ANKARA YILDIRIM BEYAZIT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES



PERFORMANCE COMPARISON OF IMAGE MATCHING ALGORITHM USING FPGA AND GPU

M.Sc. Thesis by

İrfan Alp GÜRKAYNAK

Department of Electronic and Communication Engineering

January, 2017

ANKARA

PERFORMANCE COMPARISON OF IMAGE MATCHING ALGORITHM USING FPGA AND GPU

A Thesis Submitted to

The Graduate School of Natural and Applied Sciences of

Ankara Yıldırım Beyazıt University

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Electronic and Communication Engineering, Department of Electronic and Communication Engineering

by

İrfan Alp GÜRKAYNAK

January, 2017

ANKARA

M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"PERFORMANCE COMPARISON OF IMAGE MATCHING ALGORITHM USING FPGA AND GPU"** completed by **İRFAN ALP GÜRKAYNAK** under the supervision of **ASSIST. PROF. DR. ENVER ÇAVUŞ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Enver ÇAVUŞ

Supervisor

Assist. Prof. Dr. Baha ŞEN

Jury member

Assist. Prof. Dr. Mehmet Efe ÖZBEK

Jury Member

Prof. Dr. Fatih V. ÇELEBİ

Director

Graduate School of Natural and Applied Sciences

ETHICAL DECLARATION

I hereby declare that, in this thesis which has been prepared in accordance with the Thesis Writing Manual of Graduate School of Natural and Applied Sciences,

- All data, information and documents are obtained in the framework of academic and ethical rules,
- All information, documents and assessments are presented in accordance with scientific ethics and morals,
- All the materials that have been utilized are fully cited and referenced,
- No change has been made on the utilized materials,
- All the works presented are original,

and in any contrary case of above statements I accept to renounce all my legal rights.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my supervisor, Assist. Prof. Dr. Enver ÇAVUŞ, for his tremendous support and motivation during my study. His immense knowledge and precious recommendations constituted the milestones of this study. His guidance assisted me in all the time of my research and during writing of this thesis.

I would like to thank Assist. Prof. Dr. Baha ŞEN for his understanding and endless support. He always motivated me during my hard times and his advices helped me a lot to overcome problems.

I also would like thank Assist. Prof. Dr. Mehmet Efe ÖZBEK for his valuable contributions and constructive criticisms during my thesis defense examination.

I am grateful to my friends Mustafa ÖZTÜRK and Erşen BALCISOY for their superior contribution and great effort while preparing the computer program for my thesis. Also my dearest friends Erkam GÜNDÜZ and Özgün ERSOY provided me enormous support during my study and always trusted on me. I am very pleasant to have such friends.

Finally, I must express my profound appreciations to my family for providing me endless support and continuous encouragement throughout my years of study and through the period of writing this thesis. This accomplishment would not have been possible without them. Thank You.

2017, 31 January **İrfan Alp GÜRKAYNAK**

PERFORMANCE COMPARISON OF IMAGE MATCHING ALGORITHM USING FPGA AND GPU

ABSTRACT

As the internet and technology developing very rapidly, the need for fast data processing is becoming more apparent. Even though the software computation capacity of computers are increasing day by day, their performance in large database operations becomes increasingly inadequate. For this reason, Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs) with parallel processing capability are frequently used to accelerate the intensive data processing operations.

The most effective method for identifying fingerprint images is the template matching. This method is used to match small parts of the image that match the template image or any other data in the entire image.

In this thesis, it is attempted to find the existing image within the database by using normalized cross-correlation (NCC) method. Computers that are used today are inadequate for large database operations, even though their computation capacity to process software is high. For this reason, Field Programmable Gate Arrays (FPGA) and Graphics Processing Unit (GPU) with parallel processing capability are frequently used to implement intensive operations.

Image database, used in matching, is constructed with 80 different fingerprint images that have 256x256 size, grayscale format and *.bmp extensions. When using the Verilog programming language to program the FPGA, the CUDA programming language was used to program the GPU. In the written codes, image pixel values are read in order and the correlation values between each compared pairs are calculated by storing them in the memory of the used devices.

The processing speed of FPGA and GPU implementations are compared in terms of their speed to calculate correlation values. Calculation of correlation values with the FPGA is 23 times faster than GPU implementation.

Keywords: Image Matching, Normalized Cross-correlation, FPGA, GPU.

GÖRÜNTÜ EŞLEME ALGORİTMASININ FPGA VE GPU KULLANILARAK PERFORMANS KARŞILAŞTIRILMASI

ÖZ

Görüntü işleme teknolojilerinin geliştirilmesi insanlığın bu teknolojiyi kullanarak görüntüleri anlamlandırmasını sağlamıştır. Dijital görüntüleri tanımlamak için kullanılan en etkili yöntem şablon eşleştirmesidir. Bu yöntem, görüntünün şablon resmiyle eşleşen küçük parçalarını veya görüntünün tamamının başka bir veriyle eşleştirilmesinde kullanılır.

Parmak izi tanıma işlemi güvenlik ve sağlık nedenlerinden dolayı günümüzde yaygın olarak başvurulan bir yöntemdir. Yöntemin temel amacı var olan bir parmak izi görüntüsünün database içerisinde bulunan diğer görüntüler arasındaki en benzerini bularak asıl görüntünün kime ait olduğunu çıkartmak için yapılır.

Bu tezde normalleştirilmiş çapraz korelasyon yöntemi kullanılarak var olan görüntünün database içerisinde bulunması amaçlanmıştır. Günümüzde kullanılan bilgisayarlar işlem yapma kapasiteleri yüksek olmasına rağmen büyük database işlemleri için yetersiz kalmaktadır. Bu sebeple paralel işlem kabiliyeti olan Alan Programlanabilir Kapı Dizileri (FPGA) ve Grafik İşlemci Üniteleri (GPU) bu işlemlerin gerçekleştirilmesinde sıklıkla kullanılmaktadır.

Eşleştirmede kullanılan imgeler 256x256 boyutunda, grayscale formatında ve "*.bmp" uzantılı 80 adet parmak izi imgeleri arasından seçilmiştir. FPGA programlamak için Verilog, GPU programlamak için CUDA programlama dili kullanılmıştır. Yazılan kodlarda image pixel değerleri sırayla okunmuş ve kullanılan cihazların hafızasına kaydedilerek korelasyon değerleri hesaplanmıştır.

FPGA ve GPU arasındaki performans karşılaştırmasının sonucunda FPGA uygulamasının GPU uygulamasına göre 23 kat daha hızlı çalıştığı görülmüştür.

Anahtar Kelimeler: Görüntü eşleme, Normalleştirlmiş Çapraz-korelasyon, Alan Programlanabilir Kapı Dizileri (FPGA), GPU.

CONTENTS

M.Sc. THESIS EXAMINATION RESULT FORM ii
ETHICAL DECLARATION iii
ACKNOWLEDGMENTS iv
ABSTRACT v
ÖZET vi
CONTENTS vii
NOMENCLATURE ix
LIST OF TABLES x
LIST OF FIGURES xi

CHAPTER 1 - INTRODUCTION 1
1.1. A General View of Image Matching Algorithms 1
1.2. Contributions 1
1.3. Thesis Organization 2

CHAPTER 2 - LITERATURE REVIEW 3
2.1. History and Literature 3
2.2. Template Matching 4
2.3. Normalized Cross-Correlation 5

CHAPTER 3 - GRAPHIC PROCESSOR UNIT (GPU) 8
3.1. GPU Architecture 9
3.2. CUDA 15
3.3. Program for NCC 15
3.4. Methods 17

CHAPTER 4 -FIELD PROGRAMMABLE GATE ARRAY (FPGA) 19
4.1. FPGA Implementation Steps 19
4.2. VERILOG 22
4.3. NCC Architecture Explanation 22

CHAPTER 5 - COMPARISON OF GPU AND FPGA HARDWARE 23
5.1. VIRTEX-7 Evaluation Board 23
5.2. GPU – GeForce GTX 960M 24

CHAPTER 6 - RESULTS **27**
6.1. Performance Comparison 28
REFERENCES **29**
CURRICULUM VITAE **32**

NOMENCLATURE

Letter Symbols

R	Reference Image
T	Reference Image

Subscripts

x	Position indices onto x-direction
y	Position indices onto y-direction

Acronyms

FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
CPU	Central Processing Unit
HDL	Hardware Description Language
NCC	Normalized Cross Correlation
SDK	Software Development Kit
FFT	Fast Fourier Transform
CUDA	Compute Unified Device Architecture
MATLAB	Matrix Laboratory
ALU	Arithmetic Logic Unit
DRAM	Dynamic Random Access Memory
SMM	Streaming Multiprocessor
LUT	Lookup Table

LIST OF TABLES

Table 5.1 Price Comparison 26
Table 6.1 Correlation Values of Four Sample Fingerprints 27
Table 6.2 Performance Measurements By Real Time 28

LIST OF FIGURES

Figure 2.1 An example of typical template matching [14] 5
Figure 2.2 Visual Explanation of Image Rotation and Translation [16] 7
Figure 3.1 Floating-Point operations per second for the CPU and GPU [18] 8
Figure 3.2 Memory bandwidth for the CPU and GPU [18] 9
Figure 3.3 CPU v.s. GPU [20] 10
Figure 3.4 Structures of CPU and GPU [18] 10
Figure 3.5 Automatic scalability property of the GPU [18] 11
Figure 3.6 An Example for kernel code block [18] 12
Figure 3.7 Grid of thread blocks 13
Figure 3.8 Maxwell Architecture [22] 14
Figure 3.9 Code Architecture for NCC 16
Figure 3.10 VecSum Kernel 17
Figure 3.11 VecSqSum Kernel 17
Figure 3.12 Kernel code block for Reduction [24] 18
Figure 4.1 FPGA Implementation Steps 19
Figure 4.2 Single Correlation Block Architecture 21
Figure 4.3 Parallelization of the Image Matching 21
Figure 5.1 VC707 Board Block Diagram [26] 24
Figure 5.2 GTX 960M features 25
Figure 5.3 GTX 960M Bandwidth and data transfer speeds 25
Figure 6.1 Fingerprint samples 27

CHAPTER 1

INTRODUCTION

The purpose of this thesis is to compare the performance results of GPU and FPGA with using Normalized Cross Correlation(NCC) and analyzing trade-offs of these programs about the template matching implementations for fingerprint image database.

1.1. A General View of Image Matching Algorithms

With the development of imaging technology, mankind has read and understood the images in its hand and started to use the results from it in practice. There are many specialized topics in image processing techniques such as object recognition, image segmentation, motion detection and medical scan analysis.

Normalized cross-correlation processing is used to calculate the similarity ratios between two-dimensional image views.

In this approach, it is intended to find the desired view within the template or within the entirety of the template using the mathematical function for NCC operation. By selecting the area from which the maximum value is obtained, it is possible to find the most similar image in the database. It is possible to determine what this view is or what belongs to it by finding the most similarity between the images that are significant on the database.

Gathering or evaluating any kind of knowledge in the globalizing world has gained so much importance. Because of the ability of parallel processing of FPGA and GPU.

1.2. Contributions

This thesis gives a multidimensional understanding of how GPU and FPGA based implementations can be compared with respect to their power, memory bandwidth, productivity, performance and elapsed time while executing NCC algorithms.

Theoretical performance calculated by specifications of using device datasheets can be enough for comparing device capability and peak performances. However, these values are not taken into account memory interfaces and latencies, place and route constraints and the others of an actual FPGA design stages. Maintained execution under a workload in any case, however, specify how much efficient that device performs in real-time applications.

Main contribution of this thesis to the literature is comparing accelerator device performances of Virtex-7 FPGA developer kit and GeForce GTX 960M. Both of them execute same algorithm for Normalized Cross-correlation function on 2D fingerprint images.

1.3. Thesis Organization

The following chapter of this thesis examines the literature for implementations of NCC algorithms and comparison of their performance with respect to FPGA and GPU. Chapter 3 gives detailed information about to implementing steps of NCC algorithm and which methods are used for to get minimum elapsed time. Next chapter (Chapter 4) explains FPGA architecture in detail. Chapter 5 gives a quantitive and qualitive discussion of the results for each device. At last, Chapter 6 will summarize the work, expressing key points and constraints during implementations.

CHAPTER 2

LITERATURE REVIEW

2.1. History and Literature

The NVIDIA company released the CUDA SDK in February 2007 public usage. In 2007, A researcher named Victor Podlozhnyuk explained basic convolution and FFT(Fast Fourier Transform) operations using cuFFT library [1] [2]. At the same time, Alexander Kharlamov and Victor Podlozhnyuk worked on a project about noise-cancelling on images with using CUDA [3]. On this project, 320x408 sample image has been used to reach 500 fps (Frame Per Second) for each image with using GeForce 8800GTX graphic card.

In 2008, Shuai Che et. al. compared throughputs an image processing algorithm programming Intel Xeon with OpenMP (Open Multi-Processing) and programming GeForce GTX 260 with CUDA(Compute Unified Device Architecture) [4].

In 2008, Michael Garland et. al. compared performance values of medical imaging algorithms that executed on both GPU and CPU [5].

In 2008, project studied by Zhiyi Yang et. al., commonly used algorithms for image processing like edge detection and histogram have been investigated and taken the results both GPU and CPU implementations [6].

In 2008, Jing Huang et. al. studied about motion detection algorithms with using CUDA C. Although there are small differences between GPUs, they find an important difference in the results [7].

In 2008, project studied by James Fung and Steve Mann, image processing techniques have been enhanced for GPU usage. Additionally, comparisons for GPU Memory Access has been mentioned [8].

In 2008, project studied by Lei Pan et. al., image segmentation for medical image samples have been implemented on GPU with using CUDA. In addition, the project includes CUDA and the previous GPU comparisons between technologies [9].

In 2008, project studied by Deepak Kumar Karna et. al., a normalized cross-correlation image matching approach has been proposed to increase correctness of detection accuracy. Error-rate has been decreased from 3 percent to 2 percent [10].

In 2009, presented study by Jun Sik-Kim et. al., KLT (Kanade-Lucas-Tomasi) algorithm which is known about operation complexity has been executed on both CPU and GPU [11].

In 2009, Michael Boyer and others detect and follow leukocyte parts of a medical image using CUDA. They achieve that operation on a video sample has been speed up 200 times according to MATLAB applications [12].

In 2009, project studied by Shuichi Asano et. al., has been compared CPU, GPU and FPGA performances in image processing applications for two-dimensional filters stereo-vision, and k-means clustering. Their analysis on performance results for three operations with three different devices shows that CPU has the worst performance while FPGA and GPU have had similar performances [13].

2.2. Template Matching

Template matching is a method to search input image within another image (template). The template matching algorithm takes an image as input and search it within another image, generally from the database. Searching can be described as an iterative comparison algorithm; the input image is moved across the image and at each step the input image and the template are compared. The areas where the correlation between the input image and the template is strong indicate more similarity between these images. The cross correlation equation is shown in Equation 2.1 [2].

$$C = \sum_{u,v} R(u,v)T(x+u,y+v) \tag{2.1}$$

Here I and T are the scene image matrix and the template image matrix, respectively. Figure 2.1. Here a small input image is searched in a bigger template image [2]

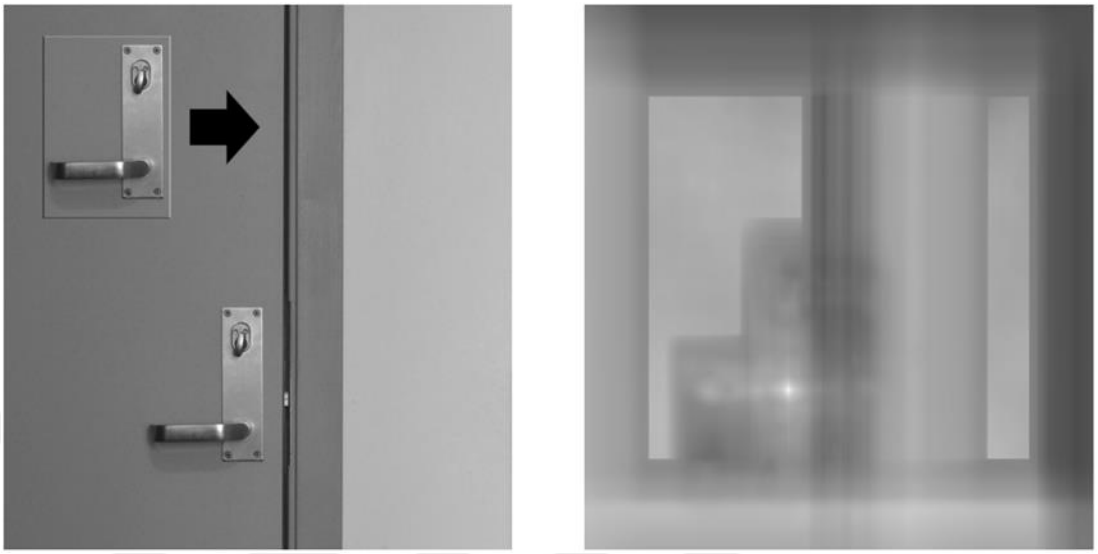


Figure 2.1 An example of typical template matching [14].

2.3. Normalized Cross-Correlation

Normalization is a fundamental step of the template matching algorithm. Two real photographs of the same object may different because of several conditions:

- Light
- camera
- background

For example, two image of the same object that taken under two different light conditions may not be pixel-wise identical. Bright parts of the image give a strong response while the image that captured under low light conditions gives a weak response. To be able to handle this problem a normalization algorithm is used after the correlation and this method called normalized cross correlation. The normalized cross-correlation function is given in Equation 2.2.

$$corr = \frac{N*\sum_{x,y}(R(x,y)*T(x,y)\,)-\sum_{x,y}(R(x,y)\,)*\sum_{x,y}(T(x,y)\,)}{\sqrt{N*\sum_{x,y}(R^2(x,y))-\sum_{x,y}(R(x,y)^2\,)}\sqrt{N*\sum_{x,y}(T^2(x,y))-\sum_{x,y}(T(x,y)^2\,)}} \quad (2.2)$$

R and *T* are standing for the image and template respectively. *T* is the arithmetic mean value of the template pixel values and R is the arithmetic mean value of the image pixel values. x and y indices represent the pixel coordinates of two-dimensional images. The value of the *corr* coefficient has the range between [–1.0, 1.0]. If value of *corr* variable equals to 1, this means perfect match between image and template [15].

Although, normalized cross-correlation offers several advantages, it has some deficiencies. Most important problem is the unreliability of the algorithm because of the pixel by pixel comparison. Therefore, the algorithm is highly sensitive to image capturing conditions beside the object itself.

The main advantage of this algorithm is its simplicity. Moreover, normalization step can help us to solve some problems about image capturing and illumination [14].

When fingerprint matching is performed, fingerprint positions in the database may be different from the original image. Therefore, correlation at one position will not be enough to find which sample belongs to whom. Rotation and shifting operations are required for implementation.

2.3.1. Rotation and Shift of a 2D Image

Rotate operation is required for matching the fingerprint image on the rotated template. To be able to do this, pixel values of the image data should be transformed into using Equation 2.3. A translation can be described as a vector addition. Rotation can be a matrix multiplication formulate. The transformation

$$x' = \begin{pmatrix}\cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha\end{pmatrix}\begin{pmatrix}x \\ y\end{pmatrix} + \begin{pmatrix}t_x \\ t_y\end{pmatrix} \quad (2.3)$$

Provides a rotation about the angle (α) counterclockwise and a displacement around the term t_x and t_y. Thus, transformation on the Equation 2.3 is sufficient to perform a

reorientation of the images. Visual explanation of the transformation operation is shown in Figure 2.2.

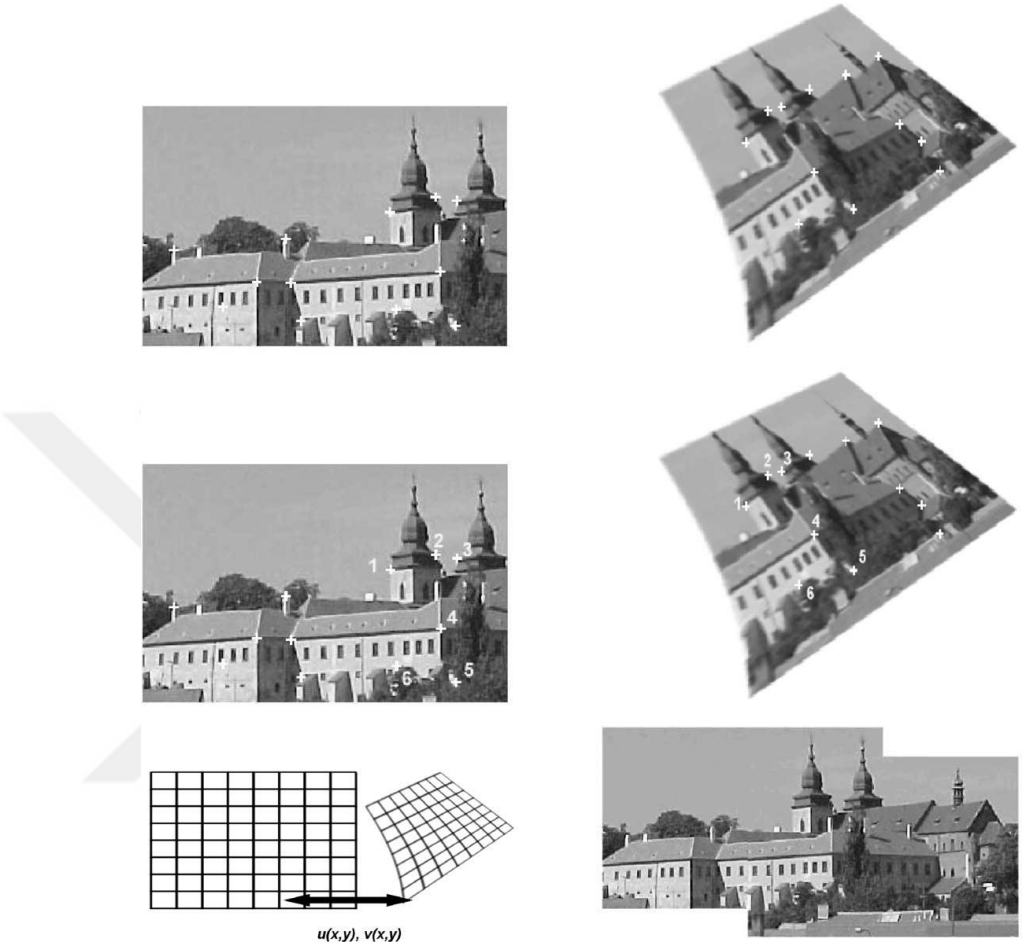


Figure 2.2 Visual explanation of image rotation and translation [16].

CHAPTER 3

GRAPHIC PROCESSOR UNIT (GPU)

For decades, one of the important methods of the performance increasing of devices have been to increase the processor's clock speed of the CPU. CPUs of the first personal computers (in early 1980s) ran with approximately 1 MHz internal clocks while, nowadays clock speed of the most desktop processors is between 1 GHz and 4 GHz. After 2005, CPUs that have multi processors (multi core) had been introduced to personal computer market to be able to increase computer performance without increase the clock speed [17].

In comparison with the CPUs, graphics processing units (GPUs) are capable to solve problems in parallel data processing structure. Floating point operation capacity increase in years for the CPU and GPU is shown in Figure 3.1. Memory bandwidth development in years for the CPU and GPU is given in Figure 3.2 [18].

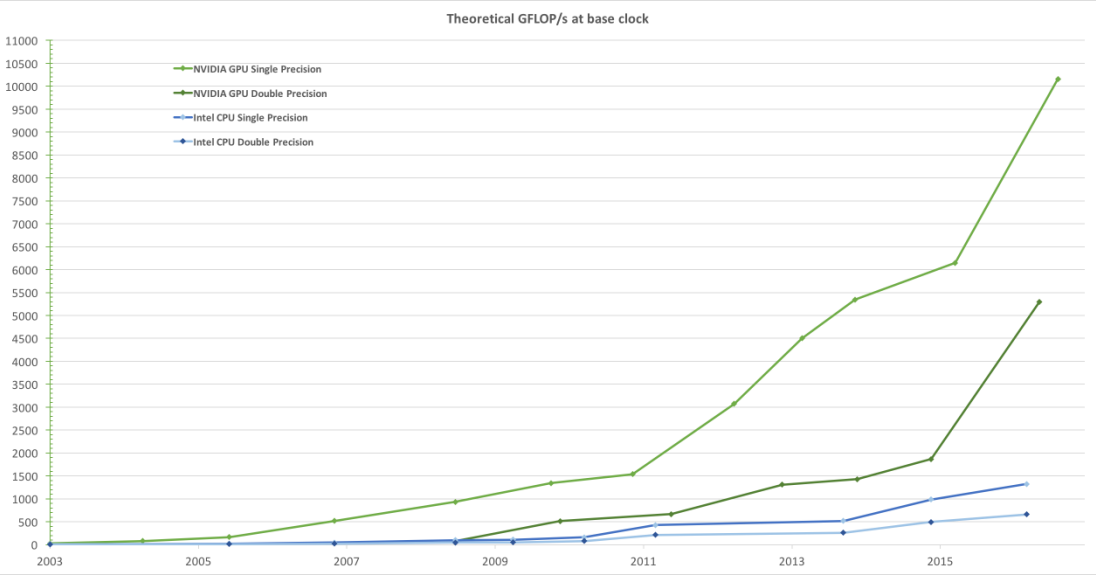


Figure 3.1 Floating-point operations per second for the CPU and GPU [18].

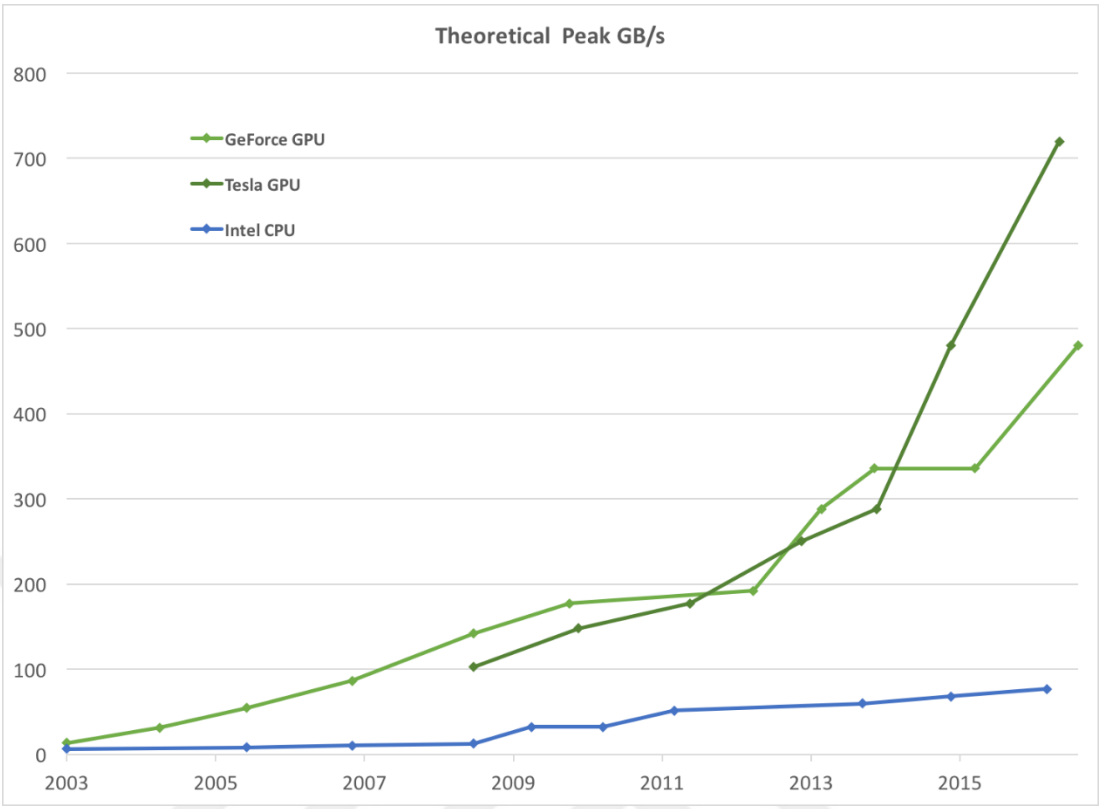


Figure 3.2 Memory bandwidth for the CPU and GPU [18].

Graphical operating systems (i.e. Windows) increase the popularity of a new type of processor and in the early 1990s. 2D display accelerators for personal computers had been started to purchase. The demand and supply of improved graphic processors had continuously increased. In 2001, NVIDIA GeForce 3, the computing industry's first chip to implement Microsoft's then-new DirectX 8.0 standard, was released. It gave some control to programmers over the exact computations that would be performed on their GPUs [19].

3.1. GPU Architecture

Basically, CPUs have a few cores and process data sequentially, however, GPUs have a massively parallel architecture which have thousands of smaller, more efficient cores designed to handle multiple tasks simultaneously [20]. Figure 3.3 shows the difference between CPU and GPU schematically.

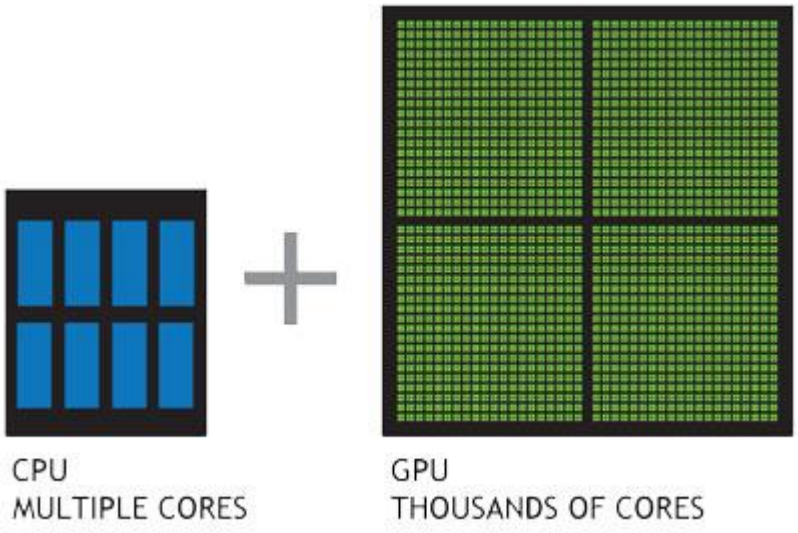


Figure 3.3 CPU vs. GPU [20].

CPUs use less transistors in calculation part compared to the GPU. This allows to reduce calculation time unlike GPU. Moreover, GPUs have numerous cores that is capable to do parallel processing. GPUs have Special Function Unit (SFU) that accelerate the calculations with transcendental functions (i.e. sin, cos, log). Figure 3.4 shows the structures of CPU and GPU [18] [21].

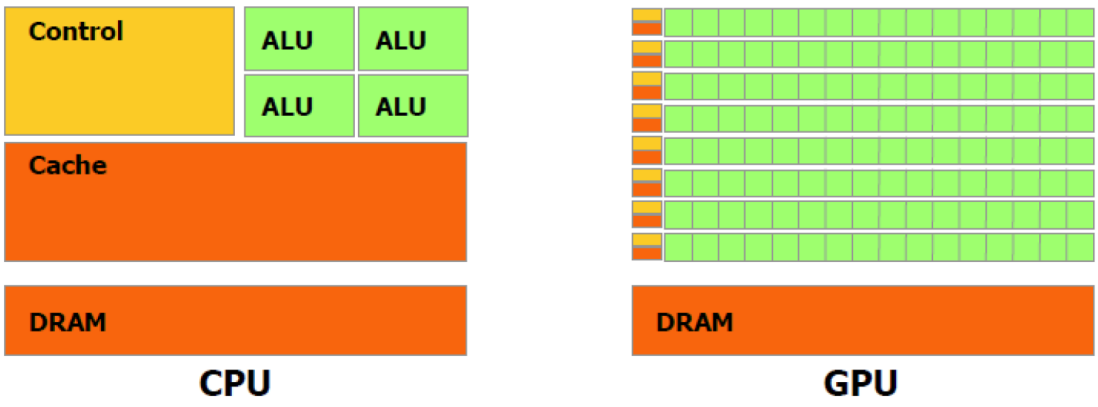


Figure 3.4 Structures of CPU and GPU [18].

Figure 3.5 shows the automatic scalability property of the GPU that arranges the duties in parallel structure with respect to the number of CUDA cores. This property allows

the scalability when different processors have different number of cores without any change in the code block [18] [21].

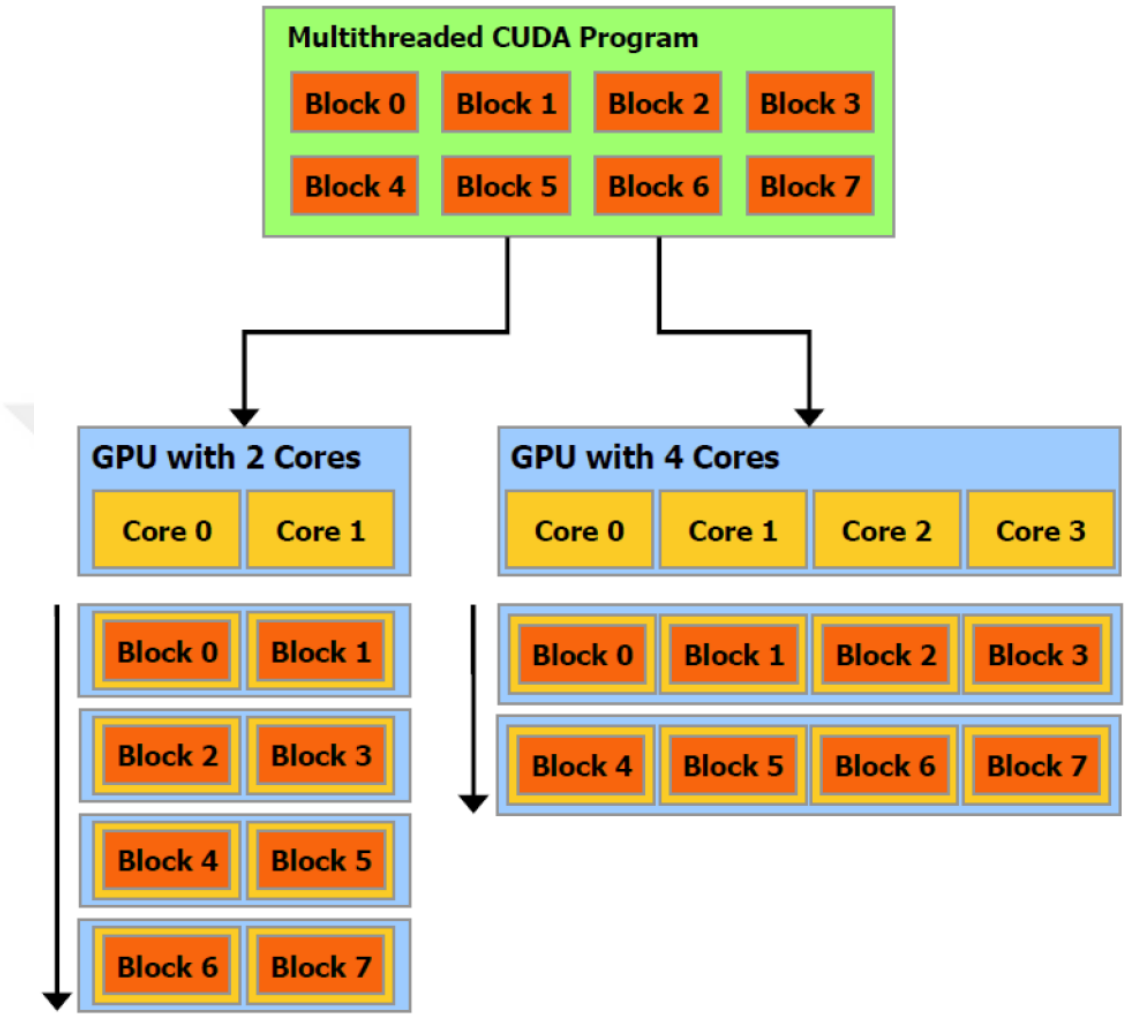


Figure 3.5 Automatic scalability property of the GPU [18].

3.1.1. Programming Model

CUDA C allows to programmers to define special C functions that is called Kernels. When a kernel is called, it is executed N times in parallel by N different CUDA rather than regular C functions. An example Kernel code block is shown in Figure 3.6. Here some extra operators can be seen that different from the standard C programming language [18].

- global_ is used to define a kernel
- <<<...>>> is execution configuration syntax that specify the number of CUDA threads that execute the kernel.
- `threadIdx`, Each tread that executes the kernel has a unique thread ID and it is defined by threadIdx. Here threadIdx is a 3-component vector, so that threads can be used as one, two or three dimensional thread [18].

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Figure 3.6 An example for kernel code block [18].

Blocks are organized as one-dimensional, two-dimensional and three-dimensional grid of thread blocks. The organization schematics for grid of thread blocks is given in Figure 3.7.

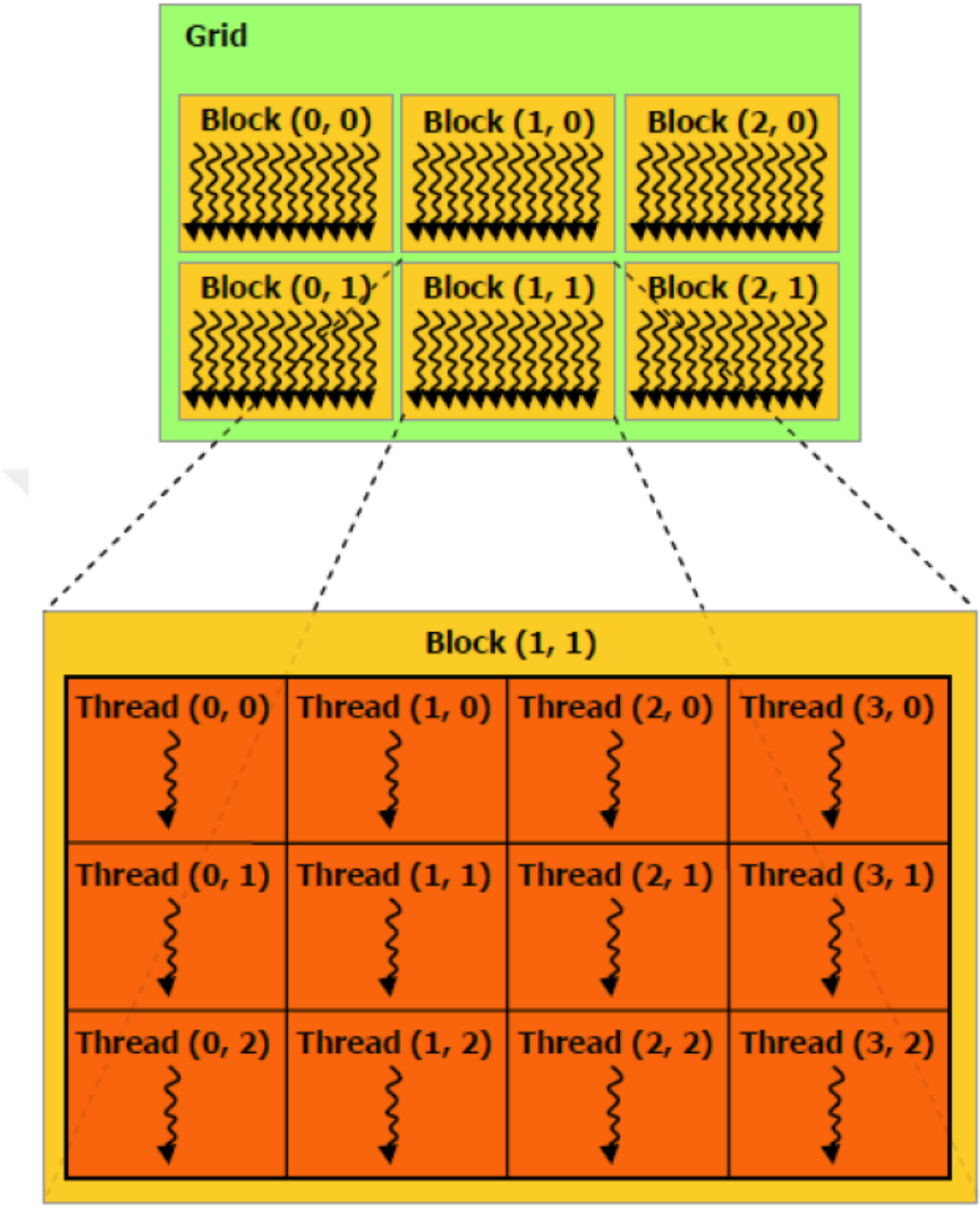


Figure 3.7 Grid of thread blocks.

CUDA threads may access to multiple memory spaces during the execution of code. Each thread has both private local memory and shared memory. The shared memory

can be visible to all threads of the block during the lifetime of the block. There is a global memory and two extra read-only memory spaces accessible by all threads.

3.1.2. MAXWELL Architecture

Streaming Multiprocessor (SMM) of the Maxwell Multiprocessor has a quadrant-based design with four 32-core processing blocks each with a dedicated warp scheduler.

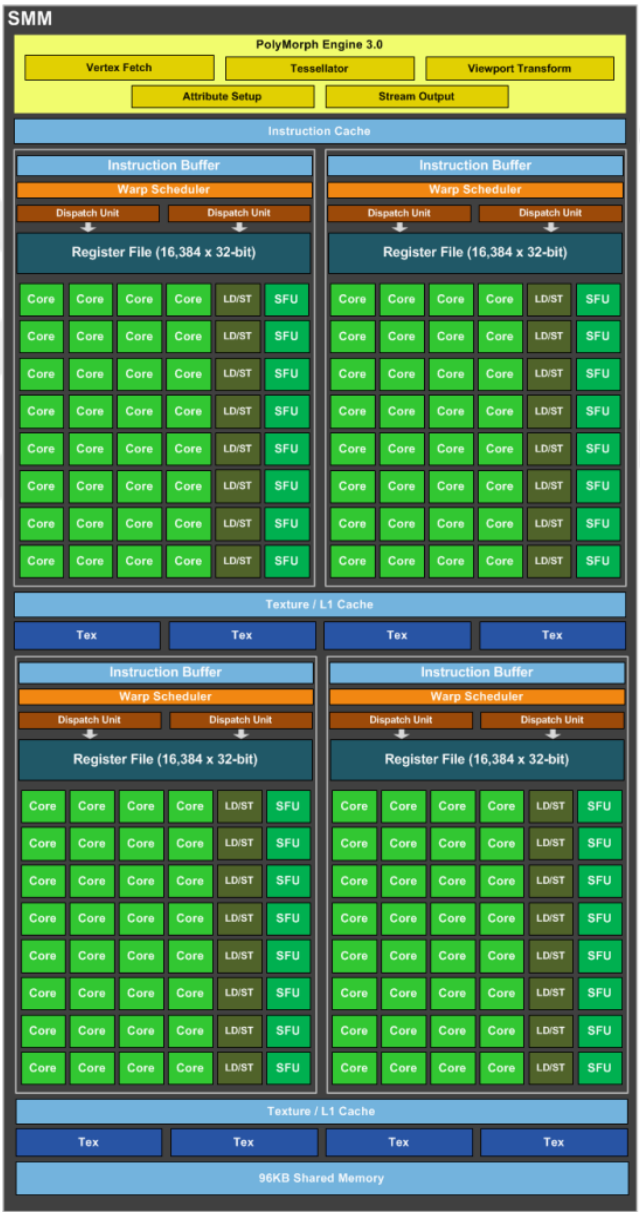


Figure 3.8 Maxwell architecture [22].

3.2. CUDA

Compute Unified Device Architecture CUDA® is a programming model and a platform for parallel computing that was invented by NVIDIA. CUDA offers high computing performance and enables to utilize the power of the graphics processing unit (GPU) [19] [23].

GPUs can be used for both graphical requirements and custom calculations with CUDA C programming language. CUDA C is modified C language that include some additional basic keywords which enable us to program to GPUs easily. CUDA C code is compiled by NVIDIA C compiler [21].

3.3. Program for NCC

The NCC function is segmented into parts to simplify calculations. Firstly, all arithmetic means of all image pixel values in the correlation equation are calculated using reduction method unrolling the last warp. The arithmetic mean is obtained by dividing the image pixel values determined by the indices of (x, y) according to the image size. In our study, all image sizes are fixed to 65536 because of size of an image is specified to 256x256. Then, the arithmetic mean values for all images calculated in GPU memory transferring to CPU memory. These values are copied to GPU memory again in order to calculate the vector that all pixel values subtracted from its mean value. The calculated values for the reference image and the test images are sent back to GPU memory in order to get correlation results.

Code architecture for image matching algorithm is shown in Figure 3.9. In this architecture 4 different kernels have been used in order to get correlation results.

VecSum kernel calculates sum of image pixels in an array. VecSqSum is used sum of squares of pixel values. The product of two mutually corresponding image pixel values for test and reference images is obtained by VecMulSum kernel. At last, CudaXCorr kernel calculates final correlation value of two interested images.

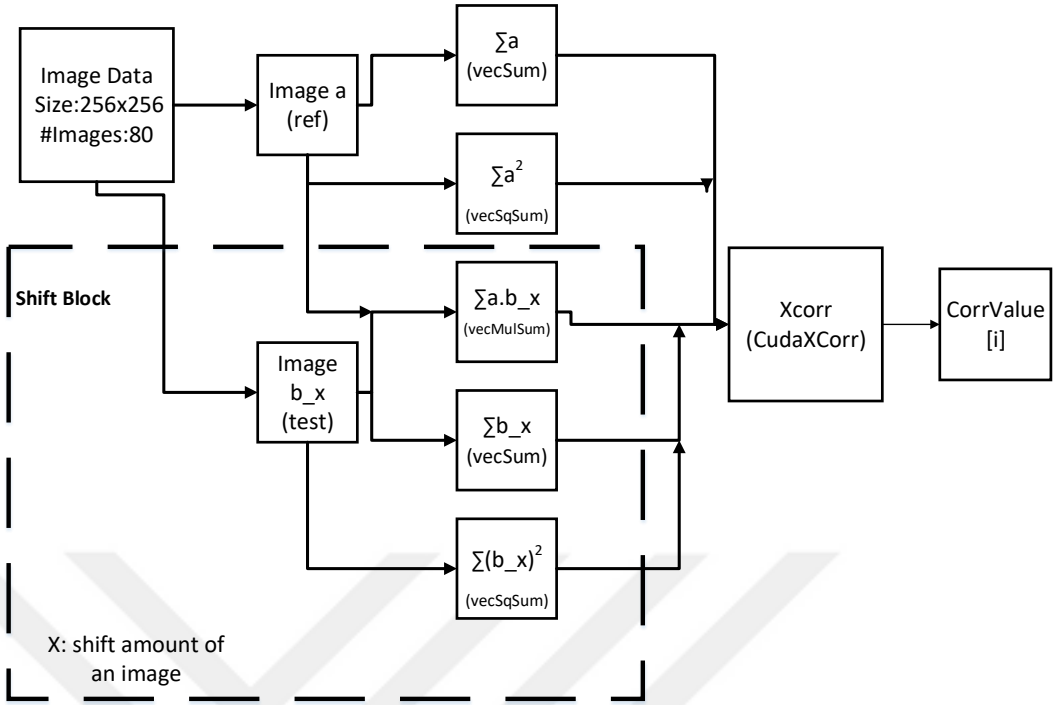


Figure 3.9 Code architecture for NCC.

Sample kernels for VecSum and VecSqSum kernels have been shown below.

In Figure 3.10, shared memory of GPU has been used because of their accessibility and speed performance. 64 blocks and 1024 threads are sufficient to operating on data that has 65536 elements. Variables such as h_shift and w_shift are moving the corresponding images according to specified direction. While h is standing for height and w is standing for width.

Similar code block for VecSqSum have been used to calculate sum of squared values in Figure 3.11. The difference between these two code blocks is image pixel values are multiplied themselves in VecSqSum.

```
__global__ void vecSum(int n, int w_shift, int h_shift, float *g_idata1, float *g_odata1)
{
    extern __shared__ volatile float s1data[];

    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(BLOCKSIZE * 2) + tid;
    unsigned int gridSize = BLOCKSIZE * 2 * gridDim.x;
    s1data[tid] = 0;

    while (i < n) {
        if (((w_shift<0) && (tid % 256 <= -1 - w_shift)) || ((w_shift>0) && (tid % 256 >= 256 - w_shift)))
            s1data[tid] = 0;
        else {
            if (i<256 * h_shift)
                s1data[tid] += g_idata1[i + BLOCKSIZE];
            else
                s1data[tid] += g_idata1[i] + g_idata1[i + BLOCKSIZE];
        }
        i += gridSize;
    } __syncthreads();
```

Figure 3.10 VecSum kernel.

```
__global__ void vecSqSum(int n, int w_shift, int h_shift, float *g_idata1, float *g_odata1)
{
    extern __shared__ volatile float s1data[];

    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(BLOCKSIZE * 2) + tid;
    unsigned int gridSize = BLOCKSIZE * 2 * gridDim.x;

    s1data[tid] = 0;

    while (i < n) {
        if (((w_shift<0) && (tid % 256 <= -1 - w_shift)) || ((w_shift>0) && (tid % 256 >= 256 - w_shift)))
            s1data[tid] = 0;
        else {
            if (i<256 * h_shift)
                s1data[tid] += g_idata1[i + BLOCKSIZE] * g_idata1[i + BLOCKSIZE];
            else
                s1data[tid] += g_idata1[i] * g_idata1[i] + g_idata1[i + BLOCKSIZE] * g_idata1[i + BLOCKSIZE];
        }
        i += gridSize;
    } __syncthreads();
```

Figure 3.11 VecSqSum kernel.

3.4. Methods

Unrolling last warp reduction method is used to calculate total sum of image pixel values. Reduction method is a core operation for the parallel execution. This method assigns all responsible threads to reach specific memory allocation.

Warp size of GeForce GTX 960M includes 32 threads. This means that 32 threads can be executed in parallel without any race condition. Each 32 threads do not wait any other threads finishing their works. Therefore, last part of the code shown in Figure 3.3 do not need any synchronization to other threads [24].

```
if (blockSize >= 512) {
    if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads();
}
if (blockSize >= 256) {
    if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads();
}
if (blockSize >= 128) {
    if (tid <  64)  { sdata[tid] += sdata[tid +  64]; } __syncthreads();
}

if (tid < 32) {
    if (blockSize >=  64) sdata[tid] += sdata[tid + 32];
    if (blockSize >=  32) sdata[tid] += sdata[tid + 16];
    if (blockSize >=  16) sdata[tid] += sdata[tid +  8];
    if (blockSize >=   8) sdata[tid] += sdata[tid +  4];
    if (blockSize >=   4) sdata[tid] += sdata[tid +  2];
    if (blockSize >=   2) sdata[tid] += sdata[tid +  1];
}
```

Figure 3.12 Kernel code block for Reduction [24].

Elapsed time for calculation of an array has 65536 elements is approximately 0.04ms.

CHAPTER 4

FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Field-Programmable Gate Array (FPGA) is an integrated circuit which can be programmed electronically in order to run any functionality. Although structures of FPGAs vary from brand to brand, an FPGA typically have interconnects and programmable logic cells [14].

4.1. FPGA Implementation Steps

The fundamental steps of configuring a FPGA is shown Figure 4.1.

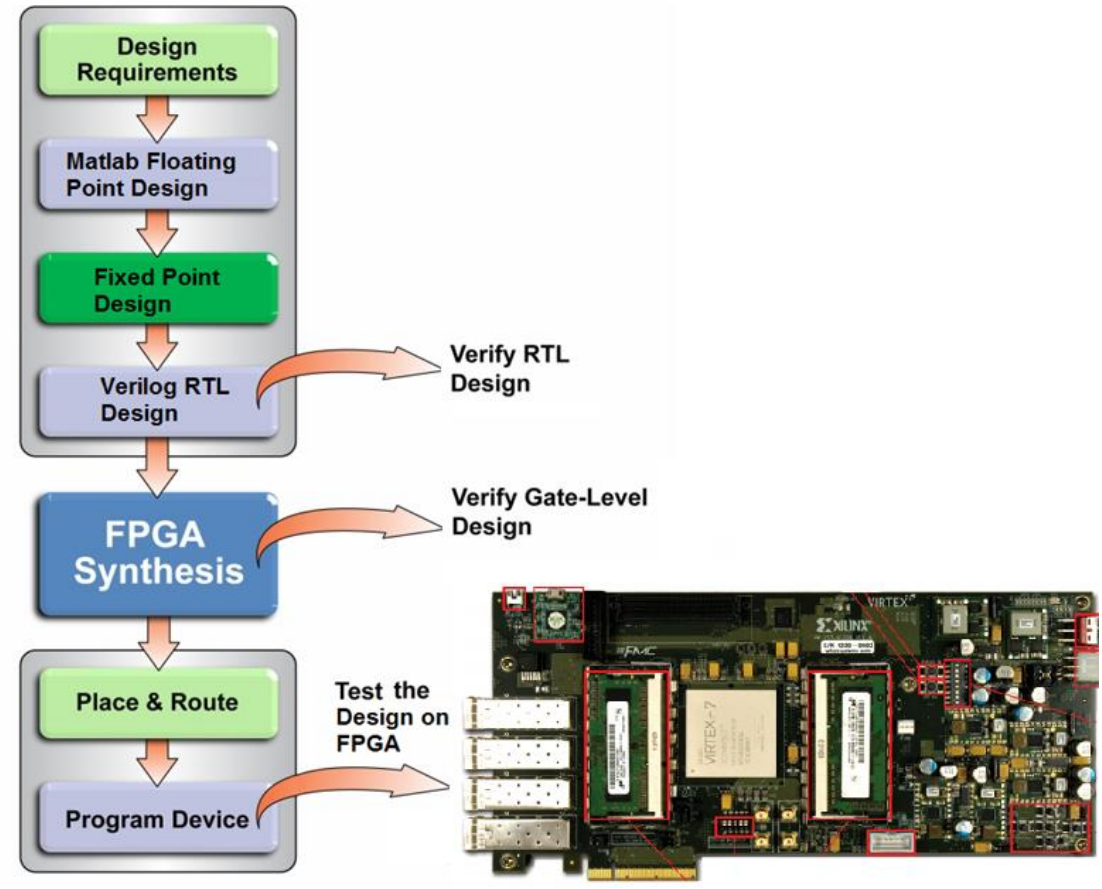


Figure 4.1 FPGA implementation steps.

1- Design Requirements:

   In this study, the problem involves 256 x 256 grayscale images.

2- MATLAB Floating Point Design:

   In this step, straight-forward implementation was made on MATLAB with using nested loops. In this step, there were not used any look up tables (LUT). MATLAB gives Cosine and Sinus values in 64 bits length.

3- Fixed Point Design:
   In this step, self-created LUT were used in order to decrease operation bit length. The main reason of converting from floating point to fixed point is to specify the bit length for the Verilog design. LUTs include the fixed point values for the values of Cosines and Sines in length of 22 bits. The values of Cosines and Sines were calculated the interval between -17 and +17 degrees. Converting from floating point to fixed point was completed with this procedure and bit length is reduced to achieve performance increase.
4- The code that was verified on MATLAB was written on FPGA with Verilog Hardware Description Language
5- FPGA Synthesis:
   In the previous step, verification was made on the level of Register Transfer Level (RTL). In this step, the code is verified on gate level.
6- Place & Route:
   In this step, the code was optimized in order to decrease energy consumption.
7- Program Device:
   In this step, optimized code was uploaded to the card.

Flowchart for verification of the code on RTL level and gate level for single correlation block architecture is shown in Figure 4.2.

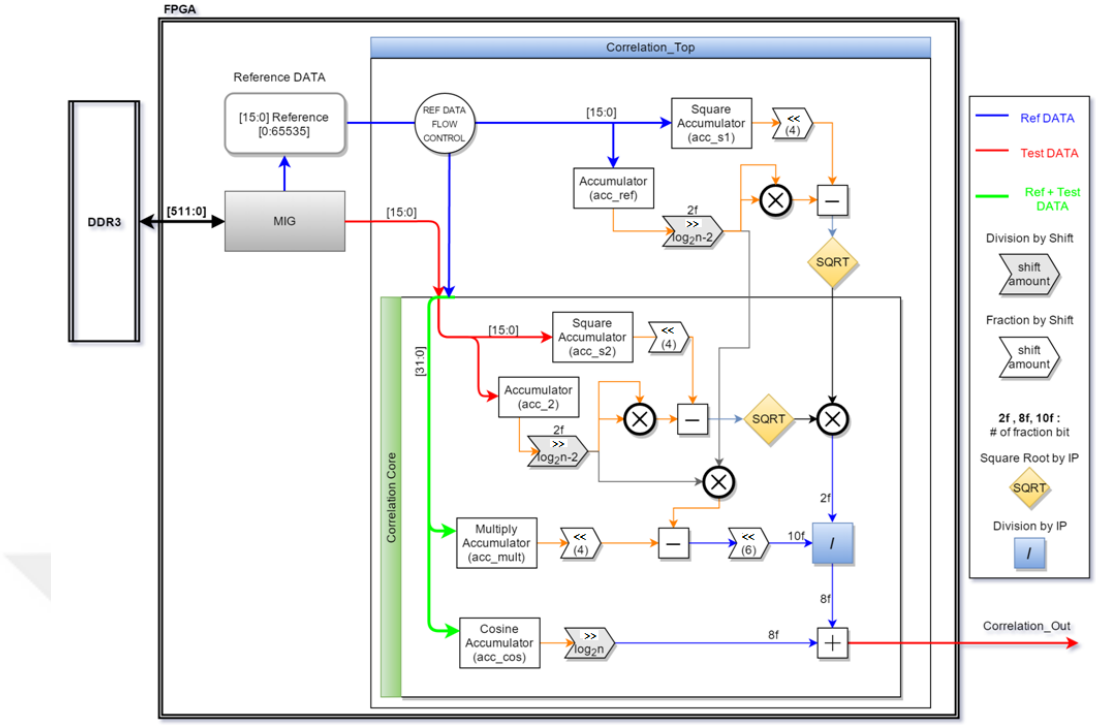


Figure 4.2 Single correlation block architecture.

Parallelization flowchart of the system is shown in Figure 4.3. Each core of the parallelized system has single correlation part.

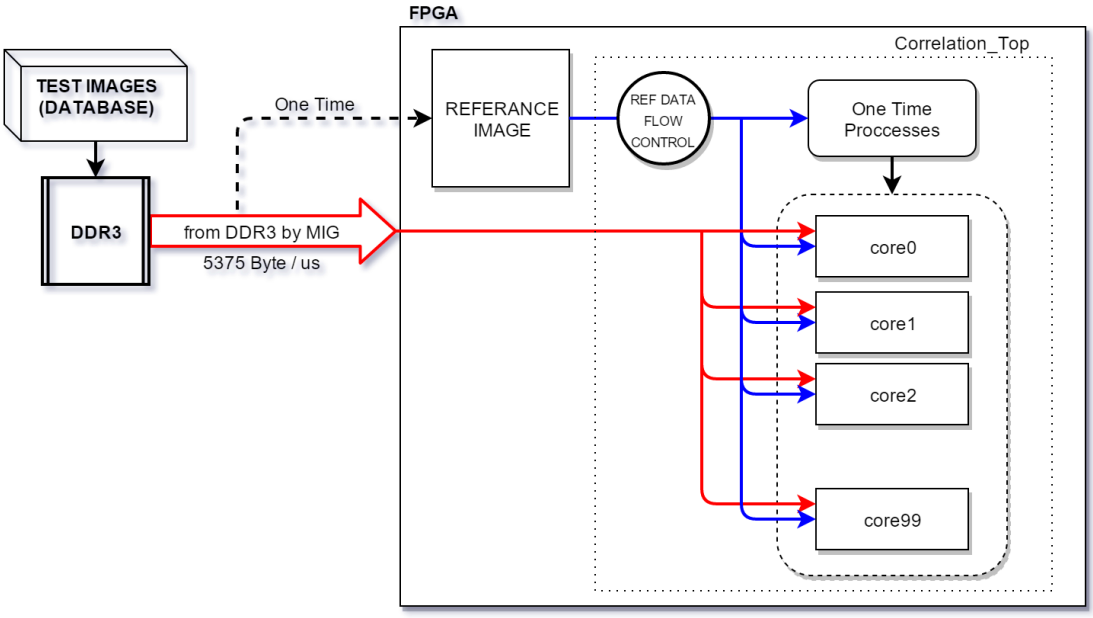


Figure 4.3 Parallelization of the image matching.

4.2. VERILOG

Verilog HDL is a C-like language that allows users to define units at the hardware level and to implement these units on the FPGA. By using this language, RTL and gate level designs can be made and task-efficient hardware can be obtained [25].

Because of its similarity to the C programming language, Verilog HDL is preferred for FPGA programming.

4.3. NCC Architecture Explanation

NCC function are divided into parts, named cores, in our study. This kind of structure provides better understanding users and get traceable code.

Image data blocks coming from DDR3 memory are segmented according to their indices. Reference image values are calculated once. Correlation between reference image and test images are calculated in parallel.

At the Correlation Core module NCC function are implemented for two images. One of them is reference and the other one is one of the test images. These operations are executed in parallel for all images on the database.

At the Correlation_Top, module comparisons of two images are made. Correlation results are calculated for two related images as shown at the right bottom of the Figure 4.2.

CHAPTER 5

COMPARISON OF GPU AND FPGA HARDWARE

For image comparison, Virtex 7 series FPGA unit and GeForce GTX 960M GPU were used. In this chapter the FPGA and GPU hardware features are given and compared.

5.1. VIRTEX-7 Evaluation Board

The Virtex 7 FPGA VC707 Kit is a high speed serial base platform that using Virtex 7 XC7VX485T-2FFG1761C FPGA. The board allows for DDR3 SODIMM memory, an 8-lane PCI Express® interface, a tri-mode Ethernet PHY, general purpose I/O, and two UART interfaces while additional functions can be obtained via cards connected to the VITA-57 FPGA mezzanine connectors (FMC) [26].

For memory, the board features 1GB DDR3 SODIMM and 128MB Linear byte peripheral interface (BPI) Flash memory. It is USB 2.0 ULPI compatible. It can generate clock signals using 200 MHz fixed and I2C programmable LVDS oscillators. The board has FMC1 HPC connector (eight GTX transceivers) and FMC2 HPC connector (eight GTX transceiver) [26].

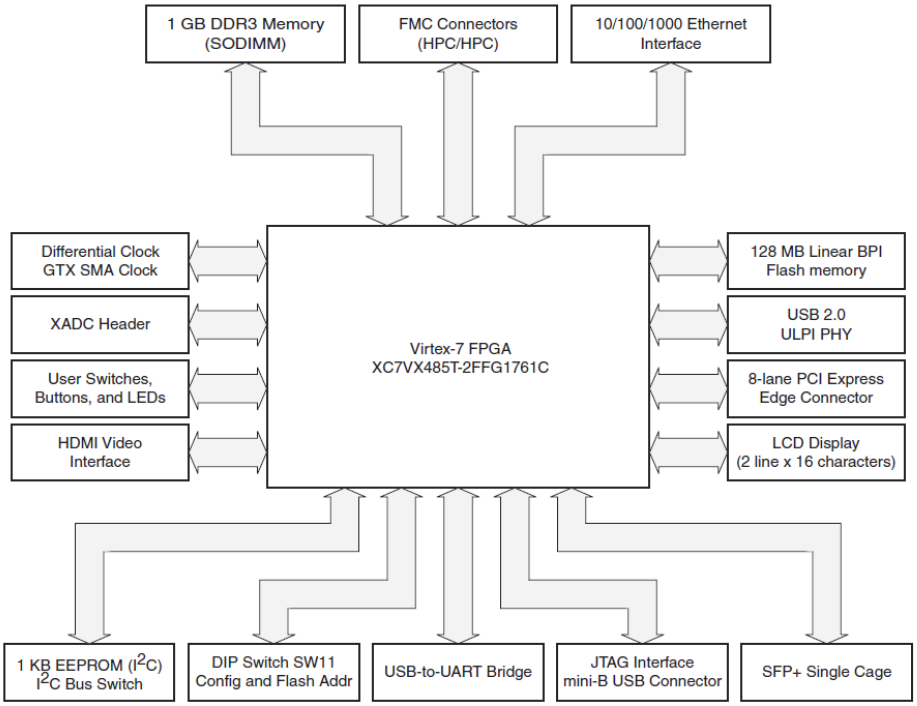


Figure 5.1 VC707 Board Block Diagram [26].

The SODIMM memory is smaller compared to conventional DIMMs. The functionality of both memories are the same while SODIMMs take up half the space of DIMMs. Hence the “small outline” (SO) prefix is added [27].

The SODIMM memory is smaller compared to conventional DIMMs. The functionality of both memories are the same while SODIMMs take up half the space of DIMMs. Hence the “small outline” (SO) prefix is added [27].

5.2. GPU – GeForce GTX 960M

The GTX 960M GPU is based on the GM107 chip and can use 2048-4096 MB GDDR5 video memory, which is attached via 128-bit memory interface at 5000 MHz memory speed. The NVIDIA Maxwell Architecture is used. The GPU has 1097-1202 MHz clock. The GPU supports 640 shading units (i.e. CUDA cores) [28].

The memory bandwidth of the GTX 960M is 80,19 GB/s. The GPU has 6310 MB/s CPU to GPU transfer speed and 6355MB/s GPU to CPU transfer speed. The GPU to GPU transfer speed is 67704 MB/s.

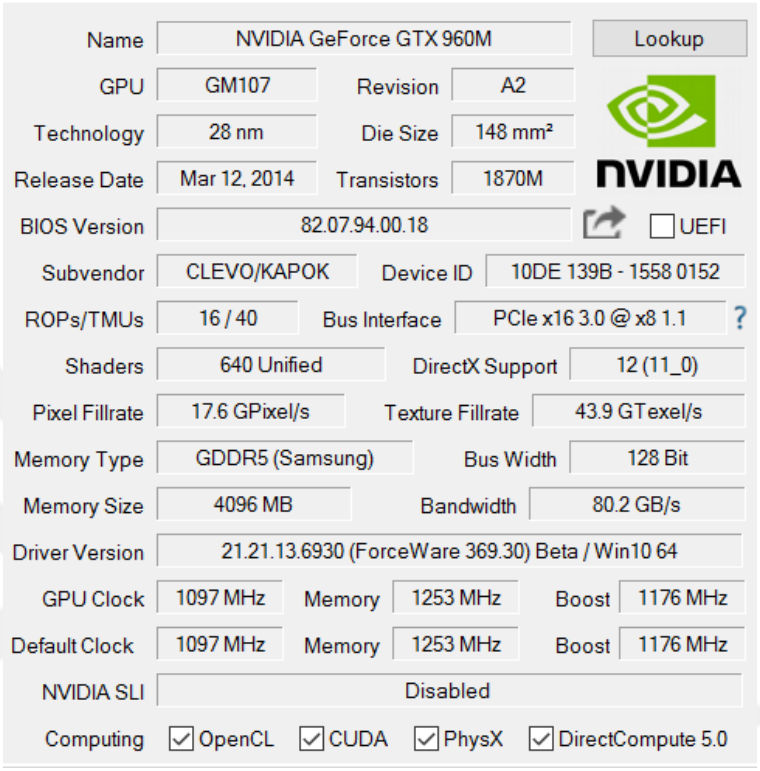


Figure 5.2 GTX 960M features.

```
Device 0: GeForce GTX 960M
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  33554432                     6310.9

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  33554432                     6355.3

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)        Bandwidth(MB/s)
  33554432                     67704.7

Result = PASS
```

Figure 5.3 GTX 960M Bandwidth and data transfer speeds.

Devices properties may vary depending on their hardware. Also, their prices should be taking into considerations this kind of studies like I did.

Price comparison of these two devices are given at Table 5.1.

Table 5.1 Price Comparison.

	Virtex-7 Evaluation Board	GeForce GTX 960M
Price*	3500 $	200 $

*Average price values on the market are shown in 2017.

CHAPTER 6

RESULTS

4 of the fingerprint images and correlation values to each other are shown in Table 6.1. These samples are in grayscale bmp format with the size of 256 x 256.



(a) (b) (c) (d)

Figure 6.1 Fingerprint samples.

Table 6.1 Correlation Values of Four Sample Fingerprints.

Reference Image	Test Image	Correlation Result
a	a	1.0000
a	b	0.6787
a	c	0.5503
a	d	0.6938

6.1. Performance Comparison

Performance calculations with respect to time for FPGA and GPU devices are shown in Table 6.2.

Table 6.2 Performance Measurements By Real Time.

Time	Virtex-7 Evaluation Board # Images	GeForce GTX 960M # Images	FPGA/GPU
1 second	129,6	5,6	23
1 minute	7776	336	
1 hour	466560	20160	
Each ref image has 441*35=15435 correlation			

Results clearly show that FPGA performance for NCC implementation approximately 20 times faster than NCC implementation on GPU.

GPU and FPGA comparison for detecting fingerprint images are investigated in this thesis. It has been seen that FPGA implementation is far better than GPU implementation. GeForce GTX 960M has 640 CUDA cores. Besides, FPGA has empty programmable logic blocks. Taking into the considerations of trade-offs between GPU and FPGA, GPU has faster than operational clock frequency, but well-structured parallel blocks of FPGA provides better performance with respect to time parameter as natural.

REFERENCES

[1] V. PODLOZHNYUK, "Image Convolution with CUDA," 2007. [Online]. Available: http://igm.univ-mlv.fr/~biri/Enseignement/MII2/Donnees/convolutionSeparable.pdf.

[2] V. PODLOZHNYUK, "FFT based 2D Convolution," 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.2396&rep=rep1&type=pdf.

[3] A. KHARLAMOV and V. PODLOZHNYUK, "Image Denoising," 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/imageDenoising/doc/imageDenoising.pdf.

[4] S. CHE, M. BOYER, J. MENG, D. TARJAN, J. W. SHEAFFER and K. SKADRON, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370-1380, October 2008.

[5] M. GARLAND, S. LE GRAND, J. NICKOLLS, J. ANDERSON, J. HARDWICK, S. MORTON, E. PHILLIPS, Y. ZHANG and V. VOLKOV, "Parallel Computing Experiences with CUDA," *Micro, IEEE*, vol. 28, no. 4, pp. 13-27, 2008.

[6] Z. YANG, Y. ZHU and P. - YONG, "Parallel Image Processing Based on CUDA," in *International Conference on Computer Science and Software*, 2008.

[7] J. HUANG, S. PONCE, S. PARK, C. YONG and F. QUERK, "GPUAccelerated Computation for Robust Motion Tracking Using the CUDA Framework," in *5th International Conference on Visual Information Engineering*, 2008.

[8] J. FUNG and S. MANN, "Using graphics devices in reverse: GPU-based Image Processing and Computer Vision," in *IEEE International Conference on Multimedia and Expo*, 2008.

[9] P. LEI, G. LIXU and X. JIANRONG, "Implementation of Medical Image Segmentation in CUDA," in *International Conference on Information Technology and Applications in Biomedicine*, 2008.

[10] D. K. Karna, S. Agarwal and a. S. Nikam, "Normalized Cross-correlation based Fingerprint Matching," *Fifth International Conference on Computer Graphics, Imaging and Visualization*, pp. 229-232, 2008.

[11] Y. KIM, M. HWANGBO and KANADE, "Realtime Affine-photometric KLT Feature Tracker on GPU in CUDA Framework," in *Robotic Institute*, Carneige Mellon University, 2009.

[12] M. BOYER, D. TARJAN, S. ACTON and K. SKADRON, "Accelerating Leukocyte Tracking using CUDA: A Case Study in Leveraging Manycore Coprocessors," in *IEEE International Symposium on Parallel&Distributed Processing*, 2009.

[13] S. Asano, T. Maruyama and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," *In Proceedings of the 2009 Field Programmable Logic and Applications* , pp. 126-131, 2009.

[14] E. Fykse, *Performance Comparison of GPU, DSP and FPGA implementations of image processing and computer vision algorithms in embedded systems*, Master of Science Thesis, Norwegian University of Science and Technology, 2013.

[15] N. Wilt, The CUDA Handbook - A Comprehensive Guide, Indiana: Pearson Education, Addison-Wesley.

[16] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, vol. 21, p. 977–1000, 2003.

[17] J. Sanders and E. Kandrot, CUDA by Example - An Introduction to General-Purpose GPU Programming, Michigan: Addison-Wesley, 2010.

[18] NVIDIA, "CUDA C Programming Guide," NVIDIA, September 2016. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf. [Accessed 03 01 2016].

[19] R. Farber, CUDA Application Design and Development, Waltham: Elsevier Inc., 2011.

[20] NVIDIA, " WHAT IS GPU-ACCELERATED COMPUTING?," NVIDIA, [Online]. Available: http://www.nvidia.com/object/what-is-gpu-computing.html. [Accessed 25 12 2016].

[21] H. Hamzacebi, *CUDA BASED IMPLEMENTATION OF FLAME DETECTION ALGORITHMS IN DAY AND INFRARED CAMERA VIDEOS,* Ankara: Master of Science Thesis, Bilkent University, 2011.

[22] NVIDIA, "Maxwell: The Most Advanced CUDA GPU Ever Made," [Online]. Available: https://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/. [Accessed 04 01 2017].

[23] NVIDIA, "What Is CUDA?," NVIDIA, [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed 25 12 2016].

[24] NVIDIA, "Optimizing Parallel Reduction in CUDA," 2016. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf. [Accessed 2017].

[25] Verilog.com, "Verilog Resources," 2012. [Online]. Available: http://www.verilog.com/. [Accessed 2017].

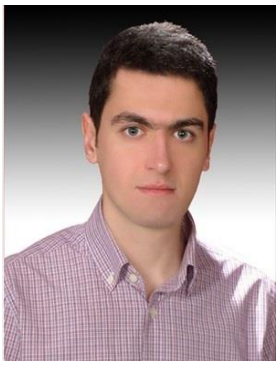
[26] "VC707 Evaluation Board for the Virtex-7 FPGA User Guide," [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf. [Accessed 25 12 2016].

[27] F. Decker, "SODIMM Vs. SDRAM," [Online]. Available: https://www.techwalla.com/articles/sodimm-vs-sdram. [Accessed 26 12 2016].

[28] F. Glasier, "Nvidia GeForce GTX 960M Review," [Online]. Available: http://www.notebookcheck.net/Nvidia-GeForce-GTX-960M-Review.137893.0.html. [Accessed 26 12 2016].

[29] "Optimizing Parallel Reduction in CUDA," 2016. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf. [Accessed 2017].

CURRICULUM VITAE



PERSONAL INFORMATION

Name Surname : İrfan Alp GÜRKAYNAK
Date of Birth : 06/09/1988
Phone : +90 507 6547069
E-mail : iagurkaynak@ybu.edu.tr

EDUCATION

High School : Fethiye Kemal Mumcu Anatolian High School/ANKARA (2002-2006) (4.69/5.0)
Bachelor : Ankara University (2007-2012) (3.23/4.0)
Master Degree : Ankara Yıldırım Beyazıt University (2014-continued)

PROFESSIONAL EXPERIENCE

Research Asst. : Ankara Yıldırım Beyazıt University (2014-continued)

TOPICS OF INTERESTS

-Parallel Programming
-Robotic Systems
-Digital Systems

January 2017 **ANKARA**	**Department of Electronics and Communication Engineering**	**İrfan Alp** **GÜRKAYNAK**	