Team Red Ninjas – Project Proposal for ECE759

Efficient GPGPU Implementation of Viola-Jones Classifier Based Face Detection Algorithm

Sharmila Shridhar Ramsai Manoj Bamdhamravuri Vinay Gangadhar
{sshridhar, bamdhamravur, gangadhar}@wisc.edu

Abstract

Applications with large amount of data level parallelism can benefit from General Purpose based Graphics Processing Units (GPGPUs) because of better energy efficiency and performance compared to a CPU. Due to GPGPUs' impressive computing throughput and memory bandwidth, many applications with enough parallelism can take advantage of acceleration using GPGPU. Computer vision algorithms are one such workload domain which can better utilize the large number of cores in GPU, and hence meet their real-time requirements of the applications. One such application is Face Detection which has real-time constraint on its execution. Sequential processing of image windows with classifiers on CPU is difficult to meet the real-time requirements. In this project, we implement the Face Detection acceleration algorithm based on Viola-Jones cascade classifier the GPGPU CUDA platform. We are considering different portions of the algorithm that can be parallelized and expect to achieve better speedup compared. We feed an image as input to the GPU, and after processing set of features and classifying the face using a cascade classifier, a rectangle is drawn on the image when face is detected. We finally compare the execution of the same algorithm executed on the CPU and analyze the gains from the GPU.

1. Introduction

With recent advances in computing technology, and new computing capabilities with GPGPU hardware [6] and programming models [5], a trend of mapping many image processing applications on GPU is seen. Compute Unified Device Architecture (CUDA) has enabled mapping generic parallel implementations of image processing algorithms [11] easier and benefits with good amount of performance and energy efficiency. Face detection is one such application which has lots of fine-grained data parallelism available and exploit the execution resources of GPU. It is processing of taking an image and detecting and locating the faces in the given image. It is an important application used world-wide for public security, airports, video conferencing and video surveillance.

Most of face detection systems today use cascade classifier algorithm based on Viola and Jones [9]. It has three important concepts tied to it – integral image calculation, Adaboost classifier training algorithm [3] and cascade classifier. Although, many of them have implemented these algorithm in CPUs, due to the inherent serial nature of CPU execution, you cannot get much of the performance benefit and may not be able to meet hard real-time constraints, even when executed on a multi-core CPU. With face detection algorithm's inherent parallel characteristics, GPGPU parallel computing substrate is a good candidate to gain performance benefits. With recent advances in NVIDIA CUDA programming model for scientific application acceleration [1], we aim to use GPGPU execution model for accelerating face detection algorithm.

In this project, we intend to implement face detection algorithm based on the Viola Jones classifier on GPU. As a starting point, we take the GNU licensed C++ program that has the algorithm implemented to detect faces in images. There are various portions in the algorithm that can be parallelized and hence can leverage the execution of GPU resources efficiently. We plan to implement all three phases of the face detection – integral image calculation, scanning window stage and classifying the output from each classifying stage. As a course project we want to limit to the implementation of these 3 stages mentioned above, and not focus on the training of classifier itself. We take some insights and principles based on previous implementations of face detection on GPGPUs, FPGA done here [4, 8, 2]. The focus of this project is to gain performance benefits out of face detection acceleration and characterize the bottlenecks if there are any.

Section 2 explains the Viola Jones algorithm briefly. Section 5 explains the project phases and work distribution. Section 3 explains the evaluation methodology we consider for comparison and also explain the portions we are going to parallelize and offload to the GPU.

2. Viola Jones Face Detection Algorithm

The algorithm has four stages as described below:

- **Haar Feature Selection:** The Viola Jones classifier method is based on Haar-like features. The features consist of white and black rectangles as shown in Figure 1. These features can be thought of as pixel intensity evaluation sets. For each feature, we subtract black region's pixel value sum from white region's pixel value sum. If this sum is greater than some threshold, it is decided that the region has this feature. This is the characteristic value of a feature. We have the Haar features to be used for face detection.

- **Integral Image Calculation:** The calculation of character-

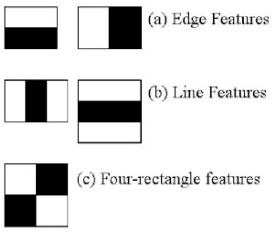

Figure 1: Five kinds of HAAR feature rectangles

istic value of a feature is an important step in face detection. It has to be calculated for every possible pixel region in the given image. In order to efficiently determine the value, integral image of the given image is used. For any given pixel *(x, y)*, the integral value is the sum of all the pixels above and to the left of *(x, y)*, inclusive. i.e., If *v(x', y')* is the value of a pixel at *(x', y')*, then the Integral Sum is given by:

$$IS = \sum_{x' \leq x, y' \leq y} v(x', y')$$

Now, the sum of any given region with corner integral pixels *A, B, C, D* is *Sum = D + A - B - C* as shown in Figure 2.

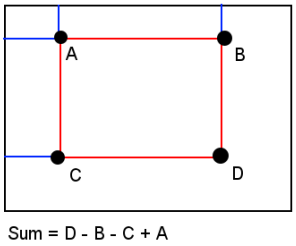

Figure 2: Sum of a sub-window using Integral Image

As shown in Figure 3, if we want to obtain the sum of 2x3 sub-window, integral image involves just 2 additions and 2 subtractions instead of 6 additions.

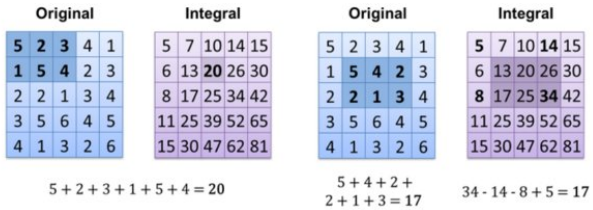

Figure 3: Example of Integral Image calculation

- **Adaboost Algorithm:** Adaboost or Adaptive Boosting is a machine learning algorithm where the output of weak classifiers is combined into a weighted sum that represents the output of a boosted (strong) classifier. This algorithm is used to combine features that cover facial characteristics to form a weak classifier. It then creates a string classifier using a group of weak classifiers. It further concatenates the strong classifiers to a cascade classifier.

- **Cascade Classifier:** A sub-window in the image that passes through the entire cascade classifier is detected as human face (Figure 4). In this project, we use 25 stages in cascade classifier.

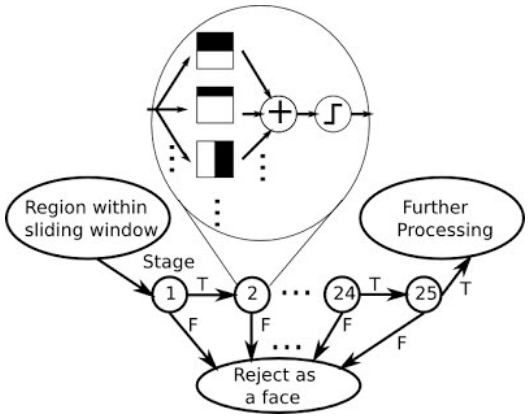

Figure 4: Cascade Classifier

3. Methodology

In this project, we consider the sections in the algorithm that can be parallelized, write kernels for each of them in CUDA and offload them to be executed on GPU. The sections that can be parallelized are explained below.

- **Image Pyramid:** We use fixed size (20x20) for each feature extracted. But the face in the image can be of any scale. To make the face detection scale invariant, we use pyramid of images that contains upscale and downscale versions of the original image. Since each pixel can be processed separately, we intend to produce image pyramid on GPU. Example of an image pyramid is shown in Figure 6.

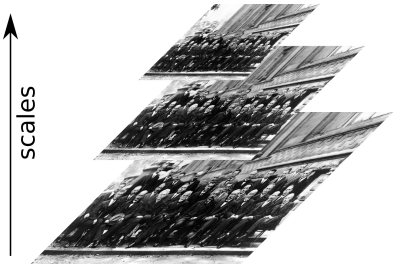

Figure 5: Image Pyramid

- **Integral image calculation:** Integral image calculation is essentially a 2-dimensional inclusive scan. Since we have already done 1-dimensional prefix scan on GPU and have seen the benefits, we intend to determine the integral image on similar lines. To avoid data dependency of image integral calculation, we adopt the algorithm of first row integral and then column integral calculation.

- **Scan Window Processing:** Since each 20x20 image sub-window has to pass through all the features, each thread

in the GPU can execute on a sub-window in parallel. But the design of cascade classifier is such that after each stage the doubtless non-face scan sub-windows are eliminated as much as possible. Hence we want to consider each stage separately and sequentially, may be execute each stage as a different kernel.

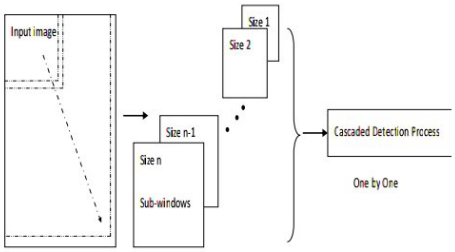

Fig. 4. Serial detection.

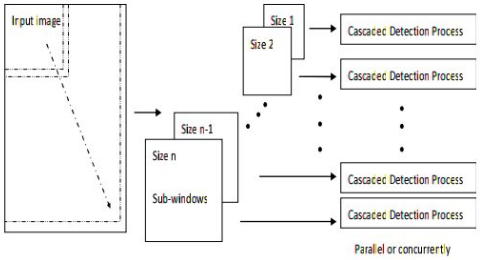

Figure 6: Parallel Scan Window Processing

4. Evaluation framework

For this project, we evaluate the performance of GPGPU implementation with that of a CPU. We are not considering the classifier training and are directly taking a CPU version of trained classifier. Haar features, number of features in each stage are already determined. We aim to compare the execution time for face detection in an image on CPU and on GPU. As mentioned in Section 3, we parallelize three different portions in the algorithm. We will analyze the contribution of each of the three sections parallelized for performance implications and try to characterize the bottlenecks associated with each sections of the algorithm. Since sufficient parallelism can be extracted in this image processing application, we hope to get better performance on GPU.

For profiling and bottleneck analysis, we aim to use NVIDIA Visual Profiler [7] and then take corresponding decisions for performance boost. For evaluating against CPU, we plan to use the Euler supported multi-core processor node. We use CUDA events [10] for timing analysis of GPGPU and CPU execution. We also plan to report the memory bandwidth usage for execution on both platforms.

5. Project phases

We explain the phases of project here and expected timeline for each step. Some of the phases can be carried out in parallel and will be distributed among the project members.

Figure 7 details the timeline of our project and work distribution among the team members.

TIMELINE	November 16 - 25		November 26 - December 10			December 10 - 12
Tentative task allocation	*Source code analysis & algorithm brainstorming*	*Output demo ideas*	*Image pyramid implementation*	*Integral image calculation*	*Cascade classifier processing*	*Output interfaces to be scripted (if any)*
Sharmila	✔	✔	✔	✔	✔	
Vinay	✔		✔	✔	✔	✔
Manoj	✔	✔		✔	✔	✔

Figure 7: Project Timeline Sheet

References

[1] I. Buck, "Gpu computing with nvidia cuda," in *ACM SIGGRAPH*, vol. 7, 2007.

[2] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, pp. 103–112.

[3] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[4] J. Kong and Y. Deng, "Gpu accelerated face detection," in *Intelligent Control and Information Processing (ICICIP), 2010 International Conference on*. IEEE, 2010, pp. 584–588.

[5] C. Nvidia, "Compute unified device architecture programming guide," 2007.

[6] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[7] P. Rosen, "A visual approach to investigating shared and global memory behavior of cuda kernels," in *Computer Graphics Forum*, vol. 32, no. 3pt2. Wiley Online Library, 2013, pp. 161–170.

[8] L.-c. Sun, S.-b. Zhang, X.-t. Cheng, and M. Zhang, "Acceleration algorithm for cuda-based face detection," in *Signal Processing, Communication and Computing (ICSPCC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–5.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511.

[10] N. Wilt, *The cuda handbook: A comprehensive guide to gpu programming*. Pearson Education, 2013.

[11] Z. Yang, Y. Zhu, and Y. Pu, "Parallel image processing based on cuda," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 3. IEEE, 2008, pp. 198–201.