Implementation Of Watermarking Technique Using NVIDIA GPUs

By

Ms.Aspriha Das

08MCES52

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2011

Implementation of watermarking Technique using NVIDIA GPUs

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Aspriha Das

(08MCES52)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2011

Dedication

To my Mother Mrs. Susmita Das for her unconditional love and to my loving Husband Mr. Rajiv Trivedi for his endless support and being a great motivator in my life.

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Computer Science and Engineering at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgement has been made in the text to all other material used.

Ms.Aspriha Das

Certificate

This is to certify that the Major Project entitled **"Implementation of watermarking Technique using NVIDIA GPUs."** submitted by **Ms.Aspriha Das** (08MCES52), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-I, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Samir Patel
Guide, Senior Associate Professor,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad.

Dr.S.N.Pradhan
Professor and PG-Coordinator,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad.

Prof.D.J.Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad.

Dr.K.Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad.

Abstract

The proliferation of digitized media due to the rapid growth of networked multimedia systems, has created an urgent need for fast copyright enforcement technologies that can protect copyright ownership of multimedia objects. Digital image watermarking is one such technology that has been developed to protect digital images from illegal manipulations. In particular, watermarking algorithms which are based on the discrete wavelet transform have been widely recognized due to its excellent spatial localization, frequency spread, and multi-resolution characteristics, which are similar to the theoretical models of the human visual system. In this project,an imperceptible and a robust DWT based watermarking algorithm have been develop using MATLAB and CUDA - **Compute Unified Device Architecture**. MATLAB is a very powerful tool which deals with immensely large image manipulations. There are several ways available to write code in MATLAB for large calculations but they are very time consuming. As a solution, the algorithm is implemented in MEX and CUDA_MEX, which takes less time compared to M-files implementation. The comparison results with respect to time in between the M-files, MEX-files, CUDA_MEX file has been measured. The parallel CUDA_MEX code will run in many threads on an NVIDIA graphical processing unit. By decreasing the completion time of the longest part of the code, largest performance increase can be achieved.

Acknowledgements

Theory and practical are essential and complimentary to each other. Success is possible only when combined with hard work and dedication. I achieved great pleasure in completing this thesis work and I want to take the opportunity to say thanks to all of them, who have, in some way or other helped me to achieve it.

I feel great pride in having done my research work under the efficient supervision of my guide and mentor **Prof. Samir Patel**, Senior Associate Professor, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad. I would like to thank him for his valuable guidance,suggestions and the pain he has taken to go through the project work and make necessary amendments as and when needed.

I express my deep sense of gratitude to **Dr. S.N.Pradhan**, Professor and PG-Coordinator of Department of Computer Engineering, Institute of Technology,Nirma University, Ahmedabad for his exceptional support and continual encouragement throughout the project work.

I would like take this opportunity to thank **Dr. Ketan Kotecha**, Director, Institute of Technology, Nirma University, Ahmedabad for his unmentionable support, providing basic infrastructure and healthy research environment. I also would like to thank my Institution, all my faculty members in Department of Computer Science and my colleagues without whom this project would have been a distant reality.

Words are not enough to express my deepest sense of affection towards my Mother, **Mrs. Susmita Das**. It was her belief and motivation that I have been able to come so far successfully. All I can say that without her faith and care it was not possible for me to achieve my goal.

Last but not the least; I would like to thank my Parents and Parents-in-law especially my

Husband and all my family members for their great love, support and encouragement because of whom I am able to complete my dissertation work successfully.

Above all I want to thank GOD, for giving me this opportunity to achieve my goal.

- Ms.Aspriha Das

08MCES52

Contents

List of Tables

List of Figures

Abbreviations

ALU	Arithmetic Logic Units
CUDA	Compute unified Device Architechture
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
FWT	Fast Wavelet Transform
GPU	Graphics Processing Unit
MEX	MATLAB executable
SIMD	single instruction is performed over multiple data

Chapter 1

Introduction

1.1 General

Digital image watermarking is one such technology that has been developed to protect digital images from illegal manipulations. Its expected that digital watermarking will have a wide span of practical applications such as digital cameras, medical imaging, image databases, and video-on-demand systems, among many others. In accordance to achieve high speed for respective application with taking care of computing infrastructure , one approach is CUDA (Compute Unified Device Architecture) which could be use as back end and MATLAB as front end in watermarking application. MATLAB is a very powerful computer program used to solve complicated mathematical problems. We could use MEX-files, which can be written in the C-like programming language and run through the MATLAB command window just like the built-in functions of MATLAB. In order to speed up MATLAB, the main code could be written in M- files and profiled to find the main bottlenecks in the sequential code. Finally the MEX-file could be written including CUDA code to decrease the compilation time of the algorithm from MATLAB. The algorithm in Cuda_Mex, takes less time compared to M-files and Mex-files implementation. In this report, the comparison results with respect to time in between the M-files, MEX-files and Cuda_Mex files has been taken. This parallel code will run in many threads on an NVIDIA

graphical processing unit. By decreasing the completion time of the longest part of the code the largest performance increase can be achieved in the watermarking Process.

In order to digital watermarking method to be effective it should be imperceptible, and robust. Current digital image watermarking techniques can be grouped into two major classes: spatial-domain and frequency-domain watermarking techniques. Compared to spatial domain techniques, frequency-domain watermarking techniques proved to be more effective with respect to achieving the imperceptibility and robustness requirements of digital watermarking algorithms. Commonly used frequency-domain transforms include the Discrete Wavelet Transform (DWT), the Discrete Cosine Transform (DCT) and Discrete Fourier Transform (DFT).,[6] However, DWT has been used in digital image watermarking more frequently due to its excellent spatial localization and multi-resolution characteristics, which are similar to the theoretical models of the human visual system.,[8]

1.2 Objective of Study

- MATLAB is a very powerful tool which deals with immensely large image manipulations. There are several ways available to write code in MATLAB but they are very time consuming. There is a way to exploit GPU capabilities in MATLAB, is through the use of CUDA. We can use MEX files to perform computations on the GPU using CUDA.

- Support heterogeneous type of computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU.

- To compile this kind of application from MATLAB, we need a specific perl script. Which uses the NVCC compiler to compile the parallel part of the code and the C compiler to compile the serial portion of the code.

- Combination of MATLAB and CUDA enables high-productivity and high-performance

solutions.

- For growing demand of digital data processing, authentication is most important and it should be fast enough to deal with the current demand. Thus,programmers can have a higher performance during watermarking process.

- Enable heterogeneous systems (i.e. CPU + GPU) CPU and GPU are separate devices with separate DRAMs.

1.3 Scope of Work

- Data conversion form double to single precision before sending the data to the GPU. As MATLAB deals with double precision and GPU deals with single precision, conversion is very important. The current generation of GPU from NVIDIA has support for IEEE single precision. Double precision support will be available in the next generation in the coming years.

- MATLAB is a column major and CUDA is row major, so to access a specific location indexing is needed.

1.4 Thesis Organization

- The dissertation started with a general introduction about background, types of digital watermarking technique and the concepts and basic features of watermarking.

- Literature survey on CUDA describes differences between CPU and GPU. It also describes CUDA Programming Model ,Memory Architecture, Hardware implementation and various memory optimization techniques. Along with that it describes about advantages of MATLAB and CUDA integration platform.

- In the second part of the report is a description that expounded the essential knowledge about the wavelet transform that will be utilized to define an approach and

foundation to implementation.

- The third part describe about the development environment and the algorithm for the watermarking.
- The fourth part is the most important one, where the implementation of the algorithm has been done in M-Files,MEX-Files and CUDA_MEX Files.
- The sixth part, where we have done the comparison of the result of all the implementation in terms of time.
- Conclusion and Future Work concluding remarks and scope for future work is presented.

Chapter 2

Literature Survey

2.1 Introduction To NVIDIA CUDA

Parallel Computing involves use of multiple cores at the same time for computation purpose. Parallel computing has evolved because, there is a limit to the clock frequency at which a core can work at(otherwise it gets over-heated). Thus, the solution given was to use multiple cores at the same time. Supercomputers like PARAM use 4608 cores.[20]

GPUs (Graphics Processing Unit), have multiple cores within (around 128 or more). But these are not used when graphics problems are not present and merely computation is to be done. Thus, the power of a PC can be greatly increased by making use of these GPUs. For utilization of a GPU, which is basically a co-processor, we have to program accordingly as normal programs do not support parallel computation.

NVIDIA has come up with CUDA (compute unified device architecture) technology, which is the only C language environment that unlocks the processing power of the GPU to solve complex compute-intensive problems. Thus, using such tactics, utilization of resources at our disposal increases.

The tremendous increase in power, in today's GPUs has led to a need for ways to utilize them for non-graphics applications. For example, the Nvidia 8800GTX is capable

of a theoretical maximum of 350 GFLOPS at a cost of $570 (April 2007). Compared to this a 3.0 GHz Intel Core2 Duo, which is capable of around 40 GFLOPS, at a cost of about $266 (April 2007). This gives $.95/GFLOP for the 8800GTX and $6.65/GFLOP for the Core 2 Duo. Memory bandwidth is also much higher on the GPU: 86.4 GB/sec vs. 6GB/sec. Figure 2.1 is an indicator of the processing power of NVIDIA GPU against Intel:[20],[15],[14]

Figure 2.1: processing power of NVIDIA GPU against Intel

2.2 GPU Hardware Architecture

GPUs are designed for parallel computing with an emphasis on arithmetic operations, which originate from their main purpose - to compute graphic scene which is finally displayed. Current graphic accelerators consist of several multi- processors (up to 30). Each multiprocessor contains several (e.g., 8, 12 or 16) Arithmetic Logic Units (ALUs). Up to 480 processors is in total on the current high-end GPUs. On the other hand, multi-core CPUs usually consist of 2-8 cores. These cores usually work asynchronously and independently.[2] Thus, each core can execute different instructions over different data at the same time. More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations. The GPU Devotes More Transistors to Data Processing like in figure 2.2.[1]Data-parallel processing maps data elements to paral-

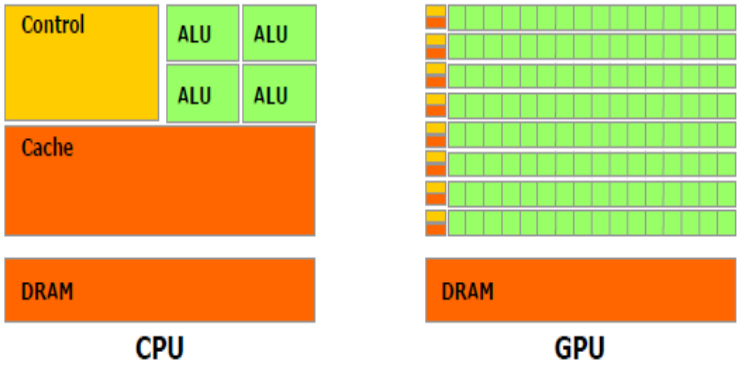

Figure 2.2: The GPU Devotes More Transistors to Data Processing

lel processing threads. Many applications that process large data sets can use a data-parallel programming model to speed up the computations.[4]

Following points make GPUs different from CPUs.

a. CPUs use SIMD (single instruction is performed over multiple data) vector units, and GPUs use SIMT (single instruction, multiple threads) for scalar thread processing.

b. CPUs use caches to increase their performance owing to reduced memory access latencies and GPUs use caches or shared memory to increase memory band- width.

c. CPU cores are designed to execute a single thread of sequential instructions with maximum speed and GPUs are designed for fast execution of many parallel instruction threads.[4]

d. CPUs can execute 1-2 threads per core, while GPUs can maintain up to 1024 threads per each multiprocessor. Switching from one thread to another costs hundreds of cycles to CPUs, but GPUs switch several threads per cycle.

e. CPUs reduce memory access latencies using large caches as well as branch prediction. GPUs solve the problem of memory access latencies using simultaneous execution of thousands threads when one thread is waiting for data from mem- ory, a GPU can execute another thread without latencies.[10]

f. GPUs contain extensive support of Stream Processing paradigm. It is related to SIMD (Single Instruction, Multiple Data) processing.

2.3 Technical Background of CUDA

CUDA technology gives computationally intensive applications access to the tremendous processing power of the recent GPUs through a C like programming interface. The GPU is especially well suited to address problems that can be expressed as data parallel computations with high arithmetic intensity. Because the same program is executed for each data element, there is a lower requirement for sophisticated control, and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches. Data parallel processing maps data elements to parallel processing threads. Many applications that process large data sets such as arrays can use a data parallel programming model to speed up the computations. In fact, many algorithms are accelerated by data parallel processing, from general signal processing to physics simulation or to computational biology. Up until now, however, accessing all that computational power packed into the GPU and efficiently leveraging it for non-graphics applications remained tricky.[15] CUDA exposes several hardware features that are not available via the graphics API. The most significant of these is shared memory, which is a small (currently 16KB per multiprocessor) area of on-chip memory which can be accessed in parallel by blocks of threads. This allows caching of frequently used data and can provide large speedups over using textures to access data. Combined with a thread synchronization primitive, this allows co-operative parallel processing of on-chip data, greatly reducing the expensive off-chip bandwidth requirements of many parallel algorithms. This benefits a number of common applications such as linear algebra, Fast Fourier Transforms, and image processing application.

The advantages of CUDA architecture are the following:

a. Hardware abstraction: NVIDIA has hidden the architectures of its GPUs beneath an

application programming interface (API). Programmers need not to know the complex details of the GPU hardware.

b. Comfortable development environment: CUDA programming interface provides a relatively simple path for users familiar with the C programming language to easily write programs for execution by the device.

c. General DRAM memory addressing: CUDA provides general DRAM memory addressing for more programming flexibility. From a programming perspective, GPU can gather data from any location in DRAM, and also scatter data to any location in DRAM, just like on a CPU.

d. Parallel data cache: CUDA has on-chip shared memory with very fast general read and write access, that threads use to share data with each other.

e. Thread synchronization: Synchronization within a thread block is entirely managed in hardware. Synchronization among thread blocks is achieved by allowing a kernel to complete and starting a new kernel.

2.3.1 Hardware Implementation : Memory Architecture

The local, global, constant, and texture spaces are regions of device memory mention in figure 2.3. Each multiprocessor has:[20],[1]

- A set of 32-bit registers per processor.
- On-chip shared memory where the shared memory space resides.
- A read-only constant cache to speed up access to the constant memory space.
- A read-only texture cache to speed up access to the texture memory space [7].

All of the above points shows that CUDA implies a special approach to development, slightly different from CPU programming. You must be mindful of different memory types,

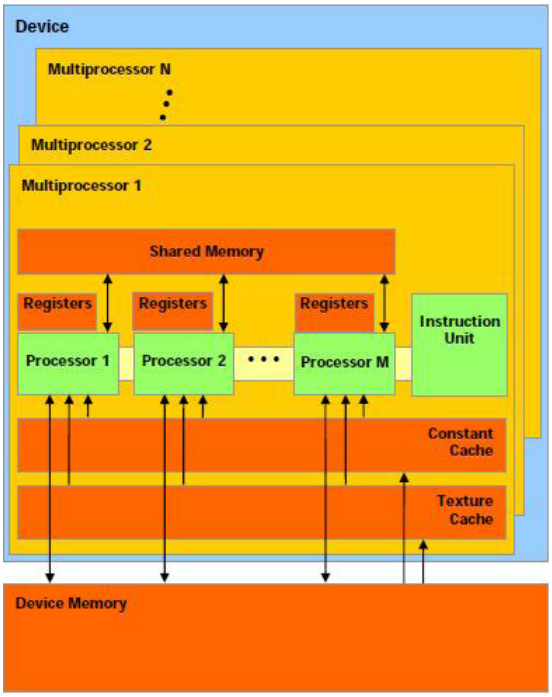

Figure 2.3: A set of SIMD multiprocessor with on-chip shared memory

of the fact that local and global memory is not cached and that their access latencies are much higher than in register memory, as it's physically located in separate chips.

2.3.2 Memory Hierarchy

Each thread has private local memory. Each thread block has shared memory visible to all threads of the block and with the same lifetime as the block. All threads have access to the same global memory.CUDA threads may access data from multiple memory spaces during their execution as illustrated by Figure 2.4.[11],[6],[12]

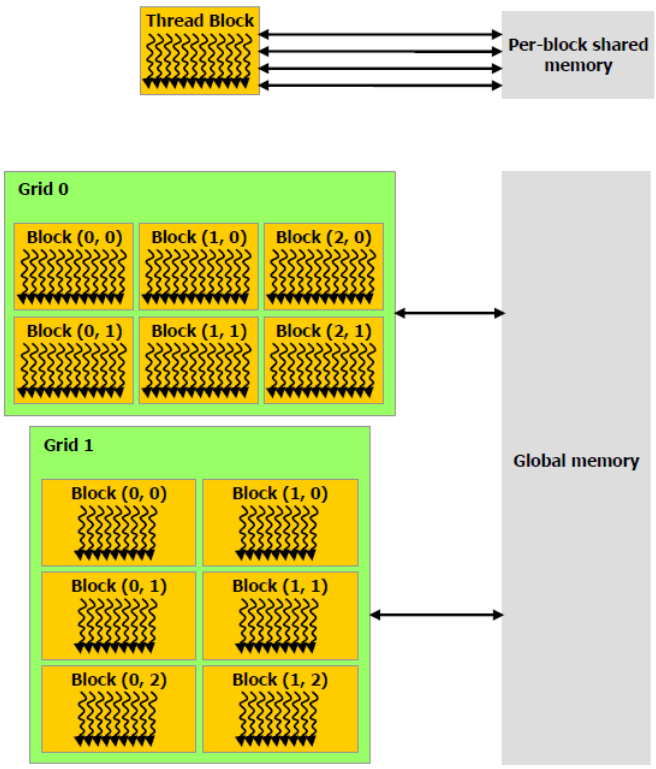

Figure 2.4: Memory Hierarchy

There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory spaces.The global, constant, and texture memory spaces are persistent across kernel launches by the same application.[13]

2.3.3 Heterogeneous Programming

CUDA programming model assumes that the CUDA threads execute on a physically separate device that operates as a coprocessor to the host running the C program. This also assumes that both the host and the device maintain their own separate memory spaces in DRAM, referred to as host memory and device memory, respectively. Therefore, a program manages the global, constant, and texture memory spaces visible to kernels through calls to the CUDA runtime. This includes device memory allocation and deallocation as well as

data transfer between host and device memory.

The programmer supplies a single source program encompassing both host (CPU) and kernel (GPU) code. The host code transfers data to and from the GPUs global memory and initiates the kernel code by calling a function. A kernel runs several blocks of threads and each thread performs a single computation. These threads are organized into a hierarchy of grids of thread blocks.The maximum number of threads per block is 512. Kernels are separated by an inter-kernel synchronization barrier. Threads may access data from multiple memory spaces during their execution. Each thread has a private local memory. Each thread block has a shared memory visible to all threads of the block that has the same lifetime as the block. Finally, all threads have access to the same global memory.

2.3.4 CUDA Programming Model

The CUDA programming model is similar to the familiar SPMD (single-program multiple data) model in their styles.

The advent of multicore CPUs and many core GPUs means that mainstream processor chips are now parallel systems. Furthermore, their parallelism continues to scale with Moore.s law. The challenge is to develop application software that transparently scales its parallelism to leverage the increasing number of processor cores.The CUDA parallel programming model is designed to overcome this challenge while maintaining a low learning curve for programmers familiar with standard programming languages such as C.[11]Indeed, each block of threads can be scheduled on any of the available processor cores, in any order, concurrently or sequentially, so that a compiled CUDA program can execute on any number of processor cores as illustrated by the Figure 2.5, and only the runtime system needs to know the physical processor count. This scalable programming model allows the CUDA architecture to span a wide.[13]

These abstractions provide fine-grained data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. They guide the programmer to

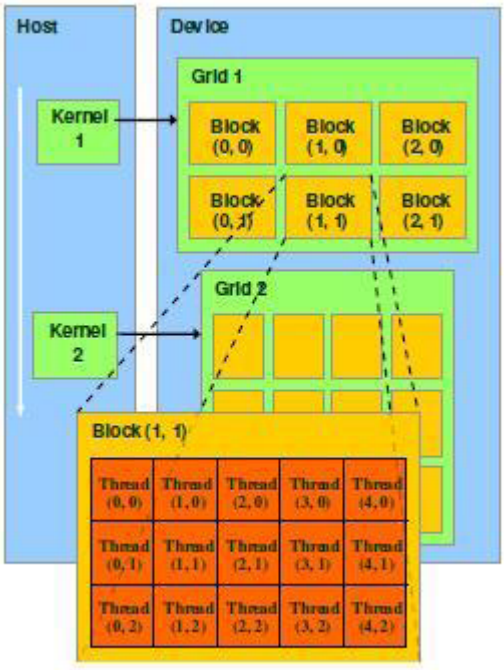

Figure 2.5: CUDA Programming Model

partition the problem into coarse sub-problems that can be solved independently in parallel by blocks of threads, and each sub-problem into finer pieces that can be solved cooperatively in parallel by all threads within the block.

2.3.5 Built-In Variables

- **gridDim** - This variable is of type dim3 and contains the dimensions of the grid.
- **blockIdx** - This variable is of type uint3 and contains the block index within the grid.
- **blockDim** - This variable is of type dim3 and contains the dimensions of the block.
- **threadIdx** - This variable is of type uint3 and contains the thread index within the block.
- **WarpSize** - This variable is of type int and contains the warp size in threads.

- **Restrictions** - It is not allowed to take the address of any of the built-in variables and to assign values to any of the built-in variables.

2.4 Research Papers

a. **Shikha Tripathi and R.C. Jain** has introduces two novel watermarking techniques for digital images. The proposed scheme allows 5 levels of security with one optional sixth level of security. Also the DWT method provides an additional level of security as compared to the DCT based method as any of the three sub-bands can be selected for embedding the watermark. A linear relation between the watermark in frequency domain and a security matrix is proposed to embed the watermark. In case of the additive approach there is a probability of (block size) (approximately zero) that the intermediate user will be able to exactly guess the correct details. Hence, the watermarks are not susceptible to being breached for most practical purposes. The multiplicative approach provides better compression as the coefficients with zero value do not change as a result of embedding. The authorization procedure is simple and fast. It allows for easy extraction of watermark and hence, easy verification of ownership. The proposed techniques differ from other techniques, in the way watermark is spread and extracted. As the polynomial degree increases, the probability of computing the right key approaches zero for the third person indicating effectiveness of the proposed scheme. The experimental results have shown that the proposed technique will not be threatened by image cropping and JPEG lossy compression.

b. **Ali Al-Haj**,the author of the paper "**Combined DWT-DCT Digital Image Watermarking**" explained the discrete wavelet transform (DWT) and the discrete cosine transform (DCT) have been applied successfully in many in digital image watermarking. In this paper, they described a combined DWT-DCT digital image watermarking algorithm. Watermarking was done by embedding the watermark in the first and second level DWT sub-bands of the host image, followed by the application of DCT on

the selected DWT sub-bands. The combination of the two transforms improved the watermarking performance considerably when compared to the DWT-Only watermarking approach. In conclusion, in DWT-based digital watermarking applications, combining appropriate transforms with the DWT may have a positive impact on performance of the watermarking system.

c. **Seung-Hun Yoo and Jin-Hyung Park**,the author of the paper "**Accelerating Multi-scale Image Fusion Algorithms using CUDA**",shows approaches to accelerate multi-scale image fusion speed on GPU (Graphics Processing Unit) using CUDA (Compute Unified Device Architecture). The GPU has evolved into a very powerful and flexible streaming processor, which provides a good computational power and memory bandwidth. The enormous computational power and flexible programmability of GPU are applied in many existing applications requiring fast speed. The main contribution of this paper is accelerating fusion speed by applying strengths of GPU to existing multi-scale image fusion algorithms. Multi-scale image fusion algorithms were implemented using CUDA software platform of the latest version of GPU, respectively.According to fusion method, GPU was 6-to-8 times faster than CPU.

d. **Ram Rakesh and Vaclav Simek** presents in their paper the details about the acceleration of 2D wavelet-based medical data (image) compression on MATLAB with CUDA. MATLAB is a high-level language and interactive computation environment that enables the user to perform computationally intensive tasks. This paper presents the speed-up achieved and benefits of the CUDA accelerated version of 2D Image Compression based on Wavelets.The Combination of MATLAB and CUDA together with GeForce8 graphics card solves the day to day increasing demand for the massive parallel general purpose computing. As the number of processing elements is increasing in the GeForce 8 series, there is a significant potential for the compression speed-up of the high dimensional medical data images (3D, 4D etc) in the future. It took about 524288 clock cycles to compute wavelet transform within MATLAB environment on the specified CPU. The accelerated version on GPU was astonishingly

fast, because it took only 20480 clock cycles.

e. **Erkan YAVUZ and Ziya TELATAR** has proposed an improved watermarking method against SVD based watermark ambiguity is proposed, tested and shown that the ambiguity problem is resolved. DWT subbands are selected for embedding the watermark to have better robustness. The system is robust for some attacks especially for 10requires synchronization between the cover and watermarked image to get U matrix correctly, we cannot make use of some features of SVD based methods such as limited robustness to cropping and rotation. However, the similar SVD based methods being robust to cropping and rotation have watermark ambiguity problem. Then, there is no degradation with this method .Gain factor for U matrix is optimized for robustness and perceptual quality; the visual artifacts are generated if it is increased further while robustness gets worse if it is decreased. Increasing the gain factors of the SVs does not degrade image quality, but knowing the SVs are the luminance values, the image becomes brighter. They embedded the whole U matrix as a control parameter, but some part of it may be enough due to the fact that SVD image layers are arranged in descending importance, this may be a future directive.

Chapter 3

Digital Image watermarking

Digital watermarking is the process of embedding information into a digital signal in a way that is difficult to remove. The signal may be audio, pictures or video. If the signal is copied, then the information is also carried in the copy. A signal may carry several different watermarks at the same time. In visible watermarking, the information is visible in the picture or video. Typically, the information is text or a logo which identifies the owner of the media. A television broadcaster adds its logo to the corner of transmitted video, this is also a visible watermark.[17] In invisible watermarking, information is added as digital data to audio, picture or video, but it cannot be perceived as such (although it may be possible to detect that some amount of information is hidden). The watermark may be intended for widespread use and is thus made easy to retrieve or it may be a form of steganography, where a party communicates a secret message embedded in the digital signal. In either case, as in visible watermarking, the objective is to attach ownership or other descriptive information to the signal in a way that is difficult to remove. It is also possible to use hidden embedded information as a means of covert communication between individuals.[18]

3.1 Classification-by place of Application

a. Spatial domain techniques: Watermarking schemes that directly perform some transformation on the image pixels are called spatial domain watermarks or spatial water-

marks.

b. Transform domain techniques:Watermarking schemes that transform the image in the frequency domain and then modify the transform coefficients are called transform domain techniques or spectral watermarks.[17]

3.2 Types of Watermark

a. **FRAGILE WATERMARK:** Fragile watermarks do not survive lossy transformations to the original host signal and their purpose is tamper detection of the original signal. Placing the watermark information into the perceptually insignificant portions of the data guarantees imperceptibility and provides fragile marking capabilities. [17]For instance, early watermark techniques for still image data propose inserting watermark information into the least significant bits of the pixel values.

b. **SEMI-FRAGILE WATERMARK:** A semi-fragile watermark is a mark which is (highly) sensitive to a modification of the stego medium. A fragile watermarking scheme should be able to detect any change in the signal and identify where it has taken place and possibly what the signal was before modification. [18]It serves at proving the authenticity of a document.

c. **ROBUST WATERMARK:** A robust watermark should be stuck to the document it has been embedded in, in such a way that any signal transform of reasonable strength cannot remove the watermark. Hence a pirate willing to remove the watermark will not succeed unless they debase the document too much to be of commercial interest.

3.3 Applications of Watermarking

a. **Give your images the power of personalization and protection:** Using this you can add a layer of protection to your images by identifying copyright ownership and delivering a tracking capability that monitors and reports where your images are

being used. You can protect your images by tracking them beyond your own domain. [17]This will increase control over your assets by tracking and reporting on their authorized and unauthorized use.

b. **Limit unlicensed use:** Using this you can limit unlicensed use and receive information to help recover otherwise lost revenue.

c. **For digital audio and video:** Similar to the process in which artist artistically signed their paintings with a brush to claim their copyrights; artists of today can watermark their work and hide for example their name in the image. Hence, the embedded watermark will allow identifying the owner of the work. It is clear that this concept is also applicable to other media such as digital video and audio. [18]Especially the distribution of digital audio over the Internet in the MP3 format is currently a big problem. In this scenario digital watermarking may be useful to set up a controlled audio distribution and provide efficient means for copyright protection.

d. **Certification:** For example, in the field of data security, watermarks may be used for certification, authentication, and conditional access. Certification is an important issue for official documents, such as identity cards or passports in figure 3.1.

Figure 3.1: Official Documents

e. **To mutually linking information on the documents:** That means that some information is written twice on the document: for instance, the name of a passport owner is normally printed in clear text and is also hidden as an invisible watermark in the photo of the owner. If anyone would intend to counterfeit the passport by replacing the photo, it would be possible to detect the change by scanning the passport and

verifying the name hidden in the photo does not match any more the name printed on the passport.

f. **The authentication of image content:** The goal of this application is to detect alterations and modifications in an image. The three pictures below illustrate an example of this application. The picture on the left shows an original photo of a car that has been protected with a watermarking technology. In the center, the same picture is shown but with a small modification: the numbers on the license plate have been changed. The picture on the right shows the photo after running the watermark detection program on the tampered photo. [17]The tampered areas are indicated in white and we can clearly see that the detected areas correspond to the modifications applied to the original photo.

g. **Conditional access and copy-control:** For example conditional access to confidential data on CD-ROMs may be provided using digital watermarking technology. The concept consists of inserting a watermark into the CD label. In order to read and decrypt the data stored on the CD, the watermark has to be read since it contains information needed for decryption. If someone copies the CD, he will not be able to read the data in clear-text since he does not have the required watermark. The picture below shows an example of a protected CD. To read the data on the CD, the user starts a program on the CD. This program asks the user to put the CD on the scanner and then reads the watermark. If the watermark is valid the program decrypts the data on the CD and gives the user access the clear-text data. (Patent pending, contact us for license.)

h. **Copy-control:** Several companies work on a watermarking system for copy control in the DVD environment. Fully functioning solutions exist already, however, for the moment they have not been entirely approved by the content producers and providers. Finally, this solution is also an efficient and simple way to prevent the use of illegal copies of software. It has a similar functionally as the anti-piracy device called ”dongle”, but is more compact and less expensive.

i. **As invisible labels and content links:** For example, photo development laboratories may insert a watermark into the picture to link the print to its negative. This way is very simple to find the negative for a given print. All one has to do is scan the print and extracted the information about the negative. In a completely different scenario digital watermarks may be used as a geometrical reference which may be useful for programs such as optical character recognition (OCR) software. The embedded calibration watermark may improve the detection reliability of the OCR software since it allows the determination of translation, rotation, and scaling.

3.4 Uses of Digital Watermarking

The list given here is by no means complete and intends to give a perspective of the broad range of possibilities that digital watermarking opens.

a. **Image Watermarking**: Many techniques have been developed for the watermarking of still image data. For grey-level or color-image watermarking, watermark embedding techniques are designed to insert the watermark directly into the original image data, such as the luminance or color components or into some transformed version of the original data to take advantage of perceptual properties or robustness to particular signal manipulations. Requirements for image watermarking include imperceptibility, robustness to common signal processing operations, and capacity. [17],[18]Common signal processing operations which the watermark should survive include compression (such as JPEG), filtering, re-scaling, cropping, A/D and D/A conversion, geometric distortions, and additive noise. Capacity refers to the amount of information (or payload) that can be hidden in the host image and detected reliably under normal operating conditions. The watermark may be scaled appropriately to minimize noticeable distortions to the host. Some examples of watermark information include a binary sequence representing a serial number or credit card number, a logo, a picture, or a signature.

For still image watermarking, watermark embedding is applied directly to the pixel values in the spatial domain or to transform coefficients in a transform domain such as the discrete cosine transform (DCT) or discrete wavelet transform (DWT). Watermark detection usually consists of some preprocessing step (which may include removal of the original host signal if it is available for detection) followed by a correlation operator.

b. **Video Watermarking:** In this case most considerations made in previous sections hold. However, now the temporal axis can be exploited to increase the redundancy of the watermark. As in the still images case, watermarks can be created either in the spatial or in the DCT domains. In the latter, the results can be directly extrapolated to MPEG-2 sequences, although different actions must be taken for I, P and B frames. Note that perhaps the set of attacks that can be performed intentionally is not smaller but definitely more expensive than for still images.

c. **Audio Watermarking:** Again, previous considerations are valid. In this case, time and frequency masking properties of the human ear are used to conceal the watermark and make it inaudible. The greatest difficulty lies in synchronizing the watermark and the watermarked audio file, but techniques that overcome this problem have been proposed.

d. **Hardware/Software Watermarking:** This is a good paradigm that allows us to understand how almost every kind of data can be copyright protected. If one is able to find two different ways of expressing the same information, then one bit of information can be concealed, something that can be easily generalized to any number of bits. This is why it is generally said that a perfect compression scheme does not leave room for watermarking. In the hardware context, Boolean equivalences can be exploited to yield instances that use different types of gates and that can be addressed by the hidden information bits. Software can be also protected not only by finding equivalences between instructions, variable names, or memory addresses, but also by altering the order of non-critical instructions. All this can be accomplished at

compiler level.

e. **Text Watermarking:** This problem, which in fact was one of the first that was studied within the information hiding area, can be solved at two levels. At the printout level, information can be encoded in the way the text lines or words are separated (this facilitates the survival of the watermark even to photocopying). At the semantic level (necessary when raw text files are provided), equivalences between words or expressions can be used, although special care has to be taken not to destruct the possible intention of the author.

f. **Fingerprinting:** This is similar to the previous application and allows acquisition devices (such as video cameras, audio recorders, etc) to insert information about the specific device (e.g., an ID number) and date of creation. This can also be done with conventional digital signature techniques but with watermarking it becomes considerably more difficult to excise or alter the signature. Some digital cameras already include this feature.

g. **Authentication:** This is a variant of the previous application, in an area where cryptographic techniques have already made their way. However, there are two significant benefits that arise from using watermarking: first, as in the previous case, the signature becomes embedded in the message, second, it is possible to create soft authentication algorithms that offer a multi-valued perceptual closeness measure that accounts for different unintentional transformations that the data may have suffered (an example is image compression with different levels), instead of the classical yes/no answer given by cryptography-based authentication. Unfortunately, the major drawback of watermarking-based authentication is the lack of public key algorithms that force either to put secret keys in risk or to resort to trusted parties.

Chapter 4

Wavelet Transform

A wavelet is a small wave which has its energy concentrated in time. It has an oscillating wavelike characteristic but also has the ability to allow simultaneous time and frequency analysis and it is a suitable tool for transient, non-stationary or time-varying phenomena. [6]

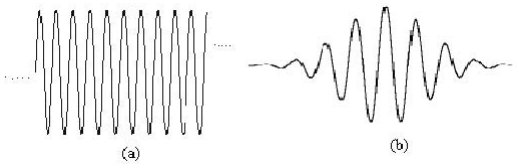

Figure 4.1: Representation of a wave (a), and a wavelet (b)

A discrete wavelet transform (DWT) is any wavelet transform for which the wavelets are discretely sampled. As with other wavelet transforms, a key advantage it has over Fourier transforms is temporal resolution means it captures both frequency and location information (location in time).

4.1 Procedure of DWT

The DWT of a signal x is calculated by passing it through a series of filters. First the samples are passed through a low pass filter with impulse response resulting in a convolu-

tion. The signal is also decomposed simultaneously using a high-pass filter shows in figure 4.2. The outputs giving the detail coefficients (from the high-pass filter) and approximation coefficients (from the low-pass).[6],[8]

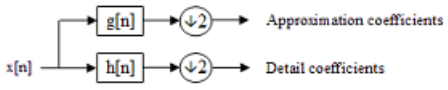

Figure 4.2: Block diagram of filter analysis

It is important that the two filters are related to each other and they are known as a quadrature mirror filter. Wavelet transform decomposes a signal into a set of basis functions.These basis functions are called wavelets. It captures not only a notion of the frequency content of the input, by examining it at different scales, but also temporal content, i.e. the times at which these frequencies occur. Combined, these two properties make the Fast wavelet transform (FWT) an alternative to the conventional Fast Fourier Transform (FFT).

4.2 The 2D-DWT transform

Wavelets are special functions which, in a form analogous to sine's and cosines in Fourier analysis, are used as basal functions for representing signals. For 2-D images, applying DWT corresponds to processing the image by 2-D filters in each dimension shows in figure 4.3. The filters divide the input image into four non-overlapping multi resolution sub-bands LL1, LH1, HL1 and HH1. The sub-band LL1 represents the coarse-scale DWT coefficients while the sub-bands LH1, HL1 and HH1 represent the fine-scale of DWT coefficients. To obtain the next coarser scale of wavelet coefficients, the sub-band LL1 is further processed until some final scale N is reached. When N is reached we will have 3N+1 sub-bands consisting of the multi-resolution sub-bands LLN and LHx, HLx and HHx where x ranges

from 1 until N, which has explained in figure 4.4.[12] Due to its excellent spatio-frequency localization properties, the DWT is very suitable to identify the areas in the host image where a watermark can be embedded effectively. In particular, this property allows the exploitation of the masking effect of the human visual system such that if a DWT coefficient is modified, only the region corresponding to that coefficient will be modified.

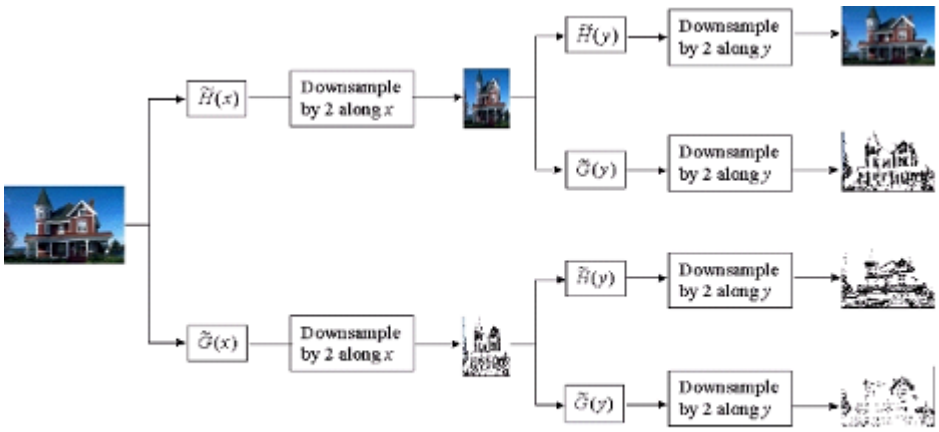

Figure 4.3: DWT function

In general most of the image energy is concentrated at the lower frequency sub-bands LLx and therefore embedding watermarks in these sub-bands may degrade the image significantly. Embedding in the low frequency sub-bands, however, could increase robustness significantly. On the other hand, the high frequency sub-bands HHx include the edges and textures of the image and the human eye is not generally sensitive to changes in such sub-bands. This allows the watermark to be embedded without being perceived by the human eye. The compromise adopted by many DWT-based watermarking algorithm, is to embed the watermark in the middle frequency sub-bands LHx and HLx where acceptable performance of imperceptibility and robustness could be achieved.

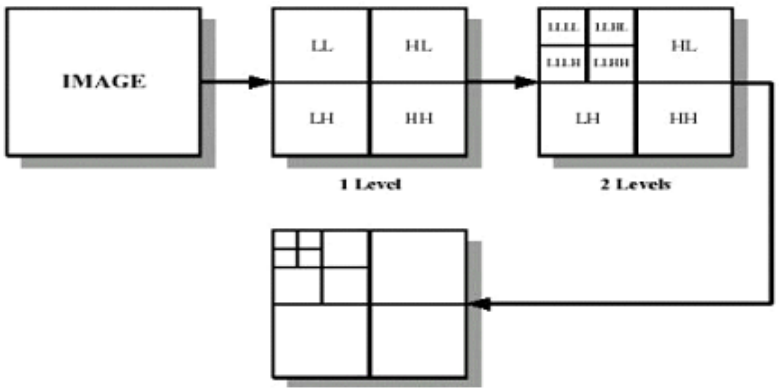

Figure 4.4: 2-D DWT for Image

4.3 Applications of Wavelet Transforms

Finally, applications of widely used standard DWT implementations, utilizing its Multi scale and Multi resolution capabilities with fast filter bank algorithms are numerous to describe. Depending upon the application, extensions of standard DWT namely WP and SWT are also employed for improved performance at the cost of higher redundancy and computational complexity.[17] A few of such applications in Data compression, De noising, Source and channel coding, Biomedical, Non-destructive evaluation, Numerical solutions of PDE, Study of distant universe, Zero-crossings, Fractals, Turbulence, and Finance etc. are comprehensively covered in. Wavelet applications in many diverse fields such as Physics, Medicine and biology, Computer Graphics, Communications and multimedia etc. can be found in various books on wavelets.

4.4 Watermarking in the Wavelet Domain

Presently, the most advanced choice among all the frequency domain methods is probably the DWT. The DWT is a hierarchical transform (unlike the FFT and the DCT), which offers

the possibility of analyzing a signal at different resolutions or levels. Such multi resolution analysis gives a frequency domain representation as a function of time; i.e., both time/space and frequency localization exists. In order to achieve this, the analyzing functions must be localized in time.

The multi resolution property makes the proposed watermarking scheme robust to image/video down sampling operation by a power of two in either space or time.A major advantage of the DWT lies in the fact that it performs an analysis similar to that of the HVS. The HVS splits an image into several frequency bands and processes each band independently.[18]

Chapter 5

MATLAB-CUDA Integration

In order to interface CUDA and MATLAB, we had to slightly modify the MEX infrastructure. CUDA files have a .cu suffix and needs to be compiled with a specific compiler (nvcc). Once we had a working infrastructure under Windows and Linux, the coding of the MEX functions is very simple. The MATLAB code is running in double precision and the data is transformed to single precision before it is transferred to the GPU. The computation on the GPU is performed in single precision and the result is transformed back to double precision before it is returned to MATLAB.[16],[13]

The advantages of using MATLAB are as follows:

- Ease of Use
- Platform Independence
- Predefined Functions
- Device-Independent Plotting
- Graphical User Interface (GUI)
- MATLAB Compiler

To retain these benefit and moreover to improve the performance, MATLAB may be integrated with CUDA, whereby a drastic improvement in the computing performance may

obtained.

The basic architecture of the code is described in figure 5.1.

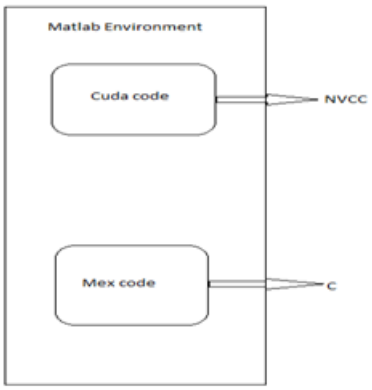

Figure 5.1: Basic Architecture

5.1 How to compile CUDA code in MATLAB

a. There are several ways to execute a CUDA code in MATLAB. The end goal of any method is to convert your ".cu" files into executable ".mexw32" or ".mexw64" files that can be called in MATLAB in the same way you would use regular ".m" files. These are the following options.[7],[8]

- Using MEX file.
- Using GPUmat.
- Using Jacket Engine

Mex files are way to built the interface between Nvidia GPU and MATLAB. For this system configuration part is the most important factor. Detail information of the setup which is depends on the specific versions will be describe. Following are the specific system configuration in which development environment is built and working nicely.

b. **System configuration**

- Os-Windows-7(32 bit)
- CL compiler of Visual Studio 2005(specifically)
- Nvcc of NVIDIA CUDA(GEFORCE-310M)
- MEX compiler of MATLAB(C compiler)
- CUDA toolkit 3.1 win 32

c. **MATLAB Setup for Building MEX**

MATLAB supports the use of a variety of compilers for building MEX-files. An option file is provided for each supported compiler. We can specify which compiler we want to use. Once it is verified that supported C, C++ or FORTRAN compiler being used, then the system is ready to configure MEX-files. In order to do this, have to run the following command from the MATLAB command prompt:

mex -setup

A series of questions are asked regarding the location of the C, C++ or Fortran compiler you would like to use to compile your code. After specifying those a MEX options file is created that gives MATLAB all of the information it needs to use your compiler during compilation.[2]

d. **Path Configuration**

Path setup is one of the most important part of these process. For every MEX file that we want to execute under MATLAB, we have to keep those in the current directory of the MATLAB along with the MEX folder which will be in the path of MATLAB.

e. **Compilation** To compile the program, from the MATLAB command prompt we need to use

Cuda_mex filename.cu

Cuda_mex script which will be generated when we compile setup_cuda_mex file shown in figure 5.2. In which MEX compiler is called, which is a perl script and generate cuda_mex script based on the system environment variables and path configuration of that specific system. To make this working Nvidia groups has developed a script nvcc which could be downloaded and place in the bin directory in the MATLAB path. As the cuda_mex code could not be compiled with simple MEX conpiler. This file will create cudamexopts.bat based on mexopts.bat according the path is specified in the system. It will Process mex.pl file for this purpose. In terns, which will create cuda_mex.pl based on mex.pl, along with that it will add current directory to the MATLAB path.[13],[16]

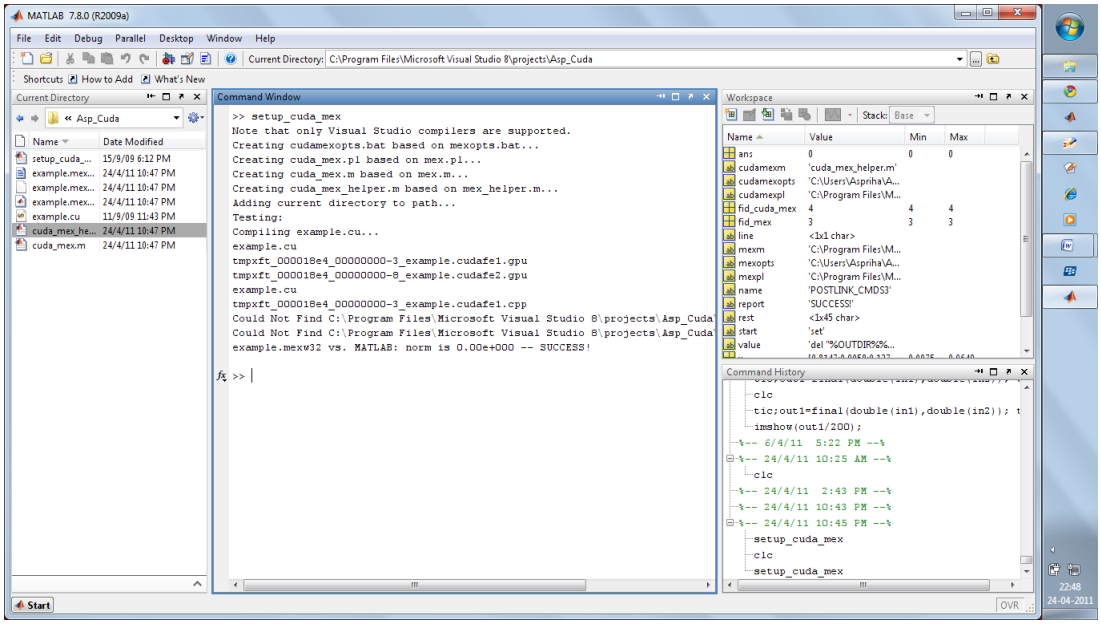

Figure 5.2: setup_cuda_mex

If the computer set up is correct (with a Cuda GPU card installed etc) we should get see ”SUCCESS” on the last line of the output. After compilation, a ”.mexw32” or ”.mexw64” file should have been created (32 bit or 64 bit respectively) which can now be executed within MATLAB similar to an ”.m” file.

5.2 Installation

5.2.1 System Requirement

To use CUDA and MATLAB on system,we will need the following configuration

a. CUDA-enabled GPU.

b. CUDA toolkit_3.1_win_32(available at free of cost from http://www.nvidia.com/cuda).

c. NVIDIA drivers

d. MATLAB 2009b and above.

e. Visual Studio 2005

5.2.2 CUDA Installation

CUDA Installation Requires:

a. The CUDA Toolkit contains the tools needed to compile and build a CUDA application in conjunction with the compilation driver. It includes tools, li- braries, header files, and other resources. The CUDA Toolkit is a C language development environment for CUDA-enabled GPUs.

b. CUDA SDK includes sample projects that provide source code and other resources for constructing CUDA programs. CUDA includes C/C++ software development tools, function libraries, and a hardware abstraction mechanism that hides the GPU hardware from developers. Although CUDA requires programmers to write special code for parallel processing, it doesn't require them to explicitly manage threads in the conventional sense, which greatly simplifies the programming model. CUDA development tools work alongside a conventional C/C++ compiler, so programmers can mix GPU code with general-purpose code for the host CPU.[9]

Chapter 6

Algorithm For Watermarking Process

6.1 EMBEDDING PROCESS

a. The original gray-scale image has to be taken as a host image and primary watermark image,each will decomposed into 2 resolution levels shown in figure 6.1.

b. A Pseudo Random Noise (PN) Sequence W_2(secondary watermark) has to be taken. And added to the horizontal coefficients (LL2) of the decomposed primary watermark image according to the equation no. 6.1.

$$W_1(i,j) = I_2(i,j) + W_2(i,j) \tag{6.1}$$

With $I_2(i,j)$ and $W_1(i,j)$ representing the DWT coefficients of the primary watermark and the watermarked primary watermark respectively.

c. The resulting $W_1(i,j)$ coefficients are then added to the horizontal,DWT coefficients of the original image according to the equation no 6.2.

$$I'(i,j) = I_1(i,j) + \alpha.W_1(i,j) \tag{6.2}$$

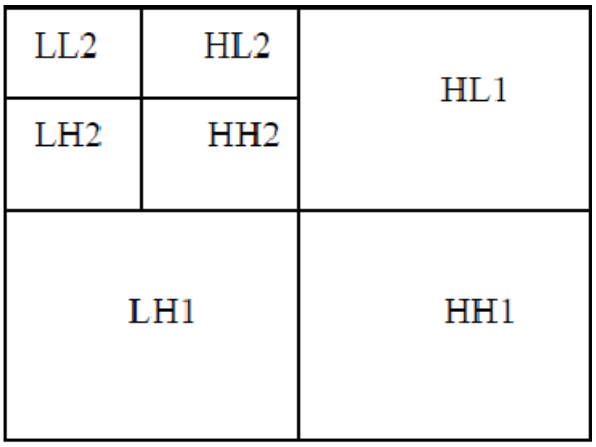

Figure 6.1: Decomposed into 2 resolution levels

In which $I_1(i,j)$ is representing the DWT coefficients of the original image, I'(i,j) is the watermarked DWT coefficients of the original image and α will be the scaling factor that is usually used to adjust the invisibility of the watermark, here it is set to 0.0001.

d. Finally, applying the IDWT to $W_1(i,j)$ and I'(i.j) we can get the watermarked primary watermark image and the watermarked host image.

6.2 EXTRACTION PROCESS

To detect the PN Sequence, a 2 level DWT is applied to the watermarked host image to obtain I'(i,j). Knowing the key α and the 2 level DWT transform of the original host image $I_1(i,j)$, the watermark of the primary watermark image DWT coefficients can be extracted using the equation no 6.3.

$$W_1(i,j) = \frac{I'(i,j) - I_1(i,j)}{\alpha} \tag{6.3}$$

6.3 Implementation of the Algorithm in MATLAB

To judge the performance of the proposed technique in MATLAB it has been extensively applied to various standard images. The technique has been tested using lena image as a host in figure 6.2 and the 64*64 MCK image as the primary watermark in figure 6.3. In this case a watermark, in form of a PN sequence (will be called the secondary watermark), is embedded in the wavelet domain of a primary watermark before being embedded in the host image. Figure 6.4 shows the watermarked lena image.

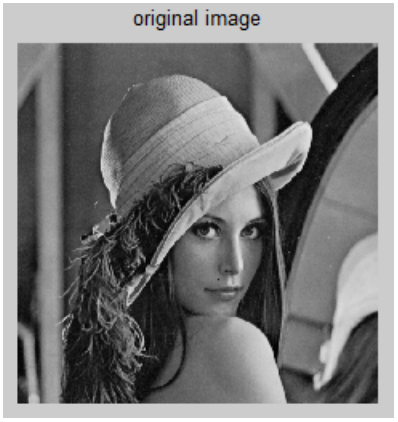

Figure 6.2: Original Image

The algorithm has been implemented in MATLAB first, to check the elapsed time taken by the algorithm in CPU using MATLAB. Finally we can compare the results with the MEX file and CUDA_MEX file,to analysis the performance.

Figure 6.3: Embedded Information

Figure 6.4: watermarked Image

6.3.1 Extraction Process

After the extraction process has completed, a 2 level of IDWT is applied to the to the extracted information to get the actual information which is embedded in to the original host image. The figure 6.5 shows the extracted information. Where 1-level IDWT is applied first and get the 32x32 image and there after 2-level of IDWT is applied to get the actual information which is embedded to the original image of size 64x64.

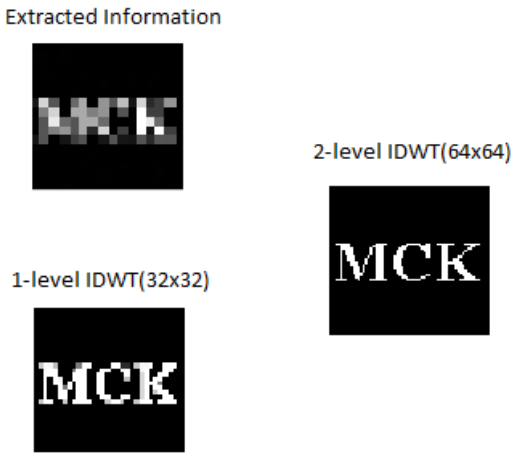

Figure 6.5: Extracted Information

6.3.2 Results

Image Size	Elapsed Time (Seconds)
512×512	1.474303
128×128	1.021206
256×256	1.042107
252×298	1.092400

Table 6.1: Results Of The M-File

Chapter 7

MATLAB Executable

We can call your own C, C++, or Fortran subroutines from the MATLAB command line as if they were built-in functions. These programs are called binary MEX-files, which are dynamically linked subroutines that the MATLAB interpreter loads and executes. MEX stands for ”MATLAB executable.” We can say,

”Dynamically linked subroutines that the MATLAB interpreter can automatically load and execute”.[5]

7.1 MEX-Files VS. M-Files

In MATLAB there are many ways to write code to perform simulations or large calculations. M-Files allow the user to write scripts containing sequences of MATLAB instructions. These scripts can be used in the same way as built-in commands, allowing the user to create new MATLAB commands.[1] Since these scripts are written with MATLAB commands they are optimized for array and matrix operations, making it easy for the user to write a new script to perform complex calculations. MEX-Files are written using C code and are then compiled using the NVIDIA MEX compiler, nvmex. The compilation of a MEX-file leads to an advantage over M-files due to the fact that the MATLAB interpreter is still needed to translate each line of the M-file while the interpreter is not used for a MEX-file. The lack of a need for an interpreter gives a performance increase to MEX-files.

This is one reason that MATLAB's built-in functions are compiled source. MEX-files have an extra overhead caused by the copying of data from MATLAB space to the MEX-file. This overhead grows as the sizes of the matrices being copied grows. The main advantages of using MEX-files over plain m-files are the fact that loops, and other operations that MATLAB struggles with should be much faster when written in a compiled language than in native MATLAB, and that existing programs can easily be ported to run from within MATLAB.[2]

The best way of thinking about MEX-files, is to consider them as an interface, with MATLAB on one side, and your program on the other. All you need to do is write a gateway function, to pass data between the two, and you are done. By using the MEX-file as a gateway, and keeping your MEX code and functional code separate, it will be much easier to reuse your code later, or to build a normal program as a front end or for debugging.

7.1.1 The Gateway Function

As mentioned above, the gateway function controls the passage of data between MATLAB, and your external program. In many ways, it's like the main function of a C program, since it is the first function to be called when invoking a MEX-file, and it always has the same prototype, like so:

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])

As shown above, it returns void, meaning that all of the input and output variables are passed as pointers, rather than as copies. Now, consider calling a MEX-file, called exampleMex from within MATLAB:

[a, b] = exampleMex(c, d, e);

nlhs and plhs relate to the arguments on the left hand side of a call to the MEX-file from within MATLAB. nlhs gives the number of arguments, and plhs is an array of pointers to the

variables. Similarly, nrhs and prhs hold the number of arguments to the function call (the right hand side of the MATLAB expression), and an array of pointers to those variables, respectively. So, using the function call above as an example, nlhs would be two, nrhs would be three, plhs would contain pointers to the (initially empty) variables a and b, and prhs would point to the variables c, d and e, which shouldn not be empty.[2]

7.1.2 Data Types

There is only one data type in MATLAB - the mxArray. MxArrays in which all types of MATLAB variable can be stored. In some ways, this makes life easier, as it reduces the number of data types to you need to remember, but it does mean that you need to be careful, and not make any assumptions about the format of the contents of the array. Along with that there are various functions which can help to determine what is inside of an mxArray and act accordingly. To access the data within an mxArray, you need to get a pointer to it. The function mxGetPr does this, as shown below.

double *a; a = mxGetPr(phrs[0]);

To check if a given mxArray contains double precision floats, use:

if **(!mxIsDouble(prhs[0])) mexErrMsgTxt(”Input image should be double.”);**

Various other functions exist which operate on mxArrays, including mxGetN and mxGetN which get matrix dimensions, and mxCreate functions for setting up mxArrays of each MATLAB data type. Example- To create a 10-by-10 matrix of real doubles, use:

p = mxCreateDoubleMatrix(10, 10, mxREAL);

This command will initialise the memory to zero, which is unlike a call to malloc, which just allocates a chunk of heap. As an aside, if we use malloc, then consider trying calloc instead, which will zero memory for us.

7.1.3 Addressing Matrices

Data within mxArrays is stored as one blob of memory, column wise. This means that a matrix is addressed as shown in figure 7.1. This is probably contrary to most people's expectations, and so is something to watch out for. For example, to address the element in figure 1 containing 3 (0, 1), you would need to look at the second element.[1]

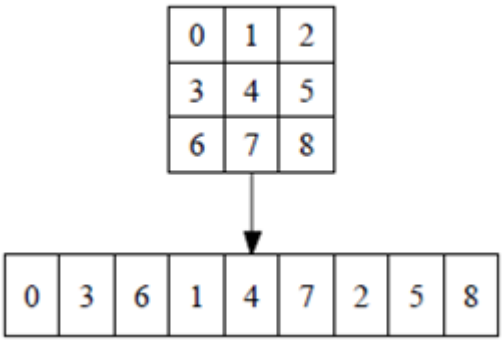

Figure 7.1: mxArray memory layout.

To convert from x and y coordinates to linear column wise addressing, use: i = xr+y, Where r is the number of rows in the matrix.

7.1.4 Managing Input and Output Parameters

MATLAB passes data to and from MEX-files in a highly regulated way. We can also take care when using an input parameter to create output data or any data used locally in your MEX-file. This is because of the way MATLAB handles MEX-file cleanup after processing. If you want to copy an input array into your local myData array, we have to call mxDuplicateArray to make of copy of the input array.

7.1.5 Allocating and Freeing Memory

MATLAB also performs cleanup of MEX-file variables. However, MathWorks recommends that binary MEX-files destroy their own temporary arrays and free dynamically

allocated memory. It is more efficient to perform this cleanup in the source MEX-file than to rely on the automatic mechanism. MATLAB provides functions, such as mxMalloc and mxFree, to manage memory. Use these functions instead of their standard C library counterparts because they let MATLAB to manage memory and perform initialization and cleanup.

7.1.6 Displaying Messages to the User

We have to use mexPrintf function, as like as a C/C++ printf function, to print a string in the MATLAB Command Window. Along with that the mexErrMsgIdAndTxt and mexWarnMsgIdAndTxt functions to print error and warning information in the Command Window.

7.2 Example: Writing a ”Hello World” MEX-file

In this first example, we will create a MEX-file (hello.c) that prints ”hello world” to the screen. We will then build and run the MEX-file from MATLAB.

a. MEX source should start like this: #include ”mex.h”

b. Every MEX-file has the mexFunction entry point. The souce now becomes#include ”mex.h” void mexFunction(int nlhs, mxArray *plhs[],int nrhs, const mxArray *prhs[])

c. Add an API function to make the MEX-file do something. The final version of the souce becomes: #include ”mex.h” void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) mexPrintf(”Hello, Aspriha.....!”); Your first MEX-file is complete. Save it as hello.c.

d. Now you are ready to compile and link the MEX-file. You can do this with the following command: mex hello.cNotice that hello.mexw32 (the MATLAB callable MEX-file for 32-bit Windows) is created in the current directory; MEX-file extensions vary by platform.

Program 1 The Hello World! program in MEX.

```
#include "mex.h" void mexFunction(int nlhs, mxArray *plhs[], int
nrhs, const mxArray *prhs[])
 {
   mexPrintf("Hello, Aspriha......!");
 }
```

e. You can now call the MEX-file like any other MATLAB file by typing its name at the MATLAB command prompt. hello

7.3 MEX Implementation

The algorithm has been implemented in MEX environment form MATLAB, for that, results in terms of time has been measured. MATLAB is a very powerful tool which deals with immensely large image manipulations. There are several ways available to write code in order to perform simulations or large calculations from MATLAB,but they are very time consuming. MEX implementation is one of the option, which takes less time compared to M-files implementation. Comparison results with respect to time in between the M-files and MEX-files has been measured.

7.3.1 Results

Image Size	Elapsed Time (Seconds)in M-file	Elapsed Time(Seconds)in Mex-file
512×512	1.474303	0.673000
128×128	1.021206	0.282000
256×256	1.042107	0.625000
252×298	1.092400	0.704000

Table 7.1: Results Of The MEX-File

Chapter 8

Strategy to implement the code in CUDA_MEX

CUDA is an architecture developed by NVIDIA that provides an application programming interface that allows programmers to write code to run in multiple processes, or threads, on a computer's GPU. The programming model is broken down into a host and the device. The CUDA API exposes functions that allow the programmer to copy data from host memory to device memory. Once the data is in device memory and ready to be processed, kernel functions are called which process the data. Two types of kernels that can be run on the GPU are global and device kernels. Global kernels can only be called from the host while device kernels can only be called from the device. When invoking a CUDA kernel, we must specify the block dimensions which determines the number of threads that will be used to execute the code.[3],[16],[19]

On top of being able to specify the size of thread blocks, grid sizes can be set which allow multiple thread blocks to execute on the same data. Figure 8.1 shows the structure of CUDA_MEX implementation flow. In which two kernels are called and all are global kernel. Each kernel will be discussed in more detail. For this project CUDA was used in MEX-files in order to perform MATLAB calculations in numerous threads.[11]

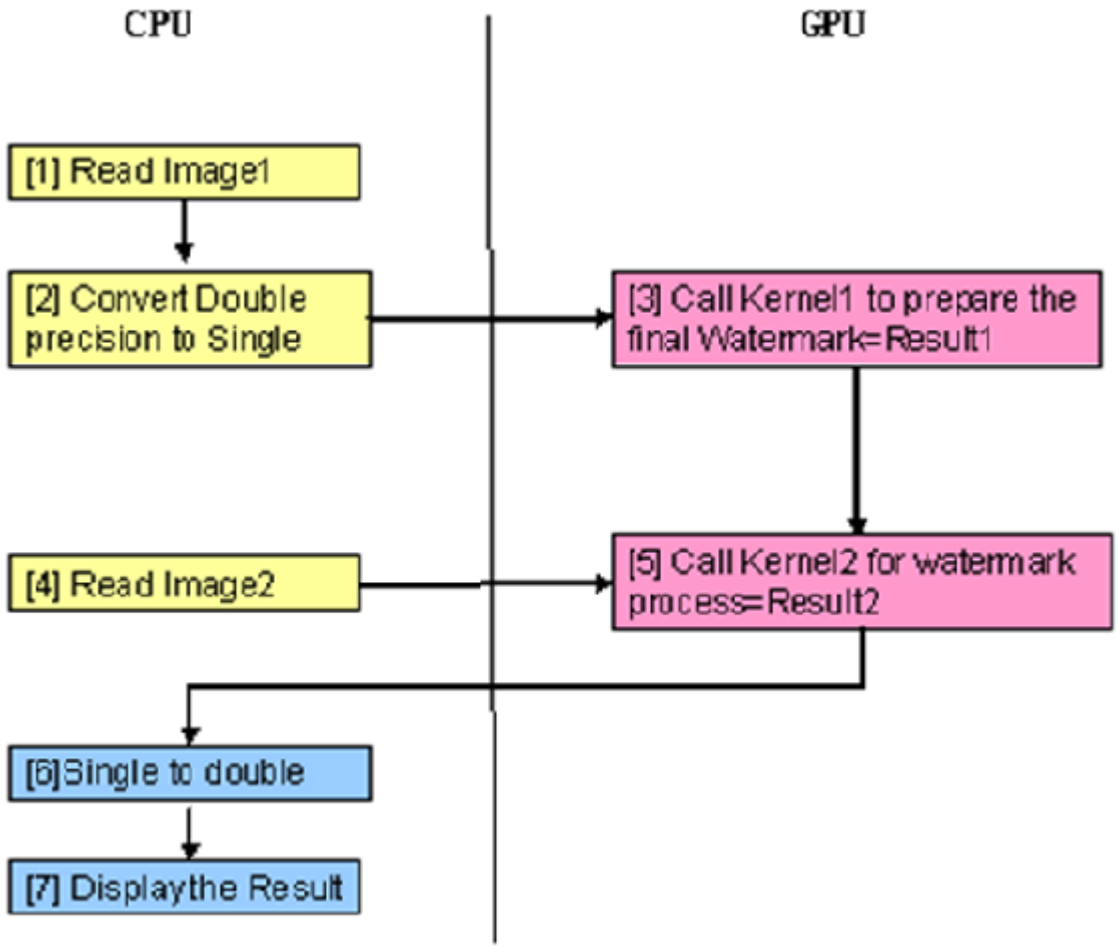

Figure 8.1: Flow of CUDA_MEX implementation

8.1 Cuda_Mex Files

Another way to exploit GPU capabilities in MATLAB is through the use of CUDA. We can use MEX files to perform computations on the GPU using CUDA. MATLAB provides a script, called mex, to compile a MEX file to a shared object that can be loaded and executed inside a MATLAB session. This script is able to parse C, C++ and FORTRAN code. This script can also call CUDA code to create greater optimization on GPUs for

Matlab computation. Even though MATLAB is calling optimized libraries internally, there is still room for further optimization. And a heterogenous type of programming is the answer.[19],[11]

8.2 CUDA-Enabled MEX Files

For most general cases, the MEX file will contain CUDA code (kernel functions, configuration) and in order to be processed by "nvcc", the CUDA compiler, the file needs to have a ".cu" suffix. This suffix .cu could not be parsed from the native MATLAB mex script. To solve this problem, NVIDIA developed new scripts that are able to parse these kind of files. The operation of the mex command is also different between Windows and Linux. In Windows, it invokes a perl script, while in Linux it invokes a shell script.[16],[11]

8.2.1 Steps to Perform:

Typical MEX files perform the following steps:

a. Convert from double to single precision.

b. Rearrange the data layout for complex data.

c. Allocate memory on GPU.

d. Transfer the data from the host to the GPU

e. Perform computation on GPU(library custom coda)

f. Transfer the data from the GPU to host

g. Rearrange the data layout for complex data

h. Convert from single to double precision.

i. Cleanup memory and return result to MATLAB

8.2.2 The basic CUDA MEX files contain the following parts

a. Memory allocation on the GPU.

b. Data movement from the host memory to the GPU. It is important to remember that the standard floating point format in MATLAB is double precision, while the current CUDA release and the underlying GPU hardware support only single precision. Before transfer to the card, if the incoming data is in double precision, the data need to be cast to single precision.

c. Currently, it is not possible to use "pinned" memory on the host, because the data on the host side needs to be allocated with the special mxMalloc function.

d. Once the data is on the GPU, it is now ready to be processed by CUDA code(custom kernel functions or CUBLAS/CUFFT functions).

e. Data movement from the GPU to host. Once again, data may need to be converted back to double precision.

f. Clean-up of the memory allocated on the GPU. [16],[11]

8.3 Implementation of CUDA_MEX

As previously mentioned, two CUDA kernels were used, and all are global kernels. A description of each of the kernels including: grid size, block size, and purpose will be presented here. The thread block size of all of these kernels is set at the maximum, 512, this allows for the largest possible matrix to be processed. The order in which they are presented is the same as the order in which they are invoked in the code. The first function is random1, the prototype for this function is shown below.

void random1();

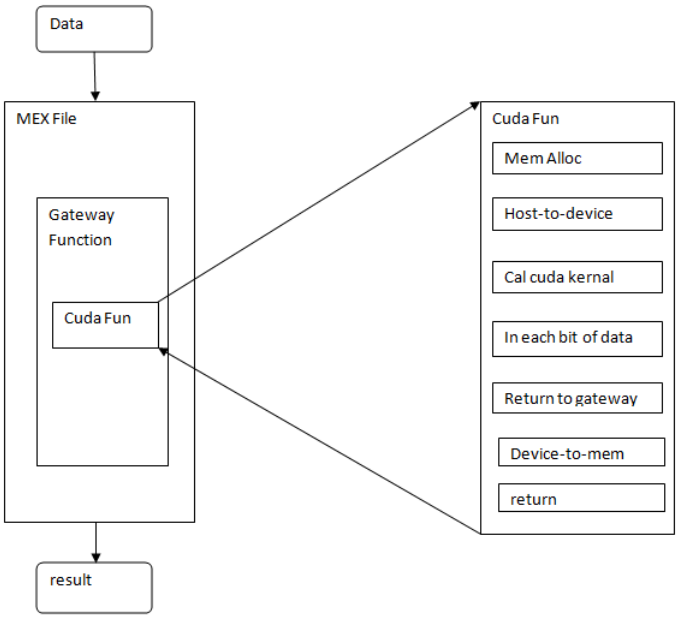

Figure 8.2: Cuda_Mex Flow

This function will create random number which will going to be used to prepare the final watermark image,which is to be embed in the original image. That is done in the next kernel.

__global void f_water();

__global void water();

The kernel,__global void water(),will produce the final watermark image.

For extraction process the kernel is **__global void extract()**.Which will extract the embedded information from the watermarked image. For DWT we have used the in-built facility of MATLAB by using the function mexCallMATLAB() provided my mex interface in MATLAB. The figure 8.3 shows the result of the embedding process. Where an information of size 64x64 is embedded to the original image of size 512x512 and get the watermarked image of size 512x512.

Figure 8.3: Information Embedding

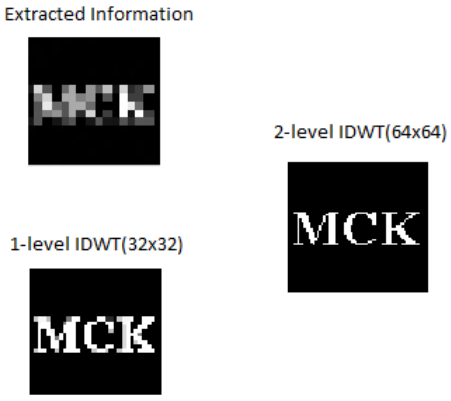

Figure 8.4: Information Extraction

The figure 8.5 shows the execution process from the MATLAB command prompt. And the figure 8.4 shows the 2-level of IDWT to get the extracted information.

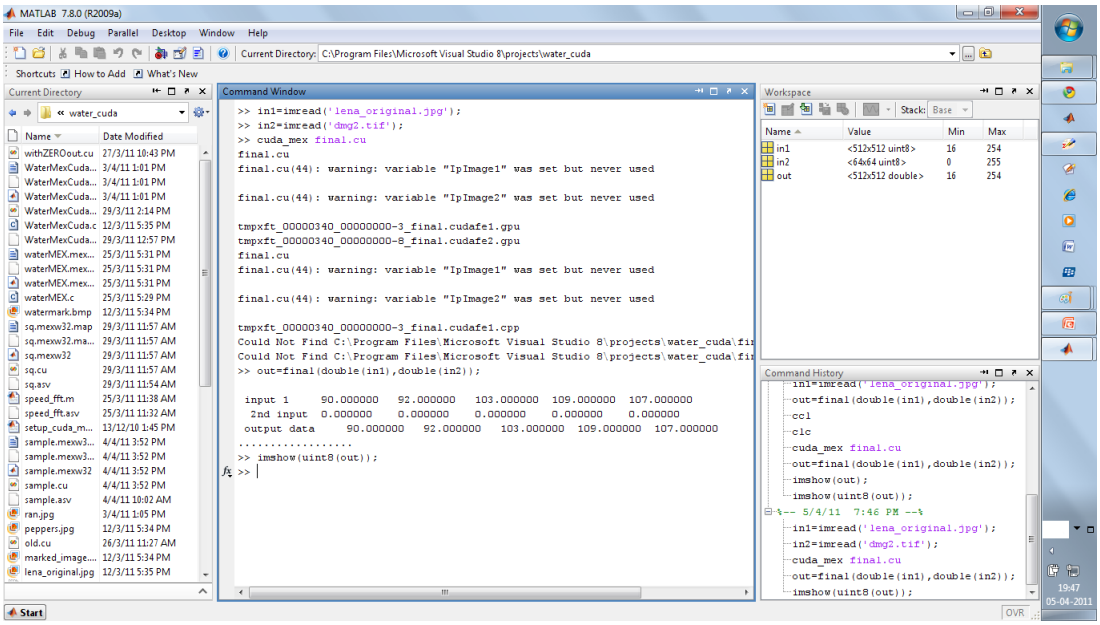

Figure 8.5: Execution

8.3.1 Experimental Results

The algorithm has been implemented now in Cuda_Mex. Experiment has been done with number of images. All the images are of gray scale with different dimension. Speedup of 18x has increased as compare to M-file and Mex-files in terms of time.

Image Size	M-file(Sec)	Mex-file(Sec)	Cuda_Mex-file(Sec)
512 × 512	1.474303	0.673000	0.05620
128 × 128	1.021206	0.282000	0.02210
256 × 256	1.042107	0.625000	0.05030
252 × 298	1.092400	0.704000	0.04220

Table 8.1: Results Of The CUDA_MEX-file

This graph shows the analysis of the results of the M-files, MEX-files and the cuda_mex files in terms of time.

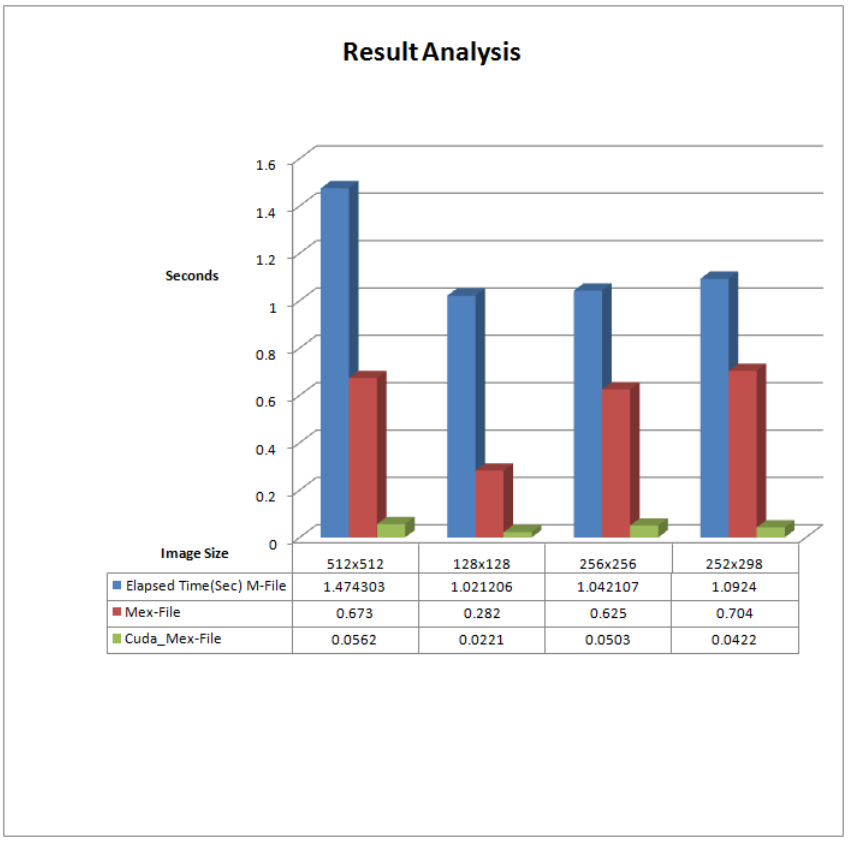

Figure 8.6: Result Analysis

Chapter 9

Conclusion and Future Work

9.1 Conclusion

With the current development of multiprocessor systems the strive for computing data on such processor have also increased exponentially. If the multi core processors are not fully utilized, then even though we have the computing power, the speed is not available to the end users for their respective applications. In accordance to this, the users or application designers also have to design newer applications taking care of the computing infrastructure available within. Our approach to this problem is CUDA_MEX implementation.

With this experiment we can say that, combination of MATLAB and CUDA enables high-productivity and high-performance solutions. With GeForce 8 series GPUs are now available even in laptops, which will be a very effective tool for engineers and scientists. This process could be used inside the MATLAB as a tool. For growing demand of digital data processing, authentication is most important and it should be fast enough to deal with the current need. Thus, programmers can have a higher performance during watermarking process.

9.2 Future Work

There is still place for further optimization with the help of shared memory concept of CUDA. Similarly, such tools may also be created for audio and video watermarking, as lot more data processing is involved, greater scope for data parallelization is possible by reducing the time. Along with this, some other future enhancement are as follows:

- The watermark can be scrambled through a well-known PN-sequence. Scrambling the logo image enhances the system security and provides a random distribution of original data.
- Error correcting codes can be used for better results.
- A hybrid approach can be applied by embedding image watermark, which includes both DWT and DCT.

Web References

[1] www.nvidia.com/cuda.

[2] NVIDIA_GPU_Computing_Webinars_CUDA_Introduction_ April-2009

[3] http://en.wikipedia.org/wiki/DWT

[4] http://www.nvidia.com/object/teslacomputingsolutions. html

[5] http://gpgpu.org/oldsite/data/history.shtml

[6] http://people.maths.ox.ac.uk/~gilesm/hpc/NVIDIA/NVIDIA_ CUDA_Tutorial_No_NDA_Apr08.pdf

[7] http://gp-you.org/.

[8] http://www.accelereyes.com/.

[9] http://developer.download.nvidia.com/compute/cuda/2_2/ toolkit/docs/CUDA_Getting_Started_2.2_windows.pdf

[10] http://forums.nvidia.com/index.php?showtopic=181472

[11] http://developer.download.nvidia.com/compute/cuda/3_1/ toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf

[12] http://code.google.com/p/gpumcml/source/browse/trunk/Fermi/NVIDIA_CUDA_BestPracticesGuide.pdf

[13] (2009) Thread block structure. [Online]. Available: http://www.cs.utexas.edu/users/fussell/courses/cs384g/projects/final/artifacts/mjeong/

References

[1] An introduction to MATLAB MEX-files, Maria Axelsson, Centre for Image Analysis, 2007-10-22

[2] By Mathworks,*"Writing MATLAB C/MEX Code, Pascal Getreuer, April 2010",* MATLAB intregration.

[3] Tze-Yun Sung, Yaw-Shih Shieh, Chun-Wang Yu, Hsi-Chin Hsin. *"High-Efficiency and Low-Power Architectures for 2-D DCT and IDCT Based on CORDIC Rotation",* Proceedings of the 7th ICPDC, pp. 191-196, 2006.

[4] "Novel DCT and DWT based Watermarking Techniques for Digital Images",The 18th International Conference on Pattern Recognition-2006,IEEE.

[5] By Mathworks, *"MATLAB External interfaces",* 2010

[6] Huang Daren,*"A DWT-BASED IMAGE WATERMARKING ALGORITHM",*2001 IEEE International Conference on Multimedia.

[7] David Kirk/NVIDIA and Wen-mei Hwu,*"CUDA Programming Model",* 2006-2008

[8] R. Kresch and N. Merhav, *"Fast DCT domain filtering using the DCT and the DST",* HPL Technical Report HPL-95-140, December 1995.

[9] Hyung Cook Kim and Edward J. Delp, *"A Comparison of Fixed point 2D 9X7 Discrete Wavelet Transform Implementations",* IEEE ICIP, vol 3,page 7803-7622, 2002

[10] "Improved SVD-DWT Based Digital Image Watermarking Against Watermark Ambiguity",2007, Seoul, Korea.2007 ACM

[11] Seung-Hun Yoo, *"Accelerating Multi-scale Image Fusion Algorithms using CUDA",*Dept. Electronics Engineering. 2009 International Conference of Soft Computing and Pattern Recognition.

[12] J. M. Shapiro. *"Embedded Image Coding Using Zerotrees of Wavelets Coefficients",* IEEE Transactions on Signal Processing, 41(12):34453462, December 1993.

[13] Ing. Vaclav Simek, *"GPU Acceleration of 2D-DWT Image Compression in MATLAB with CUDA",* Computer Architecture Group, Department of Electronics and Systems, Second UKSIM European Symposium on Computer Modelling and Simulation, 978-0-7695-3325-4/08,2008 IEEE-10.1109/EMS.2008.43

[14] M. J. Flynn, *"Some Computer Organizations and Their Effectiveness",* IEEE Transactions on Computers, C-21 (1972), pp. 948-960.

[15] D. L. N. Research, *"nvidia gpu architecture & implications",* NVIDIA Corporation 2007.

[16] By Mathworks ,*"Accelerating MATLAB with CUDA Using MEX Files",*September 11,2007 WP-03495-001_v01

[17] R. C. Gonzalez and R. E. Woods,*"Digital Image Processing",* Prentice Hall, 2001. ISBN 0-13-094650-8.

[18] Proakis and Manolakis.*"Digital signal processing Principles, Algorithms, and applications",* 2nd edition Maxwell- Macmillan pub.

[19] Halfhill, T.R., 2008, *"Parallel Processing With CUDA",* Microprocessor Report [Online] Available from: http://www.MPRonline.com

[20] University of Iceland, Reykjavik,*"The GPU on the 2D Wavelet Transform. Survey and Contributions",* Para 2010 State of the Art in Scientific and Parallel Computing extended abstract no. 26 , June 69 2010.

[21] "Accelerating Wavelet-Based Video Coding on Graphics Hardware using CUDA",Proceedings of the 6th International Symposium on Image and Signal Processing and Analysis (2009)

Index