Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 87 (2016) 156 – 163

Procedia Computer Science

2016 International Conference on Computational Science

A GPU based implementation of Robust Face Detection System

Vaibhav Jain, Dinesh Patel

Dept. of Computer Engineering, Institute of Engineering and Technology, DAVV Indore, 452017, India

Abstract

Face detection is the active research area in the field of computer vision because it is the first step in various applications like face recognition, military intelligence and surveillance, human computer interaction etc. Face detection algorithms are computationally intensive, which makes it is difficult to perform face detection task in real-time. We can overcome the processing limitations of the face detection algorithms by offloading computation to the graphics processing unit (GPU) using NVIDIAs Compute Unified Device Architecture (CUDA). In this paper, we have developed a GPU based implementation of robust face detection based on Viola Jones face detection algorithm. To verify our work, we compared our implementation with traditional CPU implementation for same algorithm.

© 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Organizing Committee of ICRTCSE 2016

Keywords: Face Detection; GPU; CUDA; Integral Image;

1. Introduction

Biometric technology utilizes the biological characteristics of human bodies or behaviors as identification or verification features. The frequently used biometric features include face, fingerprint, voice, and iris recognition. The fingerprint recognition is the most popular adopted in our daily lives. However, the sweat and the dust may reduce the accuracy. In face processing system, it is not necessary to have physical contact with the machine and the image can be captured naturally by using a video camera. This makes face processing system a very convenient biometric identification approach. A face processing system comprises of face detection, recognition, tracking and rendering. Face detection is used to distinguish faces from the background. Face detection is the process of detecting faces in input images. Face detection in images is quite complicated and a time consuming problem. Face detection is important because it is the first step in various applications like face recognition, video surveillance, Human Computer Interaction etc.

The face plays a main role in carrying identity of persons. Face detection is one of the main biometric features that many works concentrate on developing algorithms to apply it in different systems. Traditionally expensive dedicated hardware was used to achieve the desired rate of detection. Even on current hardware, face

1877-0509 © 2016 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Organizing Committee of ICRTCSE 2016
doi:10.1016/j.procs.2016.05.142

detection is very time consuming, especially at the moment when large images are used. It is the same problem when we recognize faces in real time, for example from a camcorder. This is why the detection process must be accelerated. In last few years, graphic cards are increasing in performance; actually, the graphics processing unit (GPU) has greater performance than a classic central processing unit (CPU). Today, a graphic card can be used not only for rendering graphics, but it can also be used for general-purpose parallel computations, which are not connected with the original task of graphic cards-rendering. The first real-time face detection algorithm was proposed by Viola and Jones[1]. It has now become the de-facto standard for real-time face detection. However it doesn't suits well for images with high resolution, hence we need to look for high performance face detection solutions for fast face detection with reasonable cost. Parallelization is the best way to achieve faster face detection.

In our work, we have developed GPU based face detection system based on Viola Jones algorithm. To verify our work, we compared performance of our implementation with CPU based implementation at three stages of the algorithm i.e. image resizing, integral image calculation and cascade classification. We found that our GPU based implementation of Viola Jones face detection performed 5.41 to 19.75 times faster than its CPU implementation.

2. Related Work

Real-time object detection is an important work for many applications. One very robust and general approach to this work is using statistical classifiers that classify individual locations of the input image and make a binary decision: the location contains the object or it does not. Viola and Jones[1] presented very successful face detector which combines boosting, Haar low-level features computed on integral image and a consideration cascade of classifiers. The first real time face detection algorithm was proposed by Viola and Jones[4]. A lot of work is being done for accelerating the face detection process. Face detection algorithm using Haar like features was described by Viola and Jones[1] and R. Lienhart[5] a range of its modifications are widely spread in many applications. One of these modifications was implemented in OpenCV library[6]. The OpenCV implementation compiled with OpenMP option provides only 4.5 frames per second on 4-core CPU. It is too slow to process HD stream in real time. Some parallel versions of face detection algorithm using Haar-like features[678]. The algorithm introduced by Hefenbrock[6] was the first realization of a face detection algorithm using GPU we could find. It showed an effect of using GPU versus CPU. But the algorithm could not process a stream with resolution 640x480 in real time. The next parallel implementation is found in Obukhovs algorithm[7]. It is a single realization that uses GPU and can work with OpenCV classifiers without modification that is why modern versions of OpenCV library include it. The main problem of the algorithm is texture memory usage for classifier storing because texture memory is not as effective for general operation as cached global memory on modern GPU. The work of Jaromiret al.[8] described a GPU accelerated face detection implementation using CUDA. They compared their implementation of Viola and Jones algorithm to the basic one-thread CPU version. Some works are also written about acceleration object classification with some good results. As in illustration, Gao and Lu[9] reached a detection at 37 frames/sec for 1 classifier and 98 frames/sec for 16 classifiers using 256x192 image resolution. Kong et al.[10] proposed a GPU-based implementation for face detection system that enables 48 faces to be detected with a 197 ms latency. Heroutet al.[11] presented a GPU-based face detector based on local rank patterns as an alternative to the commonly used Haar wavelets[12]. Finally, Sharma et al.[13] presented a working CUDA implementation that affected a resolution of 1280x960 pixels. They proposed a parallel integral image to discharge both row wise and column-wise prefix sums, by fetching input data from the off-chip texture memory cached in each SM.

3. Face Detection Algorithm

We used Viola Jones Face Detection algorithm in our work. At a high level, the algorithm scans an image with a window looking for features of a human face. If enough of these features are found, then this particular window of the image is said to be a face. In order to account for different size faces, the window is scaled and the process is repeated. Each window scale progresses through the algorithm independently of the other scales. To reduce the number of features each window needs to check, each window is passed through stages. Early stages have fewer features to check and are easier to pass whereas later stages have more features and are more rigorous. At each stage, the calculations of features for that stage are accumulated and, if this accumulated value does not pass the threshold, the stage is failed and this window is considered not a face. This allows windows that look nothing like a face to not be overly scrutinized. To more thoroughly understand the algorithm, we can divide the algorithm into three stages based on the functionality. The three stages are feature extraction stage, integral image calculation stage and cascade classification stage. In feature extraction stage, feature classifiers are used to detect particular features of a face. Windows are continuously scanned for features, with the number of features depending on the particular stage the window is in. The features are represented as rectangles and the particular classifiers we use are composed of 2 and 3 rectangle features. Fig. 1 shows an example of such a feature classifier.

Fig. 1. Example of 2-Rectangle feature for Face Detection

To compute the value of a feature, we first compute the sum of all pixels contained in each of the rectangles making up the feature. Once calculated, each sum is multiplied by the corresponding rectangles weight and the result is accumulated for all the rectangles in the feature. If the accumulated value meets a threshold constraint, then the feature has been found in the window under consideration.

In integral image calculation stage, to avoid computing rectangle sums redundantly, we compute the Integral Image (II) as a pre-processing step. The Integral Image at location (x; y) contains the sum of the pixels above and to the left of (x; y). II(x-1, y-1) is subtracted off since it is included redundantly in the sum II(x-1, y) and II(x, y-1). Fig. 2 shows this pictorially.

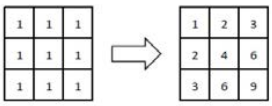

Fig. 2. Face image represented as Bitmap and Integral Image

Using the Integral Image, features can be calculated in constant time since we can compute the sum of the pixels in the constituent rectangles in constant time. Although the features can be calculated in constant time, excessive work would be done if a particular window region looks nothing like a face.

In cascade classification stage, the algorithm uses over 2000 features like eye region, upper-cheeks, nose bridge region etc. and it would be inefficient to calculate all of these features unnecessarily. To avoid this problem the algorithm performs cascade classification to divide up the number of features and eliminate windows quickly when it has been determined that they do not contain a face. Additionally, cascade keeps windows that look nothing like a face from being analyzed unnecessarily. It immediately labels a window as not a face when the window fails a particular stage. In general, earlier stages are passed more frequently with

later stages being more rigorous. Thus, the amount of work in each particular stage varies greatly. If all the features of this particular stage are found in the window, the stage is said to be passed and the window is propagated to the next stage and the window is again scanned for features of this next stage. If the window passes all stages, then it is said to be a face and the next window is then processed in the same manner.

4. GPU architecture and CUDA

The graphics processor with its massively parallel architecture is a storehouse of tremendous computing power. The Compute Unified Device Architecture (CUDA) is a C based programming model from NVIDIA that exposes the parallel capabilities of the GPU for easy development and deployment of general purpose computations. CPUs have few cores that are optimized to perform sequential computing while GPUs have thousands of cores which are specially designed for parallel processing. So a significant speedup can be achieved by executing high computational work on GPU while rest of code in CPU. Researchers have used GPU computing to accelerate various engineering and scientific problems. Moreover, pixel-based applications such as computer vision and video and image processing are very well suited to general-purpose GPU technology. A CUDA capable GPU consists of a set of streaming multiprocessors (SMs). Each streaming multiprocessor has a number of processor cores. A streaming multiprocessor processor core is known as streaming processor (SP). The number of streaming processors each streaming multiprocessor contains depends on the GPU. Generally, in modern GPU each streaming multiprocessor contains 32 streaming processors. So if a GPU has 512 cores that mean it contains 16 streaming multiprocessors each containing 32 cores or streaming processors. The programs running on GPU are independent of architectural differences which make GPU programming scalable.

Compute Unified Device Architecture (CUDA) was introduced by NVIDIA in 2007. This framework gives programmers access to the virtual instruction sets and memory of parallel processing units in an NVIDIA GPU. Instead of using graphical API instructions, a program written in C/C++ code is directed to a specialized hardware in the GPU and that hardware manages the execution of that program on the GPU. The CUDA framework is actually an extension to the C programming language. The compiler that is responsible for compiling CUDA code is NVCC. When C/C++ code is given as the input code for this compiler, it first analyses the code and separates the conventional C/C++ code and CUDA C code. The regular C/C++ code is compiled using the systems primary C compiler (GCC etc.) but CUDA C portions are compiled using NVCC. The CUDA framework is actually an extension to the C programming language. The compiler that is responsible for compiling CUDA code is NVCC.

5. Proposed GPU implementation of Face Detection

Our proposed work is based on Viola Jones Algorithm for face detection. The Fig. 3 shows the proposed plan for implementation of face detection system. The face detection system functionality has two implementation 1) CPU implementation 2) CPU and GPU implementation. In CPU implementation part all functionality of face detection system is implemented by using single thread program. In CPU and GPU implementation part some functionality has been implemented using CPU (host) and most of functionalities have been implemented using GPU (device) with data parallelization. Our proposed architecture show that image transformation and cascade classifier functionalities of Viola Jones algorithm can be implemented both CPU and GPU. The tasks like image read and generating rectangles on detected faces are done on CPU. Our GPU based Face detection implementation comprises of three main steps: 1) resizing of the original image into a pyramid of images at different scales 2) calculating the integral images for fast feature evaluation, and 3) detecting faces using a cascade of classifiers. Each of these tasks is parallelized and run as kernels on the GPU.

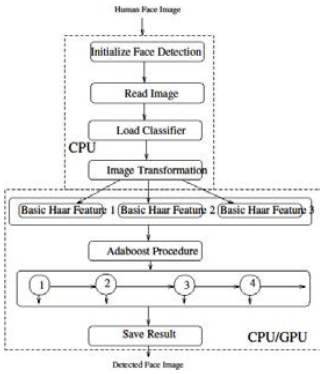

Fig. 3. Proposed Face Detection System Architecture

Following is the pseudo code for our GPU based face detection implementation.

1: for number of scales in image pyramid do [*used single thread for each image pyramid*]
2: down sample image by one scale;
3: compute integral image for current scale; [*used horizontal and vertical accumulation*]
4: for each shift step of the sliding detection window do
5: for each stage in the cascade classifier do [*used single thread for each classifier*]
6: for each filter in the stage do
7: filter the detection window;
8: end
9: accumulate filter outputs within this stage;
10: if accumulation fails to pass per-stage threshold do
11: break the for loop and reject this window as a face;
12: end
13: end
14: if this detection window passes all per-stage thresholds do
15: accept this window as a face;
16: else
17: reject this window as a face;
18: end
19: end
20: end

In the above pseudo code, image resizing is performed by line 2; line 3 is corresponding to integral image calculation. Cascade detection part is performed by line 4-20.More details about above pseudo code are explained in subsequent subsections.

In image resizing stage, the original image is resized to a pyramid of images at different scales, the bottom of the pyramid being the original image, and the top, a scaled down image at 24x24 resolution, which is the base resolution of the detector. The height of the pyramid, or in other words, the number of resized images depends upon the scaling factor which is 1.2 in our case. Computation of the pyramid of images, though straightforward, requires significant time. A simple approach for parallel image resizing is by allowing different CUDA thread blocks to compute images at different scales in parallel. Each thread in a thread block, computes the value of a pixel in an image scale. However, since CUDA thread blocks have fixed dimensions,

as the image dimensions progressively decrease, larger number of threads are rendered inactive in this approach as shown in Fig. 4.

Fig. 4. Pyramid of Images

In integral image stage, the algorithm uses Adaboost machine learning algorithm for accurate and fast face detection. The algorithm uses to pick up the most promising feature from over complete set of Haar feature to recognizes it is a face or not. It uses the frame detection window of size 24 x 24. Now these features are applied on to image to calculate the sum of all pixel values under dark region subtracted from sum of all pixel values under bright region. Though sum of pixels approach is considered primitive in comparison to other sophisticated methods, but the integration of pixels allows for faster detection and its accuracy almost comparable to other techniques. The principle of counting is divided into two parts. Using the integral image we can calculate rectangle sum easily in 4 value access. We can calculate the image integration GPU using vertical prefix or by horizontal prefix as shown in Fig. 5 for each thread calculate sum row or column wise.

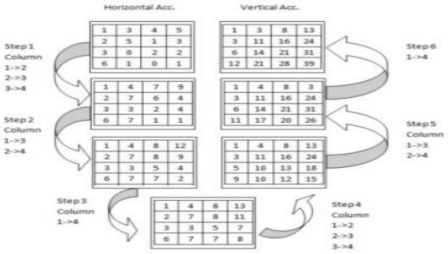

Fig. 5. Integral Image Calculation on GPU

Cascade detection stage brings some improvements in face detection time. It is based on principle that there are more areas in an image that do not contain part of a face. That is why it is not necessary to test all classifiers. Viola and Jones introduced weak classifier and strong classifier to solve this problem. The weak classifier decided as threshold value assigned to them. We parallelize the cascaded detection process, by allowing the simultaneous computation of the feature values and scores for sub-windows at different locations of the image at different scales in parallel by multiple threads. This is depicted in Fig. 6, where two threads are shown, thread 0 and thread 1, which extract sub-windows at different locations and compute the score. For fast feature evaluation, the integral images computed previously are used. Both the integral images and the features are stored and retrieved from textures to enhance performance. The cascades are initially stored in textures and transferred to shared memory for faster access. Combining these classifier gives to form a strong classifier gives probability of sub window 24 x 24 has a face or not. The image of size 960 x 640 has 43 detection sub window.

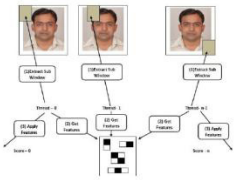

Fig. 6. Parallel Cascade Detection

6. Experimental Setup and Results

For the experiment purpose, the developed face detection system (a mixture of C++ and CUDA) has been tested on an Intel(R) Core(TM) i5 4210U CPU, 1.70 GHz host system with 4 GB RAM, having a NVIDIA GeForce 820M GPU. This GPU features 2 multiprocessors, 49152 bytes shared memory and 2 GB device memory. There can be a maximum of 1024 threads per block and 1536 active threads per multiprocessor. For comparison purposes, a CPU version of the face detection based on Viola Jones algorithm was also developed (single-threaded) for execution on the host CPU. Then this program is compared with a GPU program. Image Resizing handled by parallel threads, the Fig. 7 presents the results. It shows how the time needed for the computation depends on the image size. For test the image transformation time we take different size of images range from 10 kb to 4200 kb. The CPU implementation gave image transformation time from 0.349ms to 44.13ms and for GPU implementation from 0.047ms to 2.316ms. From the results we can see, that time computation is lowest for the GPU implementation, while the CPU program is significantly slower.

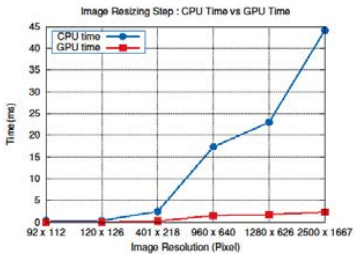

Fig. 7. Comparison of GPU and CPU Image Resize Time

The integral image is also computed by parallel threads as column and row wise. For testing, six different image sizes were chosen: 92 x 112, 120 x 126,401 x 218, 960 x 640, 1280 x 626 and 2500 x 1667 pixels. Fig. 8 shows the results of integral image computation time for CPU and GPU. The CPU computation time is ranging from 4.327 to 10.76 but GPU computation time is ranging from 0.645 to 0.792.Face detection is the major function of the algorithm it takes much time from previous process. In this we measure time taken to detect faces from an Image. Cascade classifier handle in multiple stages we assign one thread for each stage. For testing purpose we use different size of images as mentioned in previous.

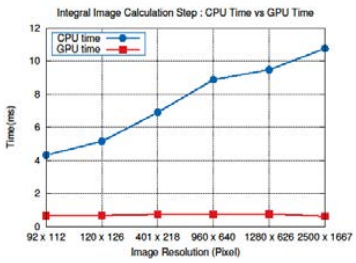

Fig. 8. Comparison of GPU and CPU Integral Image Conversion Time

The following Fig. 9 shows the performance results of the cascade detection function. For CPU program it takes 22.86ms,28.55 ms,175.90 ms,1227.40 ms,1595.02 ms and 18.644 ms. The GPU implementation takes only 4.22 ms,4.26 ms,13.37 ms,63.79 ms,82.16 ms and 312.66 ms and 0.530 ms.

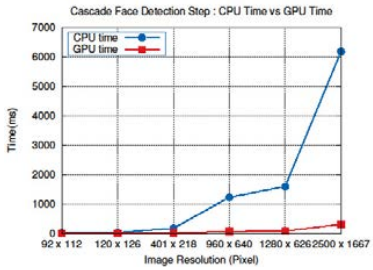

Fig. 9. Comparison of GPU and CPU cascade face detection Time

7. Conclusion & Future Work

Face detection plays an important role in security surveillance systems as we are planning for future smart cities. With the introduction of general purpose GPU programming, we can exploit parallelism to a greater extent for high computing tasks like face detection. In our work to check the efficiency of our GPU based implementation we took images at various scales (92 x 112, 120 x 126,401 x 218, 960 x 640, 1280 x 626 and 2500 x 1667) with different sizes and also varied number of faces to analyze performance with CPU based implementation. We found that our GPU based implementation performed 5.41 to 19.75times faster than the CPU version and scales much better even at higher resolutions across image resizing, integral image calculation and cascade classification stages. As future work, we are planning to work for images with side pose. We also feel that, there is a need to incorporate some new features for side pose estimation in proposed algorithm and we plan to extend the concepts discussed in this paper to face recognition.

References

[1] P. Viola, M.J. Jones. Rapid object detection using a boosted cascade of simple features. In: CVPR '01: *Proceedings of the Conference on Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA; 2001, p. 511-518

[2] Y. Freund, R. E. Schapire. A short introduction to boosting.In:Journal of Japanese Society of AI, Japan; 1999, p. 771-780.

[3] R. Farber. CUDA Application Design and Development. In: Massachusetts, Morgan Kaufmann; 2011, p. 2-16.

[4] Paul Viola, Michael J. Jones. Robust real-time face detection. In:Int. Journal of Computer Vision, Netherland; 2004, p. 137-154.

[5] R. Lienhart. An extended set of Haar-like features for rapid object detection. In: Proceedings of IEEE International Conference on Image Processing (ICIP'02), USA; 2002, p. 900-903.

[6] D. Hefenbrock et al. Accelerating Viola-Jones Face Detection to FPGA-Level Using GPUs. In: FCCM '10: *Proceedings of 18th IEEE Annual Int. Symposium on Field-Programmable Custom Computing Machines*, Charlotte, NC; 2010, p. 11-18.

[7] Anton Obukhov. Haar Classifiers for Object Detection with CUDA. In: GPU Computing Gems. Emerald Edition; 2011, p. 517-544.

[8] J. Krpec, M. Nemec. Face detection CUDA accelerating. In: ACHI '2012:*The Fifth International Conference on Advances in computer Human Interactions*, Valencia, Spain; 2012, p. 155-160.

[9] C. Gao, S.L. Lu. Novel FPGA based Haar classifier face detection algorithm acceleration. In: *18th International Conference on Field Programmable Logic and Applications*, Heidelberg, Germany; 2008, p. 373-378.

[10] J. Kong, Y. Deng, GPU accelerated face detection. In: *International Conference on Intelligent Control and Information Processing*, Dalian,China; 2010, p. 584-588.

[11] A. Herout, R. Josth, R. Juranek, J. Havel, M. Hradis, P. Zemcik. Real-time object detection on CUDA. In: *Journal of Real-Time Image Processing*, Verlag, Germany; 2011, p. 159-170.

[12] M. Hradis, A. Herout, P. Zemcik. Local rank patterns: novel features for rapid object detection. In: ICCVG '2008: *International Conference on Computer Vision and Graphics*, Warsaw, Poland; 2008, p. 239-248.

[13] B. Sharma, R. Thota, N. Vydyanathan, A. Kale. Towards a robust real-time face processing system using CUDA-enabled GPUs. In: HIPC'2009: *IEEE 18th International Conference on High Performance Computing*, Kochi, India; 2009, p.368-377.

[14] Shivashankar J. Bhutekar, Arati K. Manjaramkar. Parallel face Detection and Recognition on GPU. In: International Journal of Computer Science and Information Technologies, vol. 5; 2014, p. 2013-2018.