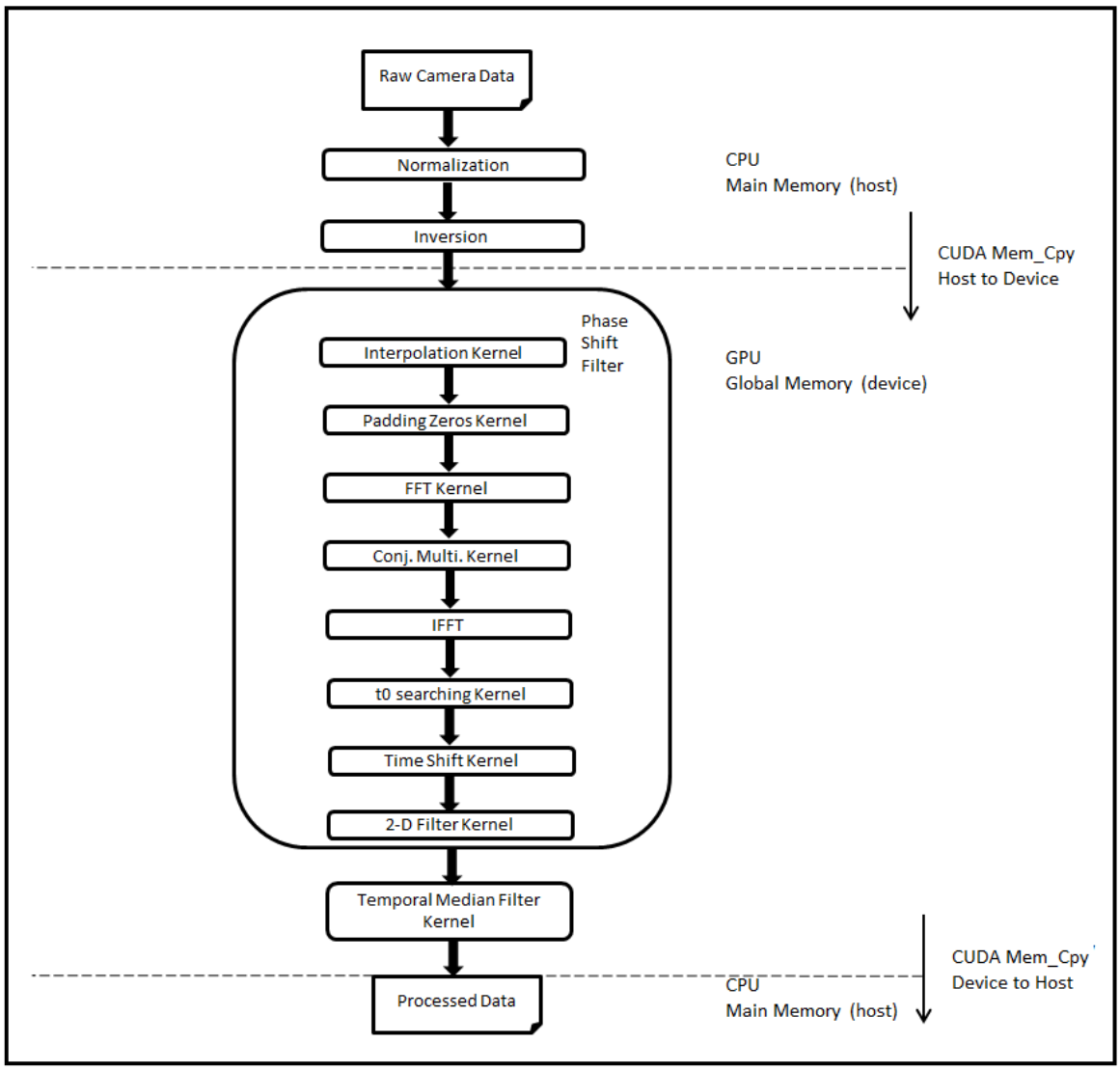


Figure 5.2: GPU accelerated optical mapping program overview. Interpolation, padding zeros, FFT, conjugated multiplication, IFFT, "$t_0$ searching", time shift, 2-D filter and temporal median filter are converted into CUDA parallel code while normalization and inversion are still Matlab serial code.

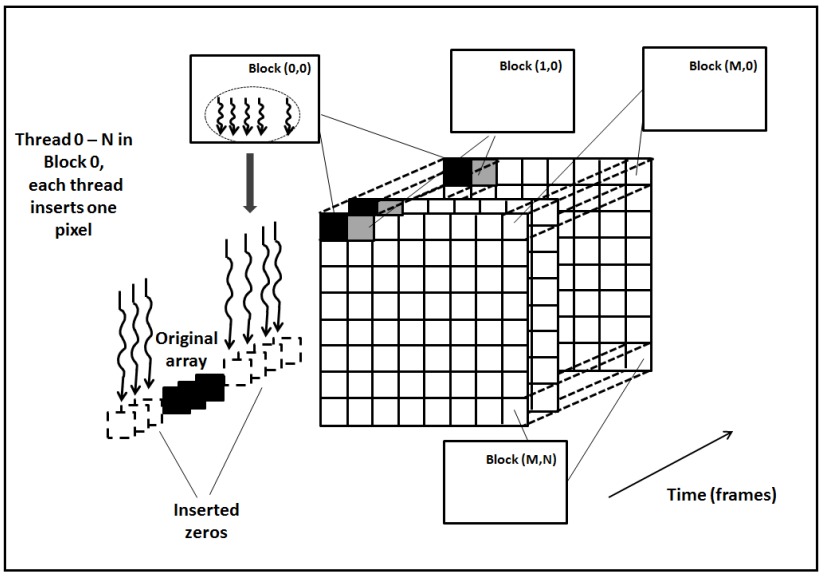


Figure 5.3: CUDA implementation of interpolation step 1.

multiplication of b(n) and x(m), shown in Figure 4.3. This addition computation could be parallelized to reduce the number of iteration, but it would dramatically increase the GPU memory accessing time which can be saved by using iteration in a thread. Usually the array size of b(n) is 89. The parallelization could reduce the times of iteration to $log89 = 7$. However, the memory accessing will increase from 89 to $89 \times (2 - \frac{2}{2^{log89}}) = 176$. The decrease of iteration is 82 while the increase of global memory access is 87. Usually, a global memory access takes much more time than a floating point operation on the GPU. Therefore, it is less efficient to parallelize this addition computation.

The third step which tails the output data is searching in the pre-output array to find the particular elements and saving them in a new array. This process is parallelized similarly to step one.

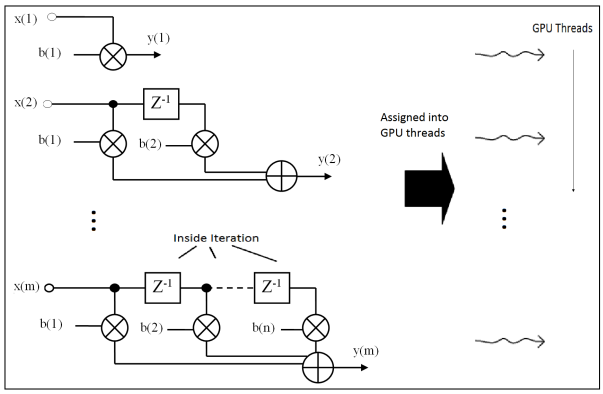


Figure 5.4: CUDA implementation of interpolation step 2.

### 5.3.2 FFT, Conjugated Multiplication and IFFT

The FFT operation is parallelized in CUDA using the "CUFFT" library developed by NVIDIA. This "CUFFT" library provides an interface for operating FFTs on a GPU. This library also supports multiple 1D transforms in parallel, which is suitable for the optical mapping algorithm. Therefore, we deployed the "CUFFT" library computing the FFT of the interpolated image data.

The conjugated multiplication is also parallelized spatially and temporally. However, this operation needs more CUDA threads and generates more data. As described in subsection 3.2.2, for phase-shift filtering, each pixel has its own filter window. Therefore, the number of CUDA threads and the size of output data are both expanded by the factor of the window size. The CUDA program structure of conjugated multiplication is shown in Figure 5.5. The block index "x" represents the index of the pixel within the filter window. The block index "y" represents the index of the window. With this thread assignment, each conjugated multiplication operation is assigned to a thread.

The IFFT operation is similar to the FFT operation. We deployed "CUFFT" library with changing the parameter which controls the transformation direction to

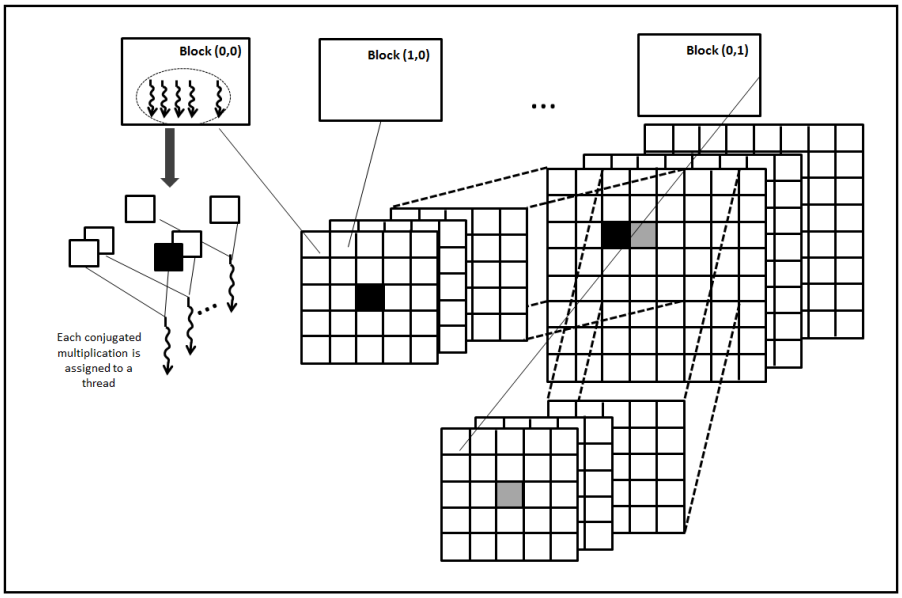


Figure 5.5: The demonstration of the CUDA programming structure of conjugated multiplication.

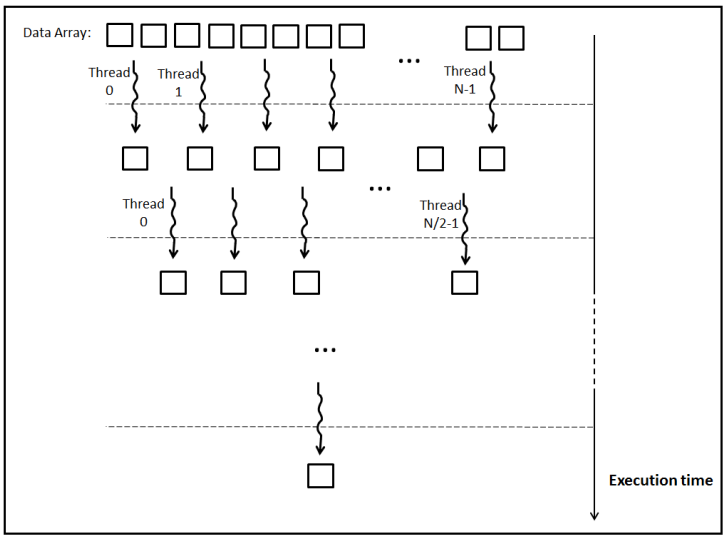


Figure 5.6: Demonstration of CUDA programming structure of "t0 search" operation.

"inverse".

### 5.3.3 Phase Difference "$t_0$" Searching and Shifting

The "$t_0$ search" operation is searching the output array of IFFT for the maximum value. This operation cannot be fully parallelized due to the data dependency. However, the operation can break into comparing two values and saving the greater one. The comparing and saving operation can be iterated until there is only one value left in the array. Each comparing and saving operation can be assigned to a thread. Therefore, the "$t_0$ search" operation can be parallelized as shown in Figure 5.6.

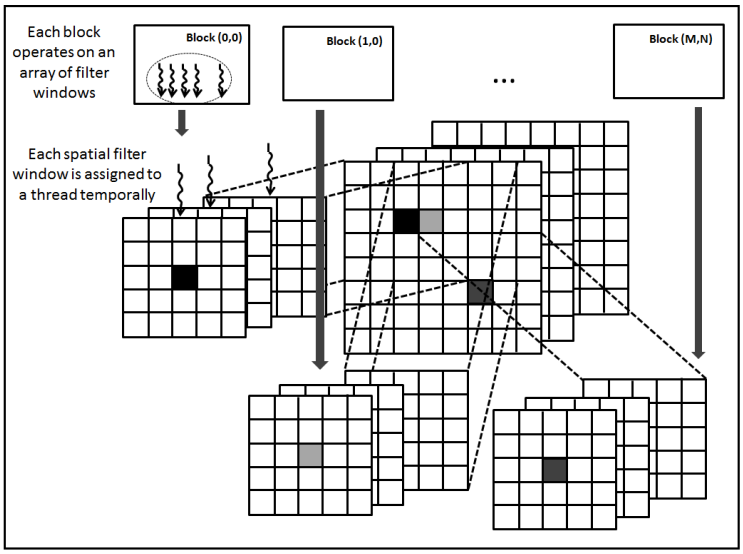


Figure 5.7: Demonstration of CUDA programming structure of 2-D filtering.

### 5.3.4 2-D Filtering

The 2-D filter is parallelized by assigning each filter window a thread, as shown in Figure 5.7. In each thread, a 2-D convolution is computed. In our case, the filter window size is $5 \times 5$ pixels. Thus, the computation intensity of this 2-D convolution is appropriate for executing within a GPU thread. Therefore, we keep the 2-D convolution inside each window in serial, while the filter operation executes on many windows of data in parallel, as demonstrated in Figure 5.7.

## 5.4 Temporal Filtering Implementation

In the temporal filtering, each pixel is replaced by the median value found from a sub-array with length of 7, as described in section 3.3. The temporal filter is parallelized on the GPU by assigning each median value searching operation to a thread, as shown in Figure 5.8.

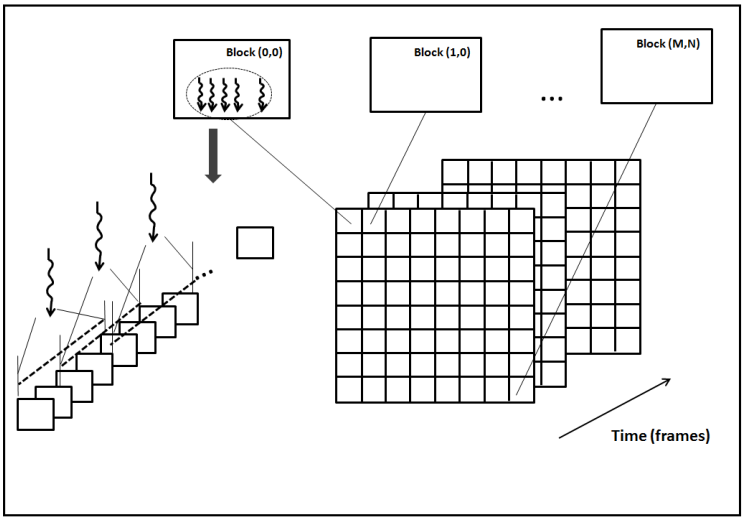


Figure 5.8: Demonstration of CUDA programming structure of temporal median filtering.

## 5.5 Batch Process Strategy

Due to the limitation of graphic memory of GPU (usually 512MB-1024MB, at most 6GB as the largest graphic memory of GPU that has been manufactured), the size of a 1024 frames and $100 \times 100$ frame size video data processed with an interpolation factor of 10 and a $5 \times 5$ 2-D filter window (the total graphic memory required is $(100-4)^2 \times 52 \times 1024 \times 10 \times 4 = 8.75 GBytes$) cannot be processed in fully parallel on a GPU. Therefore, we divided the video into several batches of data so that each batch of data can be processed in parallel. A batch can be seen as a part of the video which is under the limitation of GPU resource. For example $8 \times 8 \times 512$ frames. One batch is processed on the GPU at a time as shown in Figure 5.9. Then the outputs of the process of these batches are combined again as an output video. After each parallel process, the graphic memory is cleared and loaded with the next batch of data.

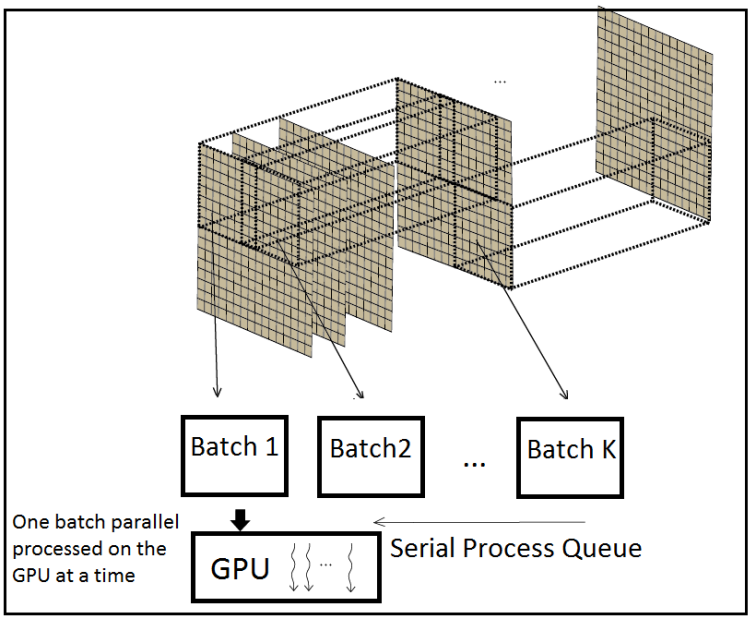


Figure 5.9: Demonstration of batch process.

# Chapter 6

# GPU Optimization

## 6.1 GPU Programming Optimization Overview

In order to reach the peak perform of the GPU as closely as possible, the CUDA optical mapping program needed to be optimized with consideration to memory access efficiency and thread scheduling efficiency. There are many constraints from the underlying hardware and threading model that make the optimization work a multi-variable optimization problem.

There are some principles for CUDA program optimization:

- leverage threads execution to hide the memory latency - make the GPU process a certain amount of threads that occupy the stream processors completely all the time while some of the threads are waiting for memory access;
- Use on-chip memory to reduce bandwidth usage and memory access penalty - use registers and shared memory;
- Arrange threads to avoid memory bank conflicts - make sure the threads are accessing different memory locations at a time;

- Avoid control flow in a CUDA kernel - if there is a control flow, other threads will have to wait for the slowest thread to make the decision.

With these principles, the optimization work can be directed. However, in many cases, there are some specific parameters that determine the run-time significantly. To find out the optimal values for these parameters, we have to use experimental methods. In the two following sections, the details of optimization are described.

## 6.2 Memory Access Efficiency

As discussed in section 3.4, the phase shift spatial filter and temporal median filter take more than 95% of the total run-time of the whole optical mapping algorithm. Therefore, we profiled the CUDA program of these two processes in details to study the possibility of further optimization. The phase shift spatial filter includes interpolation, padding zeros, FFT, conjugated multiplication, IFFT, adding index for "$t_0$ search", "$t_0$ search", time shifting and 2D filtering CUDA kernels. The temporal filter only contains the median filter CUDA kernel. For each part, the percentage of time consumption of each part is shown in Figure 6.1.

In Figure 6.1, "$t_0$ search" consumes the largest amount of time among all the parts of the optical mapping algorithm due to the number of global memory access. Therefore, in the following paragraphs, the optimization of "$t_0$ search" CUDA kernel is discussed.

As described in subsection 5.3.3, the naive implementation of "$t_0$ search" is assigning each two-element comparison to a GPU thread. This design achieves the most occupancy of the GPU stream processors. However, considering memory access efficiency, this approach needs to access the global memory twice in each thread. Thus, the computation to global memory access (CGMA) ratio is 1 to 2 which means execution of one float point operation requires two global memory accesses. Thus, most of the time, the program is waiting for memory access instead of executing

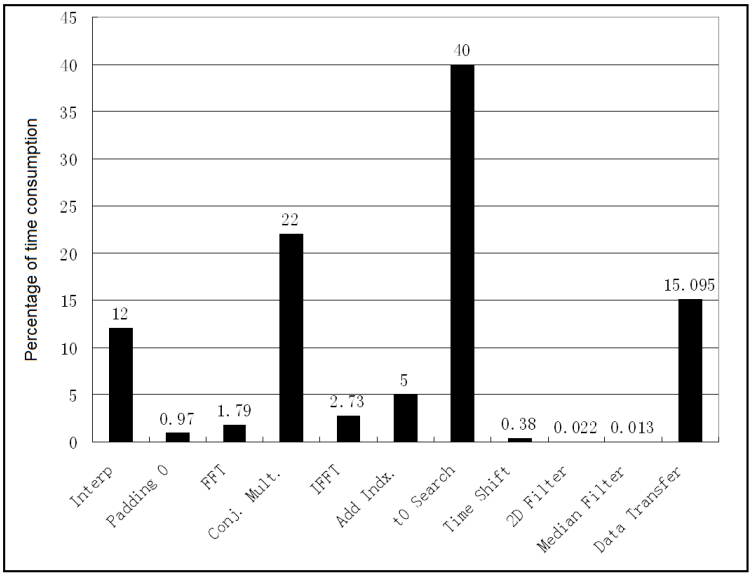


Figure 6.1: Time consumption of the GPU program in percentage with a $9 \times 9 \times 512$ video data on Geforce GT 230M hardware

computing on the stream processors. The memory access becomes a bottleneck of the program at this point. For example, for GT 230M GPU whose global memory access speed is 25.6 GB/s, with a 1:2 CGMA ratio and 4 bytes in each single precision float point datum, the Giga Floating Point Operations Per Cycle (GFLOPS) is $25.6/(4 \times 2) = 3.2$ which is a very small value compared to the peak performance of GT 230M - 158 GFLOPS. Therefore, we re-designed the "t0 search" kernel to obtain higher performance.

To avoid the high latency of global memory accesses, more comparison operations are assigned in each thread, as shown in Figure 6.2, the 4-element approach utilizes the local register when the thread is executing the float point operation. The 4-element approach needs to access the global memory 4 times when it is executing 3 float operations. Thus the CGMA ratio of this approach is 3 to 4 which is higher than the original approach (1 to 2). This is proven by the testing results shown in

Table 6.1 where the time consumption of "$t_0$ search" kernel is reduced to half.

The register re-use approach reduces the global memory access and increases CGMA to achieve better performance. However, this approach increases the serial process in each thread which slows down the operation speed. To find the optimized number of elements that should be processed in one thread, we tested different numbers of pixels of data processed in one thread with a $9 \times 9 \times 512$ video on GT 230M hardware. Referring to Table 6.1, the optimal number of elements found is 32.

Table 6.1: Testing results of the global memory access reduction approach

| Elements Processed in One Thread | "t0 search" Time (ms) |
|---|---|
| 2 | 132.69 |
| 4 | 72.24 |
| 8 | 58.04 |
| 16 | 48.16 |
| **32** | **46.44** |
| 64 | 47.27 |
| 128 | 48.92 |

This experimental result is consistent with the mathematical analysis. We have CGMA (compute to global memory access ratio):

$$CGMA = \frac{BW \times (N-1)}{(4N)} \tag{6.1}$$

and:

$$FLOPS = \frac{PeakFLOPS}{N-1} \tag{6.2}$$

where N is the number of elements per thread. BW is the global memory access bandwidth of the GPU.

When N increases, CGMA increases and FLOPS decreases. The smaller value between CGMA and FLOPS determines the computation speed. Therefore, it achieves the fastest computation speed when $CGMA = FLOPS$.

Thus, we get:

$$BW \times \frac{N-1}{4N} = \frac{PeakFLOPS}{N-1} \tag{6.3}$$

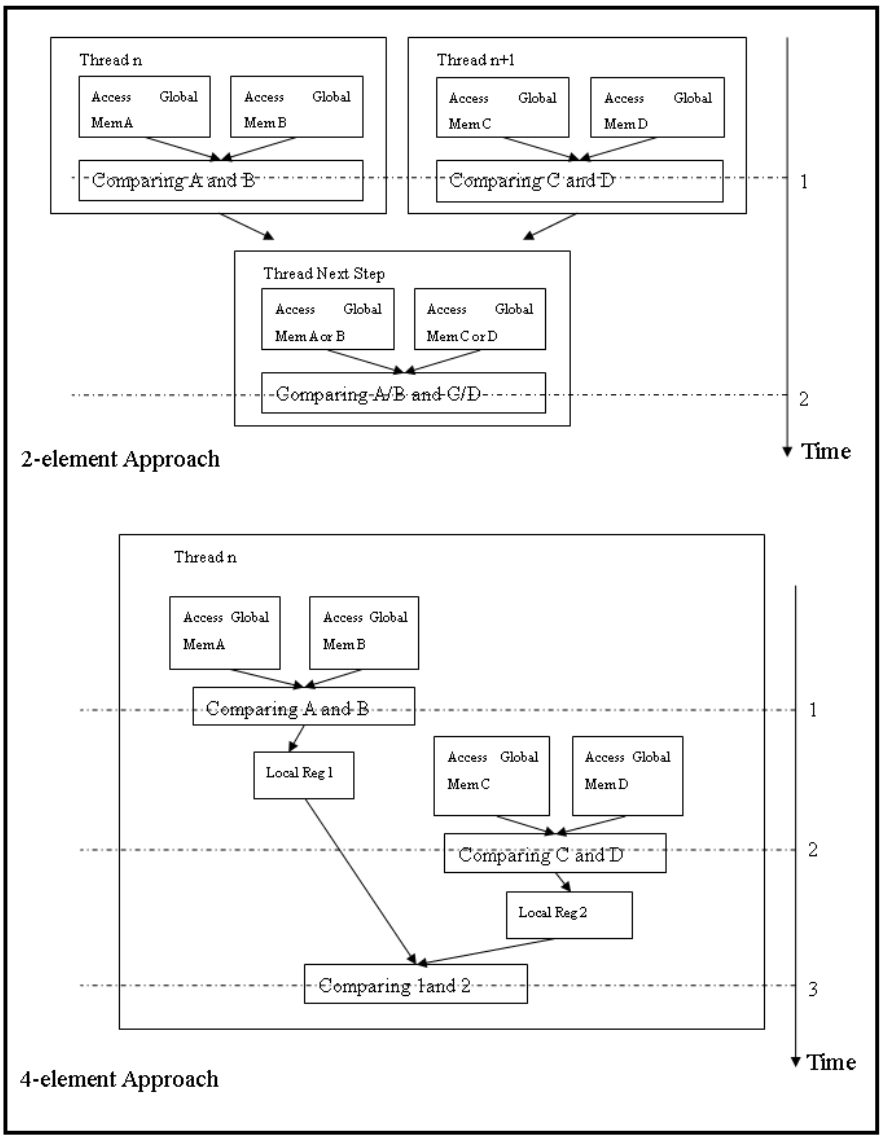


Figure 6.2: Reduction of global memory access optimization for "$t_0$ search".

As long as we know a given GPU's parameter BW and Peak FLOPS, we can solve the equation to obtain the optimal number of elements per thread. For GT230M, $BW = 25.6Gbits/s$, $PeakFlops = 158G/s$, we get N = 26.7. N has to be a power of 2 due to the optical mapping algorithm. Thus, $N = 32$ is the best among the options which is consistent with our experimental results.

## 6.3 Batch Process Input Data Size

As described in section 5.5, the input data of the optical mapping GPU program is divided into batches and each batch is processed on the GPU at a time. The size of the batch can vary as long as it is below the size of the GPU memory. A smaller batch size will cause the stream processors to be under occupancy. Although, a larger batch size may increase the stream processor occupancy, too many threads may exceed the parallel computing ability of the stream processor causing a low performance (e.g. the kernel has to synchronize all the threads at the end, therefore the speed of the kernel execution depends on the slowest thread). Therefore, selecting an optimized batch size for a certain type of GPU is important for computation performance. We evaluated processes with different batch sizes on a Geforce GT 230M GPU. All of these processes operate on a same amount of total data which is a $100 \times 100 \times 1024$ video with $100 \times 100$ pixels for each frame and 1024 frames. The timing results of these processes are shown in Figure 6.3. We tested four sets of batch sizes with four different sizes of frames respectively. For a certain number of frames, we tested different sizes of pixels. Among these sizes of pixels, there is an optimal one which consumes the least amount of time. For example, for the batch size with 128 frames, the optimal one is $17 \times 17$ which needs 211.8211 seconds to process a $100 \times 100 \times 1024$ video. This optimal batch size is at the minimum point of the curve of 128 frames in Figure 6.3.

In Figure 6.3, the batch process timing results are plotted as curves for each num-

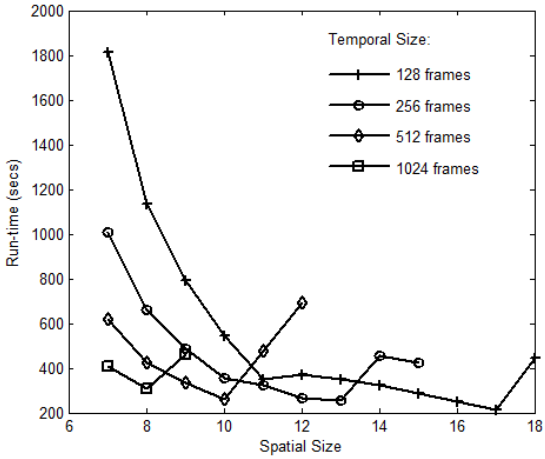


Figure 6.3: Batch process timing results curves.

ber of frames. For each curve, the time consumption decreases with the increment of the width of the frame at the beginning and then reaches the minimal time consumption point. After the minimal point, the time consumption increases tremendously at the next greater width of frame. Table 6.2 shows the total number of threads that the GPU needs to process at a time for different batch sizes. Batch size is calculated by the multiplication of the number of frames (first column) and the number of pixels (first row). In the tables, the "x" means the batch size exceeds the limitation of the GPU hardware. The phenomenon could be observed in these two tables is, when the batch size is small, the GPU will be in an under-utilized state and the batch needs to iterate more along the whole video. When the batch size reaches a certain number, the GPU stream processors will be fully utilized and due to the decrease of batch iteration, the time consumption drops to a minimum value. Once the batch size exceeds this optimal value, the extra threads which exceed the parallel computing ability of the stream processors will be held up waiting for available GPU resource which will decrease the kernel executing speed.

Table 6.2: Total number of threads that the GPU needs to process at a time for different batch sizes. "x" means the batch size exceeds the limitation of the GPU hardware.

| Frames \ Pixels | 7x7 | 8x8 | 9x9 | 10x10 | 11x11 | 12x12 | 13x13 | 14x14 | 15x15 | 16x16 | 17x17 | 18x18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 28800 | 51200 | 80000 | 115200 | 156800 | 204800 | 259200 | 320000 | 387200 | 460800 | **540800** | 627200 |
| 256 | 57600 | 102400 | 160000 | 230400 | 313600 | 409600 | **518400** | 640000 | 774400 | X | X | X |
| 512 | 115200 | 204800 | 320000 | **460800** | 627200 | 819200 | X | X | X | X | X | X |
| 1024 | 230400 | **409600** | 620000 | X | X | X | X | X | X | X | X | X |

# Chapter 7

# Results

## 7.1 Hardware and Software Environments

The hardware setup is: 2.66Ghz Quad-Core Intel Xeon CPU (256KB L2 cache per core, 8 MB shared L3 cache); Nvidia GT230M GPU with 1GB graphic memory referring to Table 7.1 as details; 6GB RAM. The software setup is: Matlab 2009a; Nvidia CUDA Toolkit v3.0.

Table 7.1: Nvidia GT230M Specifications

| | |
|---|---|
| CUDA Cores | 48 |
| Gigaflops | 158 |
| Graphic Memory | 1GB |
| Processor Clock (MHz) | 1100 MHz |
| Memory Interface Width | 128-bit |
| Memory Bandwidth (GB/sec) | 16 (DDR2), 25 (DDR3) |

An additional testing hardware setup is: Tesla Linux Cluster "Lincoln" [25]. The GPU processors on "Lincoln" are Nvidia Tesla S1070 Accelerator Units. The specifications of Nvidia Tesla S1070 are listed in Table 7.2.

Table 7.2: Nvidia Tesla S1070

| | |
|---|---|
| CUDA Cores | 960 (240 per processor) |
| Gigaflops | single: 4147.2; double: 345.6 |
| Graphic Memory | 16 GB (organized as 4.0 GB per GPU) |
| Processor Clock (MHz) | 1440 MHz |
| Memory Interface Width | 4x 512-bit GDDR3 |
| Memory Bandwidth (GB/sec) | 408 GB/sec (102 GB/s per GPU) |

## 7.2 GPU Kernels Timing Results

In this section, we present the timing results of the GPU kernels of phase shifting, 2D filter and the temporal median filter. We tested different lengths of video data on these kernels. The timing results for phase shifting, that includes GPU kernels described in subsection 5.3.1-5.3.3, are shown in Figure 7.1 (a). 7.1 (b) and (c) are the timing results for 2D filter kernel and temporal median filter kernel respectively. 2D filter and temporal median filter kernels achieve more than 200× speedup against the CPU Matlab implementations. The phase shifting kernels only achieve about 2× speedup against the CPU Matlab implementation due to the complexity of computing and high memory access latency.

## 7.3 Complete Optical Mapping GPU Program Performance

In this section, the performance of our GPU optical mapping implementation is presented. We report the performance of our design on different lengths of videos compared to the original serial CPU implementation. We also verified correctness of the output data of our design. As shown in Table 7.3, the mean value of the error of the output data is $1.5204 \times 10^{-5}$ which is tolerable in our experiments.

Since optical mapping is usually applied on an experiment setup in a biomedical

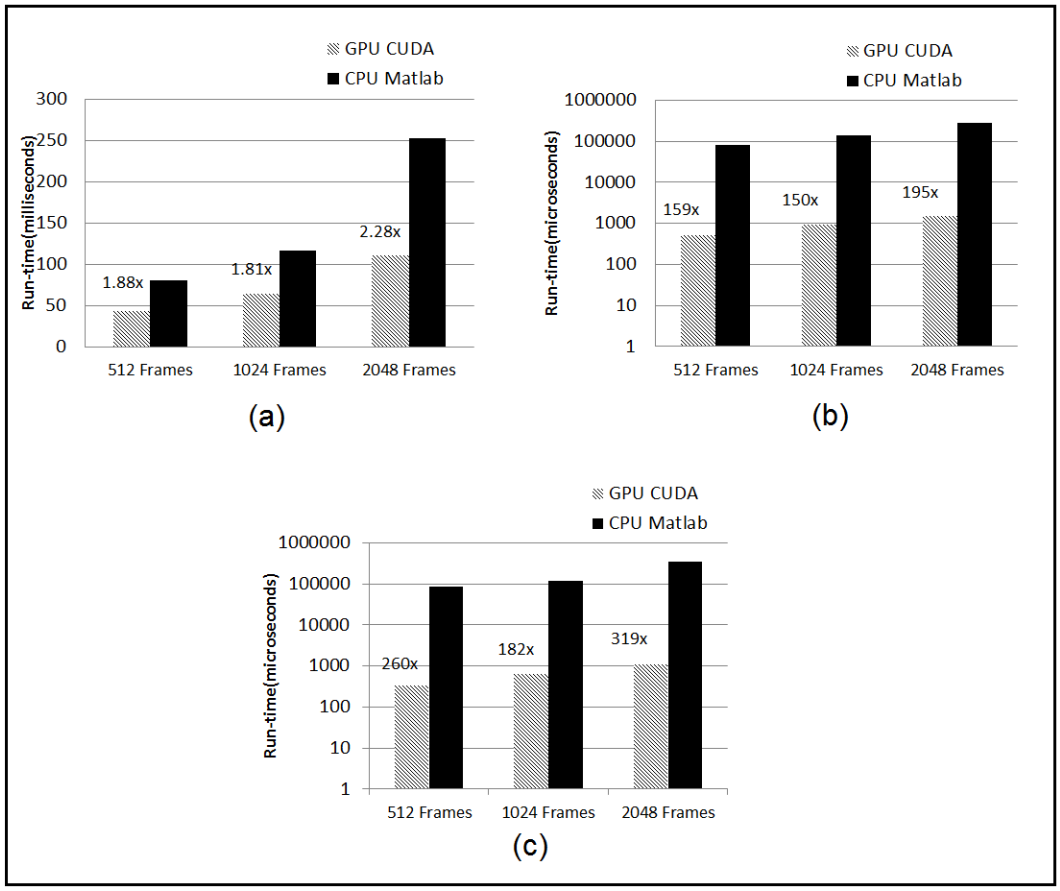


Figure 7.1: GPU kernels timing results. (a) Phase shifting kernels timing. (b) 2D filter kernel timing. (c) Temporal median filter kernel timing.

Table 7.3: Batch Process Error

| Number of Frames | Maximum | Mean Value | Std. Deviation |
|---|---|---|---|
| 128 | 9.9830 | 0.0626 | 0.0423 |
| 256 | 1.7525 | 0.0150 | 0.0039 |
| 512 | 1.5177 | 0.0105 | 0.0034 |
| 1024 | 0.0132 | $1.5204 \times 10^{-5}$ | $1.7809 \times 10^{-5}$ |

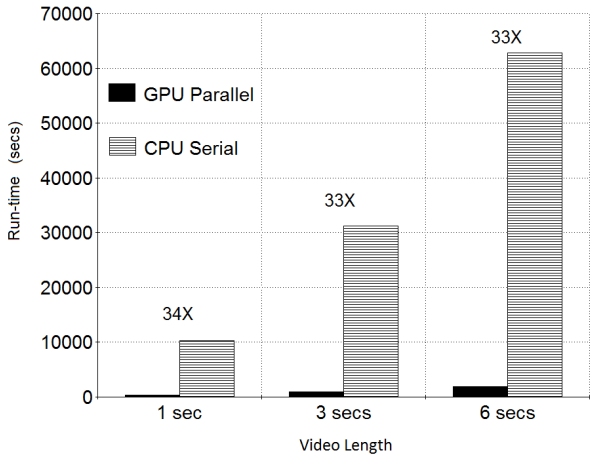


Figure 7.2: Performance on different lengths of videos comparing to CPU serial implementation.

laboratory (described in the Appendix), we choose Nvidia Geforce GT 230M as the hardware to test and measured all the implementations. The results presented in this section and Chapter 6 are obtained using this GPU. The power consumption of GT 230M GPU is 23W which is relatively low among most GPUs. Moreover its size and mobility is suitable for practically implementing a biomedical laboratory experiment system.

The run-time speed up of our design is shown in Figure 7.2. The results show that our GPU implementation has a significant speed up over the original serial CPU implementation. For example, for 6 seconds of video data, the GPU (GT230M) implementation is 33× faster than the CPU (2.66Ghz Quad-Core Intel Xeon) imple-

mentation.

We also analyzed our GPU implementation using "cudaprof", a profiling tool provided by Nvidia for CUDA programs. The analysis shows all of the CUDA kernels of our design, except "conjugated multiplication" and "$t_0$ search", have 100% occupancy. The "conjugated multiplication" CUDA kernel has a GPU occupancies lower than 75%. Therefore, it leaves some room for improvement. The optimization of "$t_0$ search" is discussed and its tradeoffs are explained in section 6.2.

We also tested our GPU implementation on "Lincoln". The run-speed on "Lincoln" is 3X faster than the one on GT230M.

# Chapter 8

# Future Work and Conclusion

## 8.1 Future Work

As shown in section 7.2, the time shifting kernels only have speedups about of 2× against the CPU implementation while the 2-D filter and median filter have speedups about 200×. Therefore, there is still a space for improving the performance of the time shifting kernels. The performance can be further improved by using shared memory in "$t_0$ search" kernel. Also, combining the kernels may be another further improvement possibility. Combining a few consecutive kernels to one kernel with the original few consecutive operations executed within one threads can reduce the overhead of launching kernels. Moreover, multiple-GPU technology can be used in batch process i.e. processing multiple batches on different GPUs separately at the same time.

The FPGA implementation will be a higher-performance option which may achieve the real-time optical mapping goal. FPGA design requires a significant amount of hardware design expertise. These design environments are a far cry from those used to program microcontrollers, microprocessors, digital signal processors and even GPUs. One solution is to use automated converting tools such as "Catapult C" which

can convert programs written in C to hardware description languages for synthesizing the designs on FPGAs.

## 8.2 Conclusion

We developed a GPU implemenation of the optical mapping algorithm for cardiac electrophysiology. We designed a set of CUDA kernels parallelizing the processes of the phase-shifted filter and the temporal median filter. In our GPU implementation, we embedded the GPU accelerated image processing part in the Matlab environment by using "mex" files. Thus, the non-time consuming analysis part written in Matlab does not need to be translated into CUDA.

We obtained a 33× speedup by running the GPU accelerated program on a GT 230M GPU. We have found an optimal memory access method using the registers on the GPU stream processors for the "$t_0$ search" CUDA kernel in our design. We have also found an optimal input batch size for the parallel process on GPU associated with stream processors occupancy and threads scheduling. Therefore, we conclude that decreasing or masking the memory access and increasing the GPU stream processors occupancy are the methods to further improve the GPU computing performance.

# Appendix A

# Experiment Setup

Our setup employs two stereoscope lenses that focus the image onto the CMOS chip of the camera. The fluorescent probe is excited with a 470 nm wavelength 3.6W LED lamp. Fluorescence is collected with a custom built tandem lens optical setup (two 1× planapo objectives), filtered with a 610 nm wavelength long-pass filter, and focused on the imaging sensor. Digital 14-bit images are recorded at 1,000 frames per second with a high speed CMOS camera with a spatial resolution of $100 \times 100$ pixels. Multiple cameras can be arranged for panoramic imaging of the heart. This is useful to observe global activation patterns and times, and to identify the origin of spontaneous or reentrant arrhythmias. Figure A.1 provides a visual depiction of this setup.

The raw image intensity signals, which are collected by the high speed camera, are covered by noise. Thus, a signal conditioning process needs to be done to reduce the noise. The effect of signal conditioning is shown in Figure A.2. This process includes spatial phase-shift and temporal median filtering.

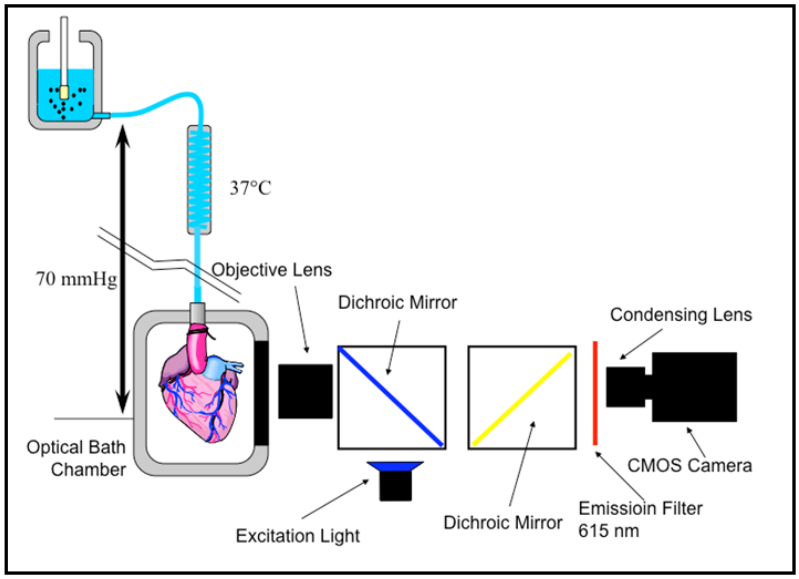


Figure A.1: Optical mapping experiment setup.

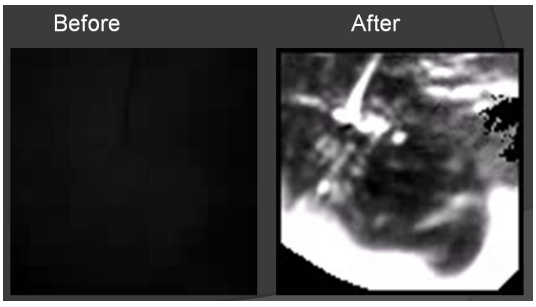


Figure A.2: Signal conditioning effect. The left picture is the picture covered with noise before signal conditioning. The right picture is noiseless after signal conditioning.

# Bibliography


[1] B. F. Hoffman, P. F. Cranefield, E. Lepeschkin, B. Surawicz, and H. C. Herrlich, "Comparison of cardiac monophasic action potentials recorded by intracellular and suction electrodes," Am J Physiol, vol. 196, pp. 1297-1301, Jun 1959.

[2] A. V. Sahakian, M. S. Peterson, S. Shkurovich, M. Hamer, T. Votapka, T. Ji, and S. Swiryn, "A simultaneous multichannel monophasic action potential electrode array for in vivo epicardial repolarization mapping," IEEE Trans Biomed Eng, vol. 48, pp. 345-353, Mar 2001.

[3] I. R. Efimov, D. T. Huang, J. M. Rendt, and G. Salama, "Optical mapping of repolarization and refractoriness from intact hearts," Circulation, vol. 90, pp. 1469-1480, 1994.

[4] I. R. Efimov, V. P. Nikolski, and G. Salama, "Optical Imaging of the Heart," Circ Res, vol. 95, pp. 21-33, July 9, 2004 2004.

[5] X. Xue, A. Cheryauka, D. Tubbs, "Acceleration of fluoro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: A simulation study," SPIE Medical Imaging 2006, 2006.

[6] K. Chidlow, T. Mller, "Rapid emission tomography reconstruction," International Workshop on Volume Graphics, 2003.

[7] T. Schiwietz, T. Chang, P. Speier, R. Westermann, "MR image reconstruction using the GPU," SPIE Medical Imaging 2006, 2006

[8] T. Sumanaweera, D. Liu, "Medical image reconstruction with the FFT," M. Pharr (Ed.), GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, Addison-Wesley,2005, pp. 765784.

[9] Fred V. Lionetti, "GPU Accelerated Cardiac Electrophysiology," Thesis for Master of Science, University of California, San Diego, 2010

[10] A. Ruiz, M. Ujaldon, J.A. Andrades, J. Becerra, Kun Huang, T. Pan, J. Saltz, "The GPU on biomedical image processing for color and phenotype analysis," Bioinformatics and Bioengineering, 2007, pp. 1124-1128

[11] Timothy D. R. Hartley, Umit Catalyurek, Antonio Ruiz, Francisco Igual, Rafael Mayo, Manuel Ujaldon, "Biomedical image analysis on a cooperative cluster of GPUs and multicores," In ICS '08: Proceedings of the 22nd annual international conference on Supercomputing (2008), pp. 15-25.

[12] D. Sung, J. Somayajula-Jagai, P. Cosman, R. Mills, and A. D. McCulloch, "Phase shifting prior to spatial filtering enhances optical recordings of cardiac action potential propagation," Ann Biomed Eng, vol. 29, pp. 854-861, Oct 2001.

[13] D. Sung, R. W. Mills, J. Schettler, S. M. Narayan, J. H. Omens, and A. D. McCulloch, "Ventricular filling slows epicardial conduction and increases action potential duration in an optical mapping study of the isolated rabbit heart," J Cardiovasc Electrophysiol, vol. 14, pp. 739-749, Jul 2003.

[14] R. J. Butera, Jr., C. G. Wilson, C. A. DelNegro, and J. C. Smith, "A methodology for achieving high-speed rates for artificial conductance injection in electrically excitable biological cells," IEEE Transactions on Biomedical Engineering, vol. 48, pp. 1460-1470, 2001.

[15] A. D. Dorval, D. J. Christini, and J. A. White, "Real-Time linux dynamic clamp: a fast and flexible way to construct virtual ion channels in living cells," Ann Biomed Eng, vol. 29, pp. 897-907, Oct 2001.

[16] R. Wilders, "Dynamic clamp: a powerful tool in cardiac electrophysiology," J Physiol, vol. 576, pp. 349-359, Oct 15 2006.

[17] D. J. Christini, K. M. Stein, S. M. Markowitz, S. Mittal, D. J. Slotwiner, M. A. Scheiner, S. Iwai, and B. B. Lerman, "Nonlinear-dynamical arrhythmia control in humans," Proc Natl Acad Sci U S A, vol. 98, pp. 5827-5832, May 8 2001.

[18] B. Echebarria and A. Karma, "Spatiotemporal control of cardiac alternans," Chaos, vol. 12, pp. 923-930, Sep 2002.

[19] P. V. Bayly, B. H. KenKnight, J. M. Rogers, R. E. Hillsley, R. E. Ideker, and W. M. Smith, "Estimation of conduction velocity vector fields from epicardial mapping data," IEEE Trans Biomed Eng, vol. 45, pp. 563-71, May 1998.

[20] Ni, Q., R. S. MacLeod, R. L. Lux, and B. Taccardi, A novel interpolation method for electric potential fields in the heart during excitation, Ann. Biomed. Eng. 26:597-607, 1998.

[21] Digital Signal Processing Committee, Program for Digital Signal Processing, IEEE Press, 1979, Algorithm 8.1

[22] Product Support: MEX-files Guide, The MathWorks, Inc. [Online]. Available: http://www.mathworks.com/support/tech-notes/1600/1605.html

[23] D. Hefenbrock, J. Oberg, N.T.Thanh, R. Kastner, S.-B. Baden, "Accelerating Viola-Jones Face Detection to FPGA-Level using GPUs," Field-Programmable Custom Computing Machines, 2010, pp. 11-18.

[24] NVIDIA CUDA Programming Guide, NVIDIA, Feb 2010, [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf

[25] NCSA Intel 64 Tesla Linux Cluster Lincoln Technical Summary, NCSA, University of Illinois. [Online]. Available: http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/Intel64TeslaCluster/TechSummary