Performance Tuning Techniques for Face Detection Algorithms on GPGPU

Yara M. Abdelaal, M. Fayez, Samy Ghoniemy, Ehab Abozinadah, H.M. Faheem

Abstract**:** ***Face detection algorithms varies in speed and performance on GPUs. Different algorithms can report different speeds on different GPUs that are not governed by linear or near-linear approximations. This is due to many factors such as register file size, occupancy rate of the GPU, speed of the memory, and speed of double precision processors. This paper studies the most common face detection algorithms LBP and Haar-like and study the bottlenecks associated with deploying both algorithms on different GPU architectures. The study focuses on the bottlenecks and the associated techniques to resolve them based on the different GPUs specifications.***

Keywords**:** ***Face Detection, GPU Performance, LBP, Haar-Like.***

I. INTRODUCTION

Graphics Processing Units (GPUs) are commonly used nowadays in almost any mobile device, laptop, and personal computer. GPUs are mainly used as accelerators for computer vision and AI applications because of the huge processing power they can provide for solving computationally intensive problems. However, GPU performance may be affected dramatically by the memory access patterns and floating-point operations. In case the algorithm you are targeting to deploy on GPU has such challenges then you will need to tune the performance of the GPU by modifying your algorithm to optimally exploit the GPU capabilities. One of the most common problems in computer vision is face detection, which requires a huge processing power to identify the location of the face(s) inside an image. The most common techniques for face detection are Local Binary Pattern (LBP), Haar-like Features, and Eigen faces. Many attempts to deploy those algorithms on GPU are made and will be discussed in Section II. Section III discusses the problem of face detection and the challenges associated with deploying the face detection algorithms on GPU. Section IV discusses the performance tuning techniques to optimize the LBP, and Haar-like detectors to optimally utilize the GPU capabilities. Section V presents the results of the optimization techniques proposed in Section III against different GPU architectures. Section VI shows the conclusion remarks of this work

Revised Manuscript Received on December 05, 2020.

* Correspondence Author

Yara M. Abdelaal, Computer Systems, Ain Shams University, Cairo, Egypt, Email: yara.medhat@cis.asu.edu.eg

M. Fayez*, Computer Systems, Ain Shams University, Cairo, Egypt, Email: Mahmoud.fayez@cis.asu.edu.eg

Samy Ghoniemy, Computer Systems, British University in Cairo, Cairo, Egypt, Email: samy.ghoniemy@bue.edu.eg

Ehab Abozinadah, Information Systems, King Abdelaziz University, Jeddah, KSA, Email: eabozinadah@kau.edu.sa

H. M. Faheem, Computer Systems, Ain Shams University, Cairo, Egypt, Email: hmfaheem@cis.asu.edu.eg

II. RELATED WORK

Face detection is an image scanning problem that requires the image to be scanned to find the location of the face(s) inside the image. The face detection algorithm must scan the image with different scanning widow sizes because the face size is variable, and its location is not predictable. This fact makes the face detection one of the best algorithms to be implemented on GPU because the GPU has many Scalar Processors (SP) that can handle those variable scanning windows in parallel and speed up the process of face detection. So, there are many attempts in the literature to use the GPUs for face detection.

Previous attempts were made to parallelize LBP algorithm to reach real time face detection as proposed by Narmada et. al [1] using OpenCL reaching computation time of 20ms for a 640*480 image resolution. Also an attempt on non-standard image resolutions was made by Marwa et al. [2] which we couldn’t consider as a benchmark to compare with. Attempts to evaluate the different performance on multiple architectures was introduced by Miguel et. al [3] introduced the different performance on general purpose processors, SIMD units, Multi-core architectures and GPUs reaching 30fps and measuring the energy needed along with the performance.

P. Král et al [6] proposed a change to the algorithm under ELBP (Enhanced Linear Binary Pattern) where the enhanced algorithm reached 2% increase in the detection accuracy, but the authors did not mention the performance in terms of detection rate/ computation time. Fayez et al. [4] proposed viola-jones enhancement on GPU reaching 37fps for HD Images, which is considered as the base to our study along with a research proposed by Rafi et al. [5] and Fayez et al. in [6] which was based on introducing an implementation for LBP algorithm by both scaling feature scaling and image scaling and a comparison between both in terms of accuracy and performance time reaching up to 50 Frame per second (fps) for feature scaling and 45 fps for image scaling.

III. PROBLEM DEFINITION

Scanning an image to find a face requires fixing the scanning windows size; then evaluate all sub-images that can fit in the scanning widow against a trained data set of features like LBP features or Haar-like features or project the sub-image into the eigenfaces domain and measure the distance between the projected sub-image and the faces domain. This work focus on LBP and Haar-like features because of the similarities between these two algorithms that makes it useful to compare them on different GPU architectures.

The scanning window size is then scaled up and the process is repeated with bigger sub-images till a face is detected. In case bigger sub-images size do not match the size of the feature set, then hawse may have two solutions which are:

- Solution 1 is to scale down the sub-image to match the size of the feature set size.
- Solution 2 is to scale up the feature set to match the size of the sub-images' size.

Solution 1 cannot be calculated offline so it will be executed with every new input image. On the other hand, solution 2 can be calculated offline and it will be executed once at the startup of the algorithm. So solution 2 is the most proper way to solve this problem as shown in many trials as in [4], [7], [8].

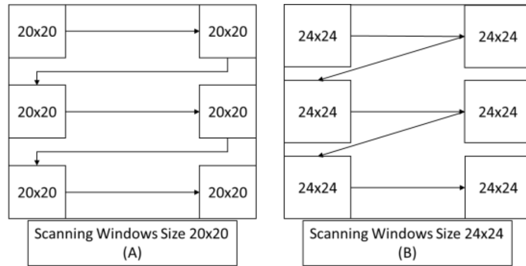

Figure 1Scanning Window moving from top left to bottom right. (A) Scanning window of size 20x20, and (B) Scanning window of size 24x24

The number of scanning windows' sizes affects the number of total sub images to be evaluated against the feature set. **Figure *1*** shows two different scanning window sizes 20x20, and 24x24. The scanning windows is moving from left to right with step of 10% of the scanning window size. The number of sub-images resulted from the two scanning windows shown in **Figure *1*** are function of the image size, scanning window size, and the displacement of the windows. This is expressed in (1) where:

- $CF$is the counting function that returns the number of sub-images.
- $W$is the input image width
- $H$is the input image height
- $w$is the scanning windows width
- $h$is the scanning window height
- $s$is the windows displacement to find a new sub-image

$$\boldsymbol{CF(W,H,w,h,s)} = \frac{\boldsymbol{W-w}}{\boldsymbol{s}} * \frac{\boldsymbol{H-h}}{\boldsymbol{s}} \qquad (1)$$

Scaling up the scanning windows and applying (1) again and sum up all the results of CF function will result in the total number of sub-images that must be evaluated against the feature set. The scanning window sizes are predefined as shown in (2) which states that scanning window is scaled up by 20% till the scanning window size is approximately equal to 300x300 pixels. **Table*1*** shows the number of sub-images with different window sizes and the total number of sub-images for the most common images resolutions which are VGA (640x480), HD (1280x720) and 4K (1920x1080). Each sub-image will be processed on GPU to check if it has a face or not. The GPU has two options which are scaling up the feature set or loading the scaled feature set from its global memory. This decision must be justified based on the GPU memory bandwidth and the number of GPU Special Function Units (SFU). Next section will discuss the complexity of each approach and the overhead on GPU architecture that is associated with each approach. Also, it will discuss two different feature sets which are LBP and Haar-Like.

$$\boldsymbol{SW} = \{(\boldsymbol{w_i},\boldsymbol{h_i}) : \boldsymbol{w_i} = \boldsymbol{20} * \boldsymbol{1.2^i}, \boldsymbol{h_i} = \boldsymbol{20} * \boldsymbol{1.2^i} \boldsymbol{where 0} \leq \boldsymbol{i} \leq \boldsymbol{15}\} \qquad (2)$$

Table1: Number of sub-images for 3 different image sizes

Scanning Window $SW_i = (w_i, h_i)$	Counting Function CF		
	Image Size (640x480)	Image Size (1280x720)	Image Size (1920x1080)
(20,20)	71300	220500	503500
(24,24)	48767	151767	347600
. . .	. . .	. . .	. . .
(257,257)	130	718	2073
(308,308)	61	423	1312
Total sub-images	219510	693829	1601588

IV. METHODOLOGY

Processing thousands of sub-images that is shown in Table1 means a small enhancement in the sub-image processing time will reflect by order of magnitude on the overall time of the input image processing time. This requires detailed study of the number of clock cycles required to scale up the feature set on GPU and the number of memory cycles to load it from the global memory. This is in order to select the best approach to be deployed with different feature sets such as LBP and Haar-Like feature sets.

The number of clock cycles is a function of many factors which are:

- Ratio between Load/Store (LD) units and Scaler Processors (SP) inside the Stream Multiprocessor (SM)
- Ratio between Special Function Unit (SFU) and SP.

This study focusses on two GPU architectures which are K20m as a Tesla GPU architecture co-processor and RTX2080 Ti as GeForce architecture co-processors. **Table *2*** shows the different hardware specifications for both GPUs. RTX2080 Ti is faster and has more processing power. However, this do not mean a tuned algorithm on K20m will scale up properly to RTX2080 Ti.

Table 2 K20m and RTX2080 Ti architecture specifications

Feature	K20m	RTX2080 Ti
Double Precession Speed	1174 GFLOPs/s	444 GFLOPs/s
Memory Speed	208 GB/s	616 GB/s
LD:SP Ratio	1:4	1:2
SFU:SP Ratio	1:2	1:2

The number of cycles requires to scale up a feature set or load it from GPU memory is the key for a successfully performance tuning. This number of cycles is different based on the feature set. Next sub section discusses the number of cycles required to scale up and load each feature set. This will make it clear and easy to take the right decision.

A. Feature Set Scaling Vs Loading

This section discusses the number of CPU cycles and Memory cycles needed to either scale the feature on GPU or load the pre-scaled feature from GPU. These numbers are feature type dependent. So, we will analyze LBP feature then Haar-Like feature.

1) LBP Feature

LBP feature is shown in Figure 2. It has 9 rectangles; the center rectangle and the 8 surrounding rectangles. Only one point, the width, and height are required to define the 9 rectangles. The LBP feature has the following data structure:

- Top left corner point (x,y), x and y are two integer numbers each has size of 8 bytes.
- Width, and Height of the LBP cell, each has size of 8 bytes.
- Lookup table of 256 bits, stored as 8 integers, which means 64 bytes.
- Left and right values, stored as double, each has size of 8 bytes.

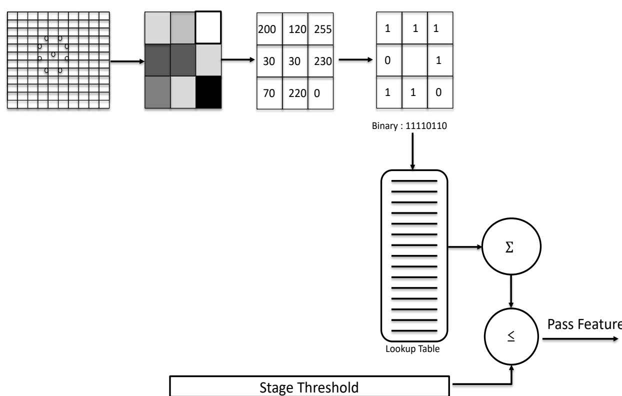

Figure 2LBP feature components

Comparing the center rectangle with the 8 surrounding rectangles will result in 8-bit result that is used to index the lookup table and find the corresponding bit which will decide to either use left or right value for this feature. Scaling the LBP feature do not require to scale all the components of the LBP feature only the top left corner point, width, and height must be scaled. This means scaling up an LBP feature require 4 floating point multiplications. We can neglect the time to load it from memory given that the feature is cached in the SM cache and shared between all SPs. On the other hand, loading the pre-scaled feature set from GPU's global memory means that the all the feature components must be loaded from the global memory. This means loading 104 bytes for each LBP feature only 32 bytes will have random access all other feature attributes are cached in SM cache. This means only 32 bytes must be loaded in case of pre-scaled feature set.

2) Haar-Like Feature

Haar-Like feature has different shapes and different number of rectangles. Figure 3 shows 4 different types of Haar-Like features. The dark rectangles are subtracted from the light rectangles to find out the feature value. Haar-Like feature has the following data structure:

- Rectangle 1 (x,y,w,h), this rectangle is stored in 4 integers which occupies 32 bytes.
- Rectangle 2 (x,y,w,h), this rectangle is stored in 4 integers which occupies 32 bytes.
- Rectangle 3 (x,y,w,h), this rectangle is stored in 4 integers which occupies 32 bytes.
- This rectangle may not be used but it is stored in memory in order to keep the feature size fixed for ease of calculating feature set indices.
- The weight of the 3 rectangles stored as double which occupies 24 bytes.
- Threshold, this is stored as double number which occupies 8 bytes.
- Left and right values, stored as double, each has size of 8 bytes.

After subtracting the dark area from the light area, the difference is compared to threshold based on the comparison result the left or right values are selected.

Scaling the feature means the area of each dark/light rectangle gets bigger. Consequently, the difference gets bigger. This means the difference must be normalized such that it is still comparable with the threshold value.

Scaling the Haar-Like feature consists of 3 different steps which are scaling 3 rectangles, and the weight of each rectangle, then normalize the rectangles' weights regarding the total area of the sub-image. This means calculating the area of sub-image and normalize the weight of the rectangles.

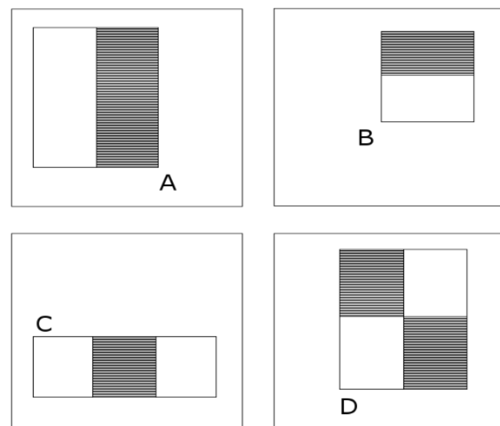

Figure 3Haar-Like Features with different shapes.

The number of cycles requires to achieve the scaling of Haar-Like feature is not clear unless the scaling function is presented as a pseudo code that is shown in Algorithm 1. Lines 2:8 would require 5 Floating point multiplication and repeated 3 times. Lines 9 to 15 would require 4 or 5 floating point multiplications. So, based on the worst-case scenario the total number of floating-point multiplications would be 20 floating point multiplications.

Algorithm 1 Scaling HAAR-Like Feature
SHAAR(*F, SF*)
Input:
F: Input feature
SF: Scaling Factor
Output:
FS: output scaled feature

```
1:  BEGIN
2:  Fori from 0 to F.RectangleCount
3:      FS.rectangle[i].X = F.rectangle[i].X * SF
4:      FS.rectangle[i].Y = F.rectangle[i].Y * SF
5:      FS.rectanlge[i].Width = F.rectangle[i].Width * SF
6:      FS.rectangle[i].Height = F.rectangle[i].Height * SF
7:      FS.rectangle[i].SignAndWeight
                = F.rectangle[i].SignAndWeight
                * SF
8:  End For
9:  If ( F.RectangleCount == 2 )
10:     FS.rectangle[0].dSignAndWeight = -(F.rectangle[1].Width * F.rectangle[1].Height *
11:     F.rectangle[1].dSignAndWeight) /(F.rectangle[0].Width * F.rectangle[0].Height);
12: Else
12:     FS.rectangle[0].dSignAndWeight = -(F.rectangle[1].Width * F.rectangle[1].Height *
13:     F.rectangle[1].dSignAndWeight + F.rectangle[2].Width * F.rectangle[2].Height *
14:     F.rectangle[2].dSignAndWeight) / (F.rectangle[0].Width * F.rectangle[0].Height);
15: End If
16: END
```

On the other hand, loading the pre-scaled feature set from GPU's global memory means that the all the feature components must be loaded from the global memory. This means loading 144 bytes for each Haar-like feature.

3) Performance Tuning

There are many decisions to be considered in order to find the best performance for LBP and Haar-Like detectors on both K20m and RTX2080 Ti. **Table *3*** shows the total cycles required for either scaling the feature set or loading it from global memory.

Table 3 Number of cycles required to scale/load each feature set

Feature Set	Number of Features	Number of Scaling Cycles	Number of Loading Cycles	Scale to Load Ratio
LBP	129	516	4,128	1:8
Haar-Like	2135	42,700	307,440	10:72

It is clear that the number of cycles for scaling the feature set cannot be optimized, however the number of cycles to load the feature set can be optimized if the algorithm sorts the threads running on GPU in a way that makes each SM process a group of threads that requires the same scale. This in turn will lead to load the feature set only once for all threads that are running on the same SM. This is called memory coalescing. This feature is GPU dependent based on the maximum number of threads that can be deployed on single SM. For both K20m and RTX2080 Ti the maximum theoretical number of threads per SM is 1024, however due to the limited number of registers on both GPUs only 768 threads were activated for both LBP and Haar-Like kernels as reported by Nvidia CUDA Profiler (*nvvp*). This would reflect back on **Table *3*** which has theoretical values and would result in an architectures based number of cycles that is shown in **Table *4***.

Table 4 Loading feature set based on memory coalescing for different GPU architectures.

Image Size	LBP			Haar-Like		
	Number of blocks	Total Number of load cycles	Total Number of scaling cycles	Number of blocks	Total Number of load cycles	Total Number of scaling cycles
VGA Image	285	903,536,640	112,942,080	285	87,620,400	9,346,176,000
HD Image	904	2,865,954,816	358,244,352	904	277,925,760	29,645,414,400
4k Image	2086	6,613,254,144	826,656,768	2086	641,319,840	68,407,449,600

Theoretical speed of K20m memory is 208 GB/s and processing speed of double precision is 1174 GFLOPS. RTX2080 ti has theoretical memory speed of 616 GB/s and double precision processing speed of 444 GFLOPS. All theoretical speeds are multiplied by factor 75% because the occupation rate of SM is 75% due to limited register file size.

The performance of the two approaches for any algorithm is subject to the type of the GPU and hence we can predict the appropriate approach that will optimally run on the target GPU. Table 4 shows that the difference between the two approaches for LBP is less than 40X which indicates that RTX2080 Ti will perform better in case of pre-scaled feature set as it has faster memory speed than double precision speed.

V. RESULTS

Experiments are conducted on 2 workstations with XEON processor and 64 GB RAM. Each workstation has only one GPU card. The first workstation has K20m and the second workstation has RTX2080 ti. The reported results of each configuration are shown in **Table *5*** and

Table 5 Results of different approaches on K20m

Image Size	HAAR-Like Features Scaling on GPU	HAAR-Like Pre-Scaled	LBP Features Scaling on GPU	LBP Pre-Scaled
640 x 480	59.989ms	21.597ms	4.2432ms	4.8599ms
1280 x 720	177.74ms	61.246ms	12.449ms	14.480ms
1920 x 1080	312.19ms	108.37ms	27.750ms	31.933ms

Retrieval Number: 100.1/ijitee.B82341210220
DOI: 10.35940/ijitee.B8234.1210220

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication

The results show that the pre-scaled feature set approach for haar-like feature set is faster on both architectures. This is expected because the number of loading cycles shown in Table 4 for haar-like is less than the number of scale cycles by 106X. This means the speed of the double precision operations must be faster than the memory speed by 106X which is not true for both GPUs as reported in Table 2.

On the other hand, LBP feature set reported different performance on both GPUs as shown in Table 5 and Error! Not a valid bookmark self-reference.. This is because the ratio of the scaling cycles to loading cycles are 1:8. This means that faster memory as in case of RTX2080 ti will produce better results for pre-scaled approach for LBP. This is shown in Table 6.

Table 6 Results of different approaches on RTX2080

Image Size	HAAR-Like Features Scaling on GPU	HAAR-Like Pre-Scaled	LBP Features Scaling on GPU	LBP Pre-Scaled
640 x 480	17.475ms	8.1016ms	0.8263ms	0.4454ms
1280 x 720	40.464ms	17.420ms	2.8095ms	1.2373ms
1920 x 1080	61.842ms	27.328ms	6.6696ms	5.0450ms

VI. CONCLUSION

Studying the target architecture and comparing the speed and size of SM components are the main factors that affect the performance of the implementation of face detection algorithms. SM register file limits the number of the threads that run concurrently on the SM. Load and Store units limits the memory speed when accessing the GPU global memory. special function units affect the double precision floating point speed. Also studying the algorithm and dividing it into I/O and Double Precision operations to estimate the ratio and find out the best performance tuning approach for the target GPU is mandatory to exploit the capabilities of the GPU

ACKNOWLEDGMENTS

This work was supported in part by the National ScienceFoundation under Grant CNS-1113191, and in part by the High-Performance Computing Project in King Abdulaziz University.

REFERENCES

1. N. Naik and G. N. Rathna, “Real time face detection on GPU using opencl,” Comput. Sci., 2014.
2. M. Chouchene, F. E. Sayadi, H. Bahri, J. Dubois, J. Miteran, and M. Atri, “Optimized parallel implementation of face detection based on GPU component,” Microprocess. Microsyst., vol. 39, no. 6, pp. 393–404, 2015.
3. M. B. López, A. Nieto, J. Boutellier, J. Hannuksela, and O. Silvén, “Evaluation of real-time LBP computing in multiple architectures,” J. Real-Time Image Process., vol. 13, no. 2, pp. 375–396, 2017.
4. M. Fayez, H. M. Faheem, I. Katib, and N. R. Aljohani, “Real-time Image Scanning Framework Using GPGPU-Face Detection Case Study,” in Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), 2016, p. 147.
5. M. R. Ikbal, M. Fayez, M. M. Fouad, and I. Katib, “Fast Implementation of Face Detection Using LPB Classifier on GPGPUs,” in Intelligent Computing, 2019, pp. 1036–1047.
6. Y. Abdelaal, M. Fayez, and H. Faheem, “PERFORMANCE EVALUATION OF IMAGE SCANNING: FACE DETECTION CASE STUDY,” Adv. Comput. Sci. Eng., vol. 18, pp. 1–15, 2019.
7. M. Chouchene, F. E. Sayadi, H. Bahri, J. Dubois, J. Miteran, and M. Atri, “Optimized parallel implementation of face detection based on GPU component,” Microprocess. Microsyst., vol. 39, no. 6, pp. 393–404, 2015.
8. M. R. Ikbal, M. Fayez, M. M. Fouad, and I. Katib, “Fast Implementation of Face Detection Using LPB Classifier on GPGPUs,” in Intelligent Computing-Proceedings of the Computing Conference, 2019, pp. 1036–1047.

AUTHORS PROFILE

Yara Medhatis an associate lecturer at faculty of computer and information sciences at Ain Shams University. She received her MSc. in computer science in 2014; her master was about face detection optimization on GPU. She studied different Face Detection algorithms and benchmarked their performance on the GPU. She proposed a new algorithm that can run faster on GPU under certain conditions and assumptions. She is a PhD student working on optimizing face detection and recognition pipelines on heterogeneous architectures including GPUs and Xeon Phi Coprocessors. Her main research interested are Computer Vision, Augmented reality, Parallel architectures, and machine learning.

Mahmoud Fayezis a lecture at Ain Shams University. He received the BSc. degree in Computer Systems from the University of Ain Shams, Egypt, in 2007, and the MSc in Computer Science from Ain Shams University, Egypt, in 2014. In 2008, he joined the Department of Computer Systems at Ain Shams university as a Teaching Assistant. He became associate lecturer in 2014. His current research interests include CUDA, MPI, OpenMP, Machine Learning, Computer Vision and Elastic Optical Network. His main research work on the master was focusing on face detection using graphic processing units (GPUs) and how to enhance the performance of the face detection algorithms on the GPU.

Samy Ghoniemyis a Professor of Computer Systems while he is the Vicedean for postgraduate studies and research, Faculty of Informatics and computerscience, the British University in Egypt (BUE). Dr. Samy obtained his B.Sc. inCommunications and Computer Engineering, and his M.Sc. and Ph.D. in Systems andComputer Engineering from Carleton University, Canada. Dr. Ghoniemy also hasextensive professional and consultancy experience in ICT. He worked and now aconsultant for National and multinational ICT private companies in Egypt. Dr.Ghoniemy’s methodological and applied research as well as a considerable portion ofhis applied and collaborative work addresses artificial intelligence, computer vision, realtime video tracking, medical image processing. In due time, he is working in the area ofcognitive science, semantic networks for cancer disease detection, dynamic graphs forpredicting and controlling epidemic and endemic diseases.

Ehab Abouzinadahis an Assistant Professor at Computing and Technology College and Director of High performance computing (HPC) center – King Abdul-Aziz University. He received his graduate degrees from the United States of America (USA) from Engineering School - George Mason University. A Ph.D in smart cybersecurity, Master degrees in Information Technology, and graduate certificate degrees in Information Security. He is Supporting AI research groups to recognize the benefit of the high-performance computer system on speeding up the data processing for AI models. Also, working in many researches that focus on Artificial intelligence – Cybersecurity and social media mining for building smart detection systems that identify cybercriminal accounts to improve security on Social media.

Hossam Faheem is a professor of computer systems at the faculty of computer and information sciences – Ain Shams University. He has a BSc of computer engineering in 1992, MSc of computer engineering in 1995, and PhD of computer engineering in 2000. His research interests include Parallel Processing, High Performance Computing, Networking, and Multi Agent Based Systems. His work focus on the hardware architecture optimization and system on chip design. He designed FPGA based pattern matching chips, DNA pattern matching co-processors, and other FPGA based algorithms for bioinformatics and computer vision algorithms. His current interests are about heterogenous architecture and meta scheduling strategies.