GPU Implementation in Visual Biometric Features: Fingerprint and Facial Recognition

Name: Yin Yiing Yeoh
GW ID: G26939366
Course: CSCI 6461
Instructor: Morris Lancaster
Date: March 18th, 2015

Table of Contents

I. Introduction

Biometrics refers to methods of identifying individuals using their unique physical and biological traits. There are various types of biometrics: chemical, visual, auditory and behavioral. Chemical biometrics consists of DNA matching; Visual biometrics consists of eyes, ear, fingerprint and face recognition; Auditory biometrics consists of voice recognition; Behavioral biometrics consists of gait, which is the use of individuals walking style for identification. [1] Fingerprints and face recognition are two of the most widely used biometric features in identification due to their usability and reliability of systems. Fingerprints are used in various applications such as forensic identifications, ID cards, access controls etc. Face recognition, on the other hand, is often used in surveillance systems to identify and track persons of interest. Both the fingerprint and face recognition have been such a popular metrics that they are even used as security features in unlocking smartphones.

Fingerprint analysis can be divided into two main categories: verification and identification. Verification is about determining if two fingerprints were produced by the same finger while identification is about matching an input fingerprint to those in the database. [2] There are two main techniques to match fingerprints: minutiae-based matching and pattern matching. Minutiae-based matching is the most widely used matching technique. Pattern matching simply compares two images and sees how similar they are. A typical face processing system, on the other hand, comprises of face detection, face recognition and tracking. Face detection is about separating face from its background while face recognition is about matching the face detected to the intended face image.

As security has gradually become an important domain, there is an increase pressure in being able to process fingerprint and face recognition and tracking quickly in order to determine person of interests and keep track of them through surveillance system. However, the current CPU performance is not able to handle the large amount of database images and the intensity of fingerprint matching, face detection and recognition algorithms quickly. Through this paper, the algorithms used in fingerprint matching, face detection and recognition proposed by various authors and GPU implementation in these areas will be examined in detail. Though ideally I would like to have a published paper that addresses all three face detection, recognition and tracking using one particular algorithm in surveillance system that works in real-time and robust environment, however, there is no such paper being published currently. Hence, I've chosen several papers that address some combinations of the three face processing elements using various algorithms in real-time.

The remainder of the paper is organized as follows: Section II provides the background of fingerprint and face processing systems along with its algorithms; Section III describes about GPU, its uses and structures, and how the various algorithms in both fingerprint and face processing system utilizes GPU to speed up its performance; Section IV presents the experimental results from various researches; Section V concludes the paper and outlines possible future work.

II. Background

Fingerprint System

Minutiae Matching

A minutia is a change on the ridges of the fingerprint. Though there are various types of ridge features, the two fundamental features are ridge endings and bifurcation. A bifurcation is a split along a ridge path. Figure 1 depicts the two main features of a minutia. Figure 2 depicts the minutia on a fingerprint. Figure 3 depicts other types of ridges.

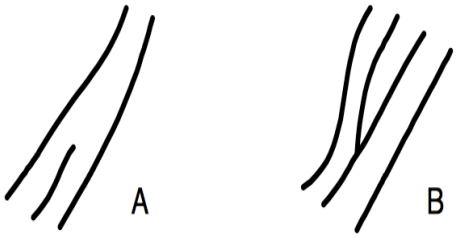


Figure 1: Ridge ending (A) and ridge bifurcation (B) [3]

Figure 2: Ridge ending and bifurcation on a fingerprint [3]

Figure 3: Other fingerprint characteristics [3]

There are two types of minutia-based matching algorithms: local and global. Local minutia matching defines a neighborhood and finds similarities between each minutia of two fingerprints within a neighborhood. The neighborhood can be classified into nearest neighbor (i.e. neighbors are defined as K closest minutia from the central minutia) and fixed radius (i.e. neighbors are defined to be within a radius R to the central minutia). Global minutia matching uses the result from local minutia matching and finds sets of matching minutia pairs.

Minutia Cylinder-Code Algorithm

The Minutia Cylinder-Code algorithm (MCC) [4] is one of the most elaborated algorithms used in the fingerprint matching system. It combines both local and global matching to achieve its accuracy. Each minutia is associated with a 3-dimensional structure called cylinder where it stores its neighborhood characteristics. The cylinder is located in the center of the minutia and has a fixed radius R and fixed height of 2π. All minutiae have cylinders of the same size. Given an input fingerprint and database fingerprint, a series of cylinders will be computed for both fingerprints. Only cylinders with sufficient information will be retained. This results in cylinder set of both input fingerprint and database fingerprint. Once we've obtained this cylinder set, local matching will be conducted. In the local matching process, every pair of cylinders will be compared and determined if they are matchable and the results will be stored in a matrix. Once the local similarities between cylinders are determined, a global matching will be conducted where scores will be computed based on the pairs of the matrix.

Face Processing System

A face processing system consists of face detection, recognition and tracking. In this section, three types of algorithms will be presented. Each of these algorithms offers to process a combination of two out of the three elements: Viola and Jones for detection and tracking; Local Binary Pattern for detection and recognition; Histogram for detection and tracking.

Viola and Jones Algorithm

Viola and Jones is the first to use Adaboost learning algorithm to detect face in real-time. Viola and Jones algorithm uses integral image, learning algorithm based on Adaboost and cascaded classifiers to distinguish faces and non-faces in an image. This algorithm utilizes Haar features to do so. There are four kinds of Haar features, as shown in Figure 4.

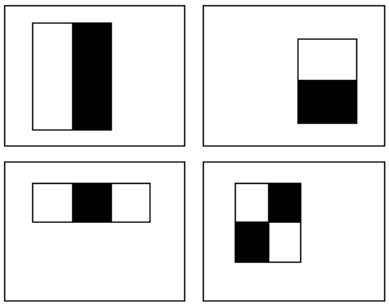


Figure 4: Four kinds of Haar features [5]

Each feature has a value that is calculated by the difference between the sum of pixel values within white region and the sum of pixel values within the dark region. The computational cost to calculate the sum of the pixel values in a region can be reduced by using integral image where the integral image at location (x,y) is the sum of pixels above and to the left of the location. After the feature value is calculated, a classifier will be constructed based on the feature value and Adaboost learning algorithm will be employed

to select the best classifiers. According to Viola and Jones work, the detection test window best works in the size of 24 x 24 pixels. Thus, any images with other sizes will have to be rescaled. The rescaled images will have a series of 24 x 24 sub-windows. To accelerate the face detection in these sub-windows, Viola and Jones uses cascade classifiers where the procedure is shown in Figure 5. If a sub-window is rejected on the first stage (i.e. the sub-window does not contain a face), then it will not be passed on to the consequent stages to be processed.

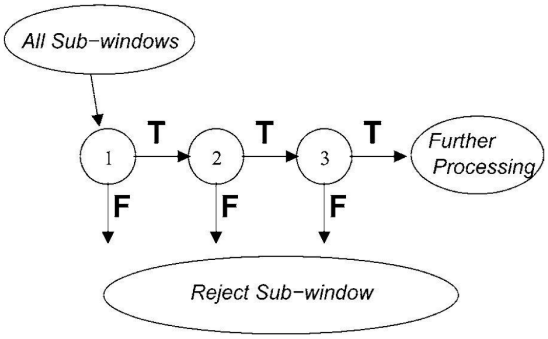


Figure 5: Use of cascade classifiers to check sub-windows [5]

The above Adaboost face detection method assumes uniform illumination. However, in order to track a person through the surveillance, it is highly possible that the surveillance will experience illumination changes or even human pose changes. Combining algorithm proposed by Thota et al. [6] and image based parametric illumination model along with the above face detection method, we could easily track a detected face under robust environments. Thota et al. proposed a fusion of Kale and Janes illumination compensation with the face detection. A detailed description about the algorithm can be found in Thota et al. [6]

Local Binary Patterns

Local Binary Patterns (LBP) uses both shape and textures to describe face image features. Before we could perform face recognition (i.e. match an extracted face image to database face), we need to first detect a face in an image. Figure 6 depicts the steps in LBP-based face detection. There are three dominant steps in LBP-based face detection: resizing, LBP and scanning.

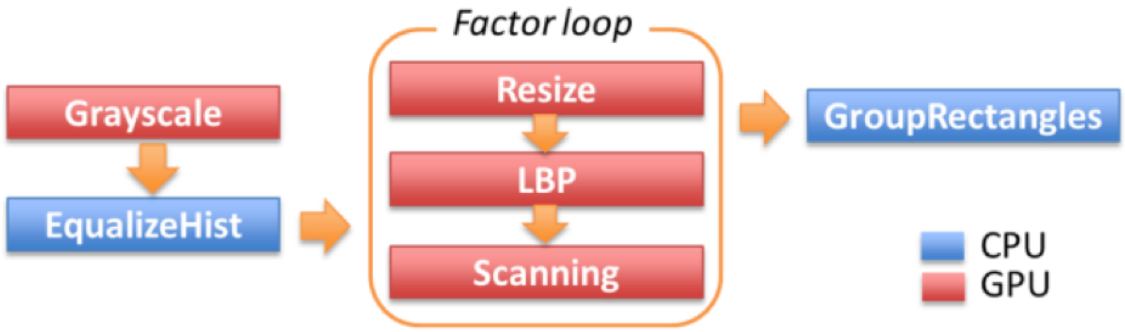


Figure 6: Pipeline of LBP-based face detection algorithm [7]

Given an input image, a scale factor will be employed to resize the image into multiple smaller images. Since the size of the face in the input image is unknown, the search for a face in the image should be made with different patches. In order to detect the face more efficiently, a fixed patch size would be maintained and the input image will be resized instead (refer to Figure 7(a)). Then the LBP operation is applied to each of the resized images. The LBP operation is about comparing each pixel in the input image with its neighboring pixels and then encoding it into binary code. The result of the entire pixel comparisons forms the LBP feature value. Once the image is transformed into LBP domain, the patch slides through all search points (Figure 7(b)) to classify if a window contains a face or not by comparing the LBP feature values in the current patch with some trained value. If a window passes through all the stages in the classifier, it is then

classified as a face (Figure 7(c)). The classifier is called cascade classifier. Note that this cascade classifier is also used in Viola and Jones algorithm. The combination of both the sliding patch and cascade classifier forms a step known as scanning. Once a face has been detected in an input image, the histogram of the LBP will be generated and the chi-square distance will be computed. This process is also repeated for a database image. Then, both the input image and database image will be compared. Figure 8 provides a summary of the face recognition/matching steps. The chi-square distance is used to measure the similarity between two LBP images and the calculation is defined as follow:

$$\chi^2(x,\xi) = \sum_{j,i} w_j \frac{(x_{i,j}-\xi_{i,j})^2}{x_{i,j}+\xi_{i,j}}$$

where χ and ξ are histograms to be compared and indices i and j refer i^{th} bin in the histogram corresponding j^{th} region.

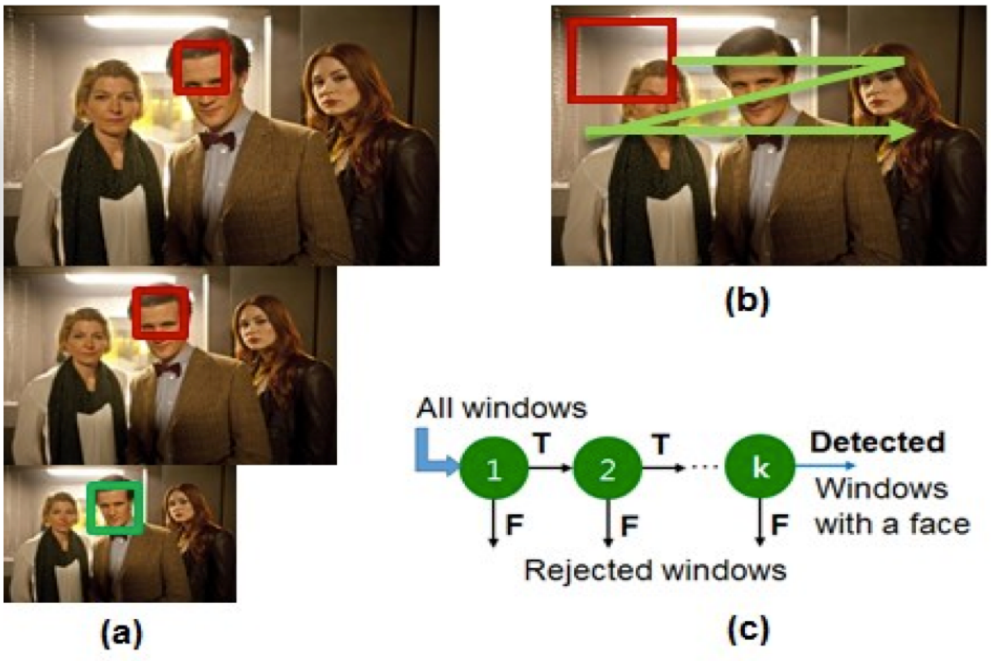


Figure 7: (a) Resizing, (b) Scanning, (c) Cascade classifier in Scanning [7]

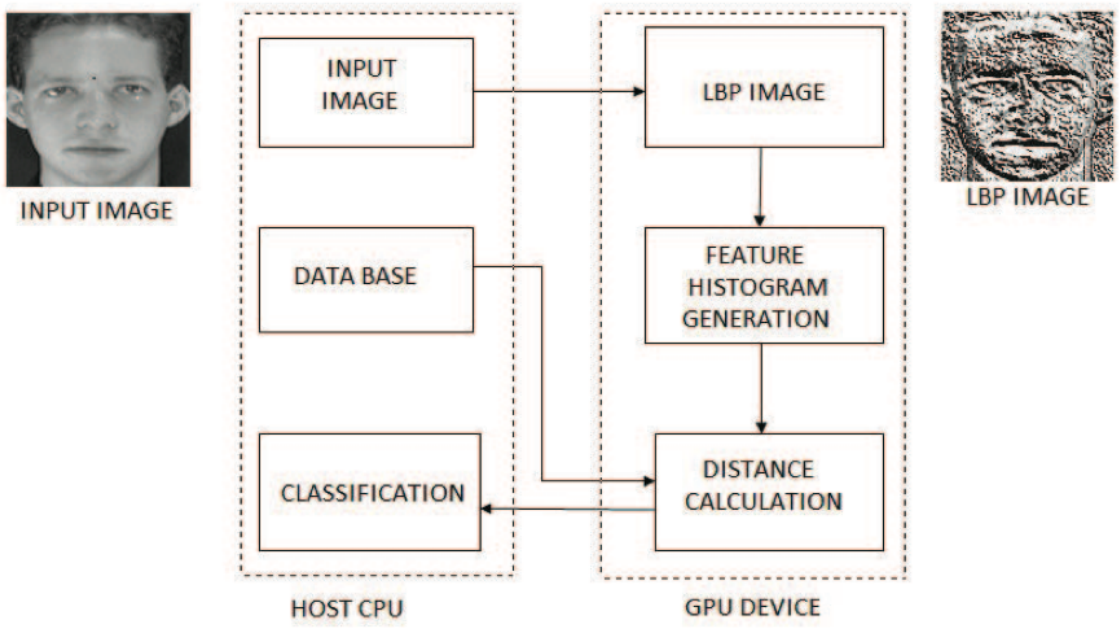


Figure 8: Summary of LBP-based face recognition [7]

Histogram of Gradients

Dalal et al. [8] proposed a face detection method based on histogram of gradients (HoG) feature descriptor. Given an input image, a sliding window is applied and traverses across the frame to extract HoG features. This extraction is performed on every position and scale and each of them is independent from one another. The HoG features are obtained through gradient computation. After getting the HoG features, we calculate some true positive and false positive rate and detect a face by filtering out all the false positives (i.e. the face is represented by the true positives). In order to obtain a more accurate result, motion and appearance are added to the HoG feature to reduce the occurrence false positive. Once the face is detected, it is tracked by EMD tracking algorithm with SURF points.

III. Graphics Processing Units

GPU Background

Graphics Processing Unit (GPU), initially designed to process 3D graphics in games and CAD applications, is a data-parallel computing device that consists of a set of multiprocessing units. Each of the multiprocessing units is a set of Single Instruction Multiple Data (SIMD) processing cores. It is this SIMD architecture that allows parallelism to take place. Though primarily used in computer graphics, video games and visual computing industry, GPU has found its way into other fields recently due to its capability of performing large-scale parallelism, which can be utilized to accelerate computations performance.

In order to facilitate the use of GPU for non-graphic developers, NVIDIA introduced a programming model called Compute Unified Device Architecture (CUDA). CUDA provides a friendly interface where non-graphic developers can use GPU to perform general-purpose computations without the need to transform algorithms to graphics terminology. With CUDA, a GPU can be programmed as a co-processor to the CPU and programs executed on GPUs, known as kernel, are always organized into a grid. A grid consists of thread blocks, which are composed of threads. Thread are instances of kernel and threads in a grid share the same code and execute the same instruction, but each thread can work on either the same or different datasets. Threads of the same block always run on the same multiprocessor and share a small high-speed memory area called shared memory. Memory latencies can be hidden more effectively and performance can be improved if there are massive threads running.

Open Computing Language (OpenCL) is another programming model to write GPU-accelerated programs. Compared to CUDA, OpenCL could be used to run across heterogeneous platforms, which in turn increases the portability of GPU programming capability. Two of the important OpenCL terminologies are work-item and work-group. A work-item is equivalent to CUDA threads and work-group is equivalent to CUDA thread blocks. Similar to CUDA where threads of same block have a shared memory, a local memory is shared within the work groups as well.

GPU Implementation on Fingerprint Matching

In order to improve the performance in GPU fingerprint matching, two measures have been taken: reduction of GPU idle periods and packaging of several matching processes into one. [2] Typically, if a GPU is being fed with one thread, it will turn into idle state when the CPU is performing the matching process for the input fingerprint, then GPU will return to its active state after CPU has finished processing and another thread is being fed into GPU (refer Figure 9 top). To achieve efficiency, GPU is being supplied with two threads instead of one, where GPU will perform the matching process for the database fingerprints i and i+1 (refer Figure 9 bottom). At the next iteration, it will process database fingerprints i+2 and i+3 and so on. This greatly reduces GPU idle time.

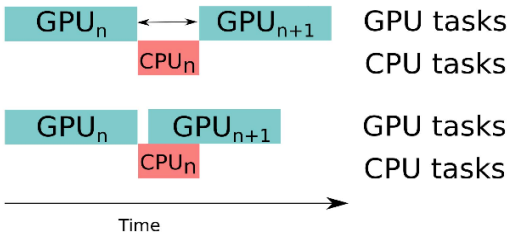


Figure 9: Reduction of GPU idle time [2]

Also, in general, replacing many small workloads with fewer bigger workloads helps enhance performance. Hence, instead of processing an input fingerprint with one

database fingerprint, the kernel can call to process the input fingerprint and a set of database fingerprints. Figure 10 depicts this packaging.

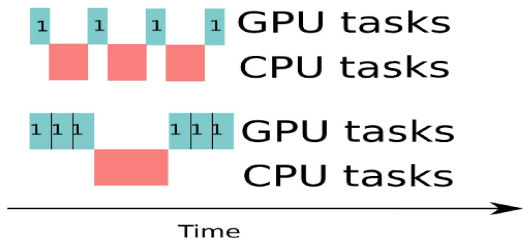


Figure 10: Package several matching processes [2]

In addition, multi-GPU will be employed to improve efficiency in computation and matching process. Two GPUs will be used where each GPU is assigned to perform half the computation of the minutia. As for the matching process, each GPU could either perform half of the matching process (collaborative) or each GPU could compute the matching score of the input fingerprint with different database fingerprints (in parallel).

Also, depending on the fingerprint database size, memory transfer between the database fingerprint and GPU global memory could slow down the overall process. This problem will occur if the database size is so large that it is impossible to transfer the entire database into the GPU's memory during the initialization process. In this case, asynchronous memory transfers can be employed where the database is divided into chunks and transfers are being done in chunks. While the input fingerprint is matched to chunk i (computation associated with chunk i), chunk i+1 is loaded into the GPU memory (transfer of chunk i+1) in parallel (refer to Figure 11).

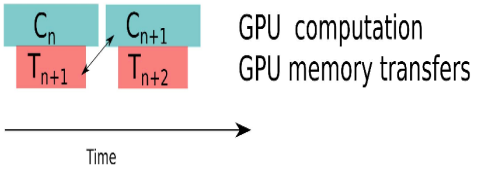


Figure 11: Asynchronous memory transfers [2]

GPU Implementation on Face Processing

Recalled from Section II Viola and Jones Algorithm, there are three main steps in this algorithm: resize original image, calculate integral image and detect face using cascade classifiers. To achieve efficiency in face detection, all three steps are run in parallel where the resizing of $(n+2)^{th}$ frame being done in parallel with the integral image calculation of the $(n+1)^{th}$ frame and face detection of n^{th} frame. Besides utilizing parallel implementation, spatial locality will also be exploited where write-once, read many data are employed for data such as the integral images, features, cascades, illumination coefficients and detected sub-windows. For instance, the cascades are populated in the shared memory from textures for faster access. It is essential to store the result of the calculations from face detection in the local memory instead of global memory so that the face recognition process can easily access these information, thus speeds up the process.

While using the HoG method for face detection and tracking, parallelism can be utilized in the extraction of HoG features. As mentioned in Section II Histogram of Gradient, HoG feature will need to be extracted in every position in the input image and they are very computational intensive all together. By running all these calculations in multi-thread (shown in Figure 12), it helps to speed up the performance. In addition, the result of the HoG computation is stored in a shared memory for faster access.

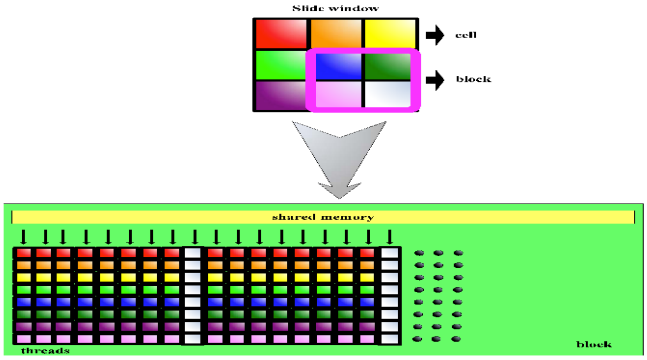


Figure 12: The diagram of GPU based HoG feature extracting algorithm. Different color represents different cells in the sliding window. In the blocks, different color demonstrates the threads used for processing corresponding pixels in the cells. [9]

As for LBP, parallelism is employed in various steps in both face detection and face recognition. Recalled from Section II Local Binary Pattern, LBP operation is performed after the input image is resized and the LBP operation involves the comparison of each pixel in the input image with its neighboring pixel to generate a binary code. To improve efficiency, each pixel in the image can be accessed simultaneously through work items in a work group. Hence, the work items in the work group are executed in parallel. On top of that, scanning is one of the bottlenecks for LBP-based face detection steps; one way to resolve this issue is to perform multi-phase scanning. Sequential execution of multi-phase kernels can speed up processing time as compared to simultaneous execution in one kernel. Figure 13 illustrates multi-phase kernel launching which could be applied to the scanning step. In this case, instead of evaluating all the patches in a single kernel, we would have N kernels process one Nth of the total patches. The last two calculations, histogram generation and chi-square distance calculation, can also be used to help boost the performance. Though histogram generation is not computational intensive, it is a highly memory bound operation. To reduce memory overhead, generated histogram are stored at locally shared memory for easy access. As for the chi-square distance calculation, the sum reduction for the sigmas in the equation is parallelized by mapping each reduction to one thread.

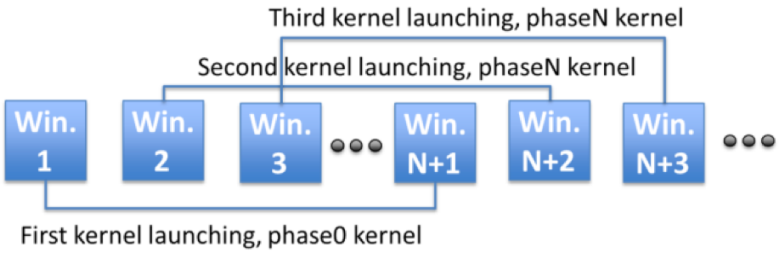


Figure 13: Multi-phase kernel launching [7]

IV. Experimental Results

Fingerprint Matching Using MCC Algorithm

In this experiment, two types of GPUs (Tesla GPU and GTX GPU) are used and three different databases with various sizes and types of fingerprints are being used. These two GPUs are integrated using CUDA. The three databases are listed as follows:

1. DB4 database: Consists of 2,000 pairs of rolled fingerprints and have average number of minutiae of 135.84
2. DB14 database: Consists of 27,000 pairs of rolled fingerprints (only the first 19,000 were used) and have average number of minutiae of 206.9
3. SfinGe database: Consists of 100,000 synthetic fingerprints and have average number of minutiae of 40.69

The experiment is tested with single GPU and multi-GPU. Table 1 to Table 6 shows the results using only one GPU on the three databases. Table 7 and Table 8 shows the results of using multi-GPU where two Tesla GPUs is experimented with the SfinGe database. As mentioned in Section III GPU Implementation on Fingerprint Matching, there are two ways to perform the matching processing in multi-GPU: collaborative or in parallel.

Table 1 [2]

DB4 RESULTS USING A TESLA DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	23914.0	692.0	34.6
Ns 16; LSS	93539.0	1756.0	53.3
Ns 8; LSSR	102474.1	2581.5	39.7
Ns 16; LSSR	174963.7	3530.3	49.6

Table 2 [2]

DB4 RESULTS USING A GTX DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	23914.0	603.6	39.6
Ns 16; LSS	93539.0	2004.5	46.7
Ns 8; LSSR	102474.1	1810.7	56.6
Ns 16; LSSR	174963.7	3187.8	54.9

Table 3 [2]

DB14 RESULTS USING A TESLA DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	633 546.0	25 415.9	24.9
Ns 16; LSS	2 205 491.2	42 103.0	52.4
Ns 8; LSSR	2 269 426.8	113 308.4	20.0
Ns 16; LSSR	3 749 775.2	105 424.8	35.6

Table 4 [2]

DB14 RESULTS USING A GTX DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	633 546.0	21 854.6	29.0
Ns 16; LSS	2 205 491.2	49 868.2	44.2
Ns 8; LSSR	2 269 426.8	80 559.8	28.2
Ns 16; LSSR	3 749 775.2	91 140.5	41.1

Table 5 [2]

SFINGE RESULTS WITH A TESLA DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1998.8	30.2
Ns 16; LSS	239 241.4	4763.0	50.2
Ns 8; LSSR	141 631.1	3568.6	39.7
Ns 16; LSSR	331 559.1	6486.2	51.1

Table 6 [2]

SFINGE RESULTS WITH A GTX DEVICE

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1796.3	33.6
Ns 16; LSS	239 241.4	5888.1	40.6
Ns 8; LSSR	141 631.1	2775.4	51.0
Ns 16; LSSR	331 559.1	6994.6	47.4

Table 7 [2]

SFINGE RESULTS USING TWO TESLA DEVICES COLLABORATING

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1374.8	43.9
Ns 16; LSS	239 241.4	3250.5	73.6
Ns 8; LSSR	141 631.1	2932.7	48.3
Ns 16; LSSR	331 559.1	4689.5	70.7

Table 8 [2]

SFINGE RESULTS USING TWO TESLA DEVICES WORKING IN PARALLEL

	CPU time (ms)	GPU time (ms)	Speed-up
Ns 8; LSS	60 396.0	1024.0	59.0
Ns 16; LSS	239 241.4	2393.9	99.9
Ns 8; LSSR	141 631.1	1839.5	77.0
Ns 16; LSSR	331 559.1	3288.6	100.8

From the above tables, it can be concluded that the GPU implementation in the fingerprint matching algorithm is more efficient than the CPU implementation. The differences between the results in single GPU experiment between Tesla GPU and GTX GPU are due to their architecture differences. As for the multi-GPU experiment, the increase in speed-up is greater when the matching process is performed in parallel as compared to in collaboration. This is because data needs to be exchanged when the process is done collaboratively and hence slows down the process. Note that LSS and

LSSR listed in the table two techniques to compute the local similarities. Local matching is mentioned in Section II Minutia Cylinder-Code Algorithm, however the two different techniques of local matching, LSS and LSSR are not being discussed in this paper. The N_S 8 and N_S 16 listed in the table indicate the size of the cylinder and the subject of how cylinder sizes of a minutia can affect the computation performance is not being addressed in this paper.

Face Detection and Tracking Using Viola and Jones Algorithm

In this experiment, GTX GPU is used and face detection is performed on images of various sizes taken either from a static database or live frames captures from a camera. This GPU is integrated using CUDA. All these images have complex background and different illuminations. The result (Figure 14) shows that the GPU implementation is 12 to 38 times faster than the CPU implementation.

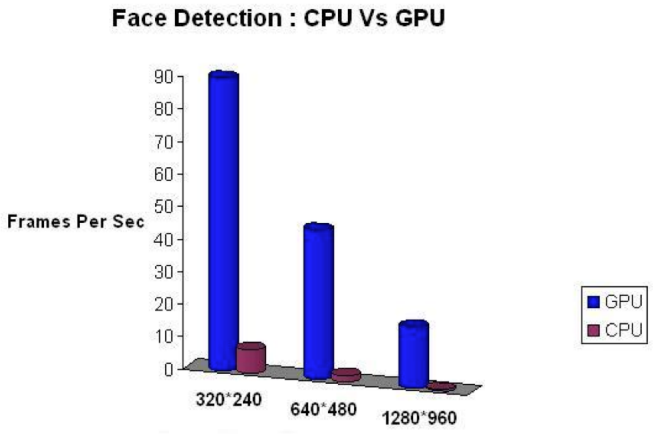
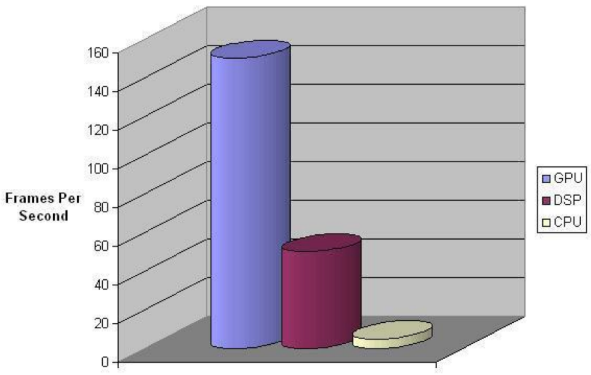


Image Dimension

	320 × 240	640 × 480	1280 × 960
GPU	90	46	19
CPU	7.4	2.15	0.5

Figure 14: Detection rate in frames per second of GPU and GPU face detector for images of various resolutions [5]

As for face tracking, various indoor and outdoor extreme illumination conditions are tested. The results of tracking rates are shown in Figure 15. Three indoor illumination modes/centroids (refer to Figure 16) and two outdoor illumination modes are tested (refer to Figure 17).



	Tracking Speed(Frames Per Sec)
GPU	150
DSP	50
CPU	5

Figure 15: Tracking rate for different implementations [5]

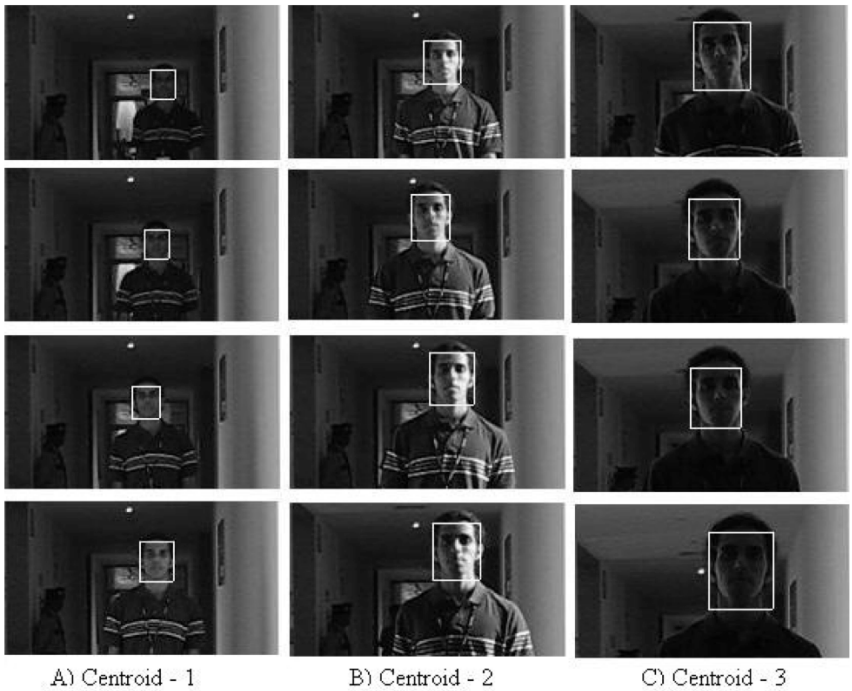


Figure 16: Face tracker output under 3 extreme illumination conditions in an indoor environment [5]

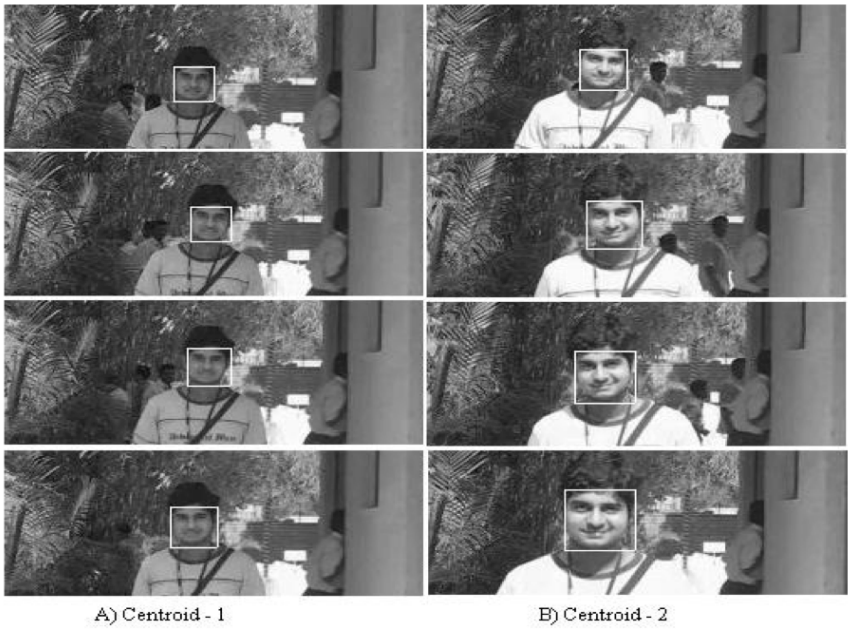


Figure 17: Face tracker output under 2 illumination conditions in an outdoor environment. Centroid-1 corresponds to a shadowed environment while Centroid-2 corresponds to a sun-lit environment. [5]

From the above results, it can be concluded that the use of Viola and Jones algorithms in face detection and tracking in extreme conditions are reliable and the use of GPU implementation is advantageous.

Face Detection and Recognition Using LBP Algorithm

In this section, results from two papers will be presented where one focuses on both face detection and face recognition while another focuses only on face recognition.

Experimental Result 1 (Face Detection and Face Recognition)

In this experiment, Mali GPU and Tegra K1 GPU are used and a movie trailer that consists of 3000 frames is used as input image for face detection. Mali GPU is integrated using OpenCL while Tegra K1 GPU is integrated using CUDA. Table 9 and Table 10 show the difference in execution time and speedup of the face detection for each GPU

and CPU respectively.

Name	CPU (ms)	OpenCL (ms)	Speedup
Grayscale	4.06	3.44	1.18x
Resize	17.90	6.70	2.67x
LBP	32.83	4.94	6.64x
Scanning	59.93	21.14	2.83x
FPS	**8.25**	**23.50**	**2.84x**

Table 9: Execution time and speedup of face detection on Mali GPU [7]

Name	CPU (ms)	CUDA (ms)	Speedup
Grayscale	2.95	1.78	1.66x
Resize	11.05	2.78	3.97x
LBP	26.01	0.57	45.63x
Scanning	48.52	13.98	3.47x
FPS	**10.81**	**40.98**	**3.79x**

Table 10: Execution time and speedup of face detection on Tegra K1 GPU [7]

After the faces are detected from the movie trailer, they are used to compare with database faces from FERET database for the face recognition process. The FERET database contains 250 images of upper body of 86 people. Note that the faces of the FERET database images have been extracted manually before the experiment. Table 11 and Table 12 show the difference in execution time and speedup of the face recognition for each GPU and CPU respectively.

Name	CPU	CUDA	Speedup
Histogram	0.15	-	-
Chi-Dist	6.57	1.67	x3.93
FPS	10.47	38.47	x3.67

Table 11: Execution time and speedup of face recognition on Mali GPU [7]

Name	CPU	OpenCL	Speedup
Histogram	0.05	0.5	0.1x
Chi-Dist	12.61	2.59	4.86x
FPS	7.57	21.85	2.89x

Table 12: Execution time and speedup of face recognition on Tegra K1 GPU [7]

Experimental Result 2 (Face Recognition Only)

In this experiment, AMD 6500 GPU and ORL database that consists of 40 subject images are employed. This GPU is integrated using OpenCL. Table 13 shows the computational time with the use of GPU and CPU implementation and Figure 18 depicts the relative performance increase. Both Table 13 and Figure 18 indicate that the GPU implementation is better than CPU implementation.

Image Size	***Sub Histograms***	**CPU (Core 2 Duo)**	**GPU (AMP 6500)**
		Feature Ext.(ms)	***Feature Ext.(ms)***
128x128	16	34.5	25.9
225x225	49	105.3	27.3
256x256	64	134.6	28.5
512x512	256	545.6	36.3
1024x1024	1024	2204.7	64.7

Table 13: Performance of LBP Feature Extraction [10]

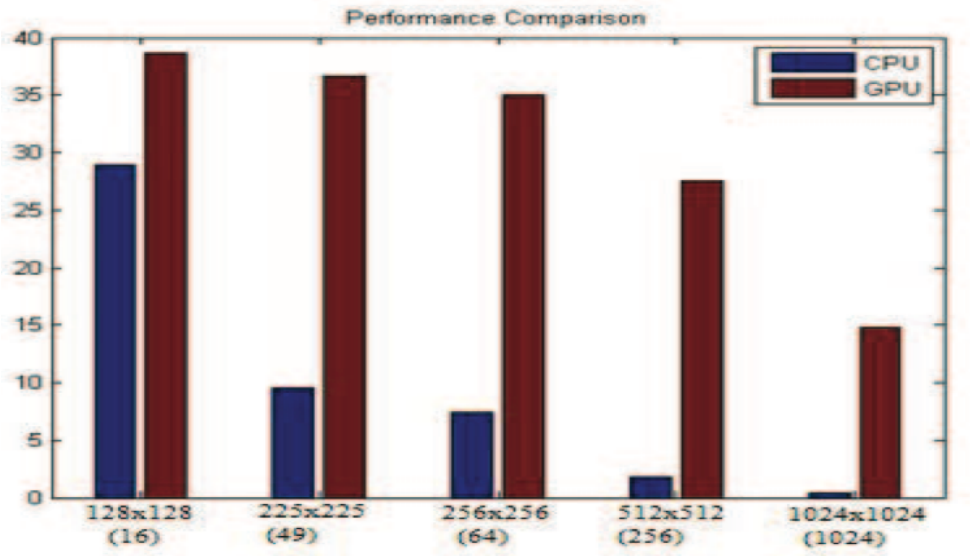


Figure 18: Computational Efficiency of GPU vs CPU [10]

According to the results from the above two experiments, it is evident that implementing GPU in the LBP-based face detection and recognition enhance the computational performance of the algorithm.

Face Detection and Tracking using Histogram of Gradients

In this experiment, GeForce GTX GPU is used and the face detection and tracking are tested on 4 videos with different resolutions. This GPU is integrated using CUDA. Table 14 shows the results of face detection with both CPU and GPU implementation. As for face tracking, a sequence of PETS video data is used. This video tracking is compared with the EMD tracking algorithm with color signature based method proposed by Zhao et al. [11]. Figure 19 shows the result of EMD tracking algorithm with SURF points and Zhao's method. It can be concluded that in video based face detection and tracking, the use of GPU helps to boost the performance and the EMD tracking algorithm with the support of SURF points is more reliable. Although the tracking algorithm didn't utilize the use of GPU, this could possibly be a future area of research where this tracking algorithm will be able to exploit GPU to result in better performance.

	Resolution	CPU Time(ms)	GPU Time(ms)	Speedup
Video1	352 × 288	3600	77	46.8
Video2	528 × 384	7800	162	48.1
Video3	704 × 576	20000	296	67.6
Video4	1280 × 720	45000	664	67.8

Table 14: Comparison between CPU and GPU for face detection [9]

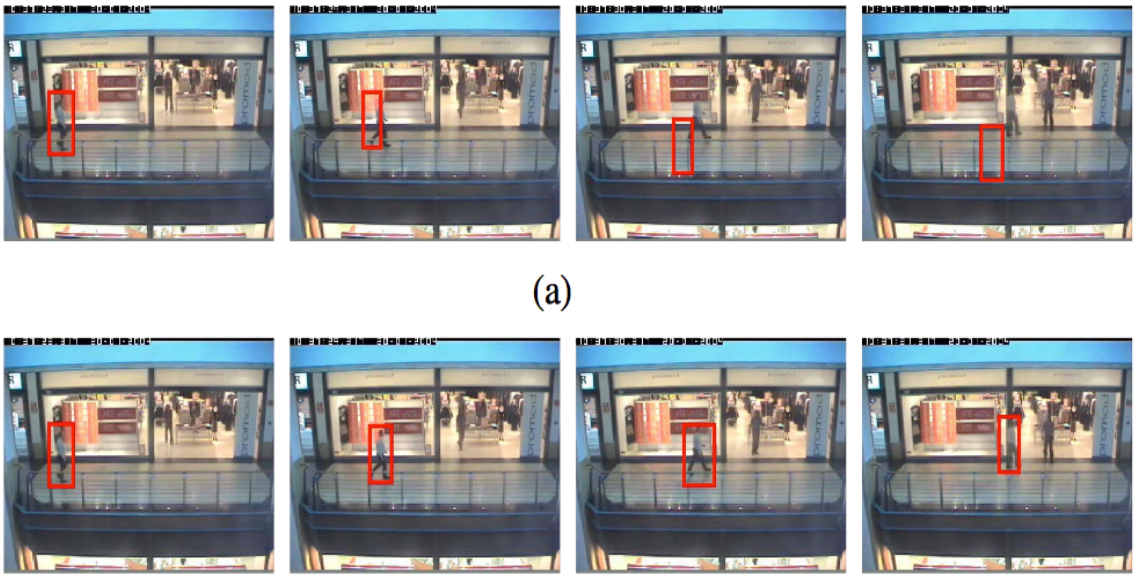


Figure 19: (a) The color signature based method has lost track of the main. (b) The EMD tracking algorithm used maintain a robust focus on the object throughout the sequence with the support of SURF points. [9]

V. Conclusion/Future Work

This paper presented the implementation of GPU in fingerprint matching and face processing system and all the experimental results confirm that GPU helps in enhancing performance as compared to stand-alone CPU implementation. It is evident that similar mechanisms are being utilized to achieve efficiency in both systems where parallelism is used to speed up the algorithm intensive computations and quick access to information is available at the local shared memory.

In terms of future work, there are plenty of areas where research could be conducted. For GPU implementation in the fingerprint matching system, there currently exist only one published paper on high performance matching system; hence there are opportunities to conduct more research on this topic. Also, since GPU is famous for its image processing capability, researchers could utilize this property to conduct research on fingerprint image enhancement using GPU as there's a high possibility that fingerprint images collected from crime scenes are not complete and distorted, and thus image enhancement are required.

As for the GPU accelerated face processing system, there exists various algorithms and methods for face detection, recognition and tracking. However, there is no published paper that addresses all three elements of the face processing system using one algorithm in surveillance system. This could also possibly be a potential research area. Lastly, the security issues and forensic examinations on GPUs are still fragmentary. In order to leverage the usage of GPU in the defense industry, this issue must be addressed.

VI. References

[1] Biometrics Institute, "Types of Biometrics." http://www.biometricsinstitute.org/pages/types-of-biometrics.html

[2] Gutierrez, P.D.; Lastra, M.; Herrera, F.; Benitez, J.M., "A High Performance Fingerprint Matching System for Large Databases Based on GPU."*IEEE Transactions on Information Forensics and Security, Volume 9, Issue 1,* 2013, pg. 62-71

[3] FBI Biometric Center of Excellence, "Fingerprint Recognition." http://www.fbi.gov/about-us/cjis/fingerprints_biometrics/biometric-center-of-excellence/files/fingerprint-recognition.pdf

[4] Cappelli, R; Ferrara, M.; Maltoni, D., "Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence Volume 32, Issue 12,* 2010, pg. 2128-2141

[5] Sharma, B.; Thota, R.; Vydyanathan, N.; Kale, Amit, "Towards a Robust, Real-time Face Processing System using CUDA-enabled GPUs." *2009 International Conference on High Performance Computing,* 2009, pg. 368-377

[6] Thota, R.; Kalyansundar, A.; Kale, Amit, "Modeling and Tracking of Faces in Real-life Illumination Conditions." *2009 IEEE International Conference on Acoustics, Speech and Signal Processing,* 2009, pg. 761-764.

[7] Saehanseul, Yi; Illo Yoon; Chanyoung Oh; Youngmin Yi, "Real-time Integrated Face Detection and Recognition on Embedded GPGPUs." *2014 IEEE 12th Symposium on Embedded Systems for Real-time Multimedia,* 2014, pg. 98-107

[8] Dalal, N.; Triggs, B, "Histograms of Oriented Gradients for Human Detection." *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 1,* 2005, pg. 886-893

[9] Di Xie; Lu Dang; Ruofeng Tong, "Video Based Head Detection and Tracking Surveillance System." *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery,* 2012, pg. 2832-2836

[10] Dwith, C.Y.N.; Rathna, G.N., "Parallel Implementation of LBP based face recognition on GPU using OpenCL." *2012 13th International Conference on Parallel and Distributed Computing Applications and Technologies,* 2013, pg. 755-760

[11] Qi Zhao; Brennan, S; Hai Tao, "Differential EMD Tracking." *IEEE 11th International Conference on Computer Vision,* 2007, pg. 1-8