Face Detection with Improved Local Binary Patterns in CUDA

March 25, 2013

Jeaff Wang
jeaff.wang@utoronto.ca

Richard Abrich
richard.abrich@utoronto.ca

Abstract

As mobile computing and user interactivity become more ubiquitous, accurate and fast facial detection mechanisms are necessary. And with the development of accessible parallel computing, it becomes possible to leverage the power of parallel algorithms to increase both speed and accuracy of facial detection systems. In this paper, we propose and analyse one such system based on the Improved Local Binary Pattern.

1 Introduction

1.1 Motivation

With the rapid proliferation of computer technology in recent decades, face detection using computer vision techniques has proven its utility in many applications, ranging from video surveillance to authentication and beyond. Simultaneously, low power devices such as mobile phones have become increasingly important in our daily lives. Many applications on such low power devices require face detection to be processed in real-time.

Processing speed has always been an important performance criterion for evaluating face detection algorithms. To illustrate, consider that real-time face detection in video sequences typically requires more than 15 face detections to be processed per second. [2] As such, a real-time face detection system capable of running on low-power platforms is necessary.

With the recent popularization of general purpose GPU (GPGPU) programming, the parallelization of algorithms for data-intensive workloads is more accessible than ever. For example, the face detection technique considered in this project requires on the order of 10^{10} simple arithmetic calculations, and lends itself well to parallelization. Therefore, the goal for this project is to increase the face detection processing speed by parallelizing the face detection algorithm in CUDA.

1.2 Problem Definition

The face detection system considered in this project is depicted in Figure 1.

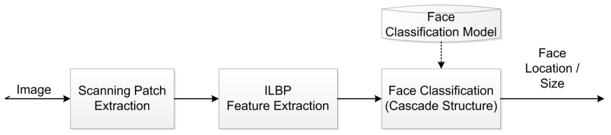

Figure 1: Block diagram of face detection system

The input to the face detection system is a gray-scale digital image in 8-bit unsigned integer format, and the output is the position and size of a box containing a detected face, as illustrated in Figure 2.

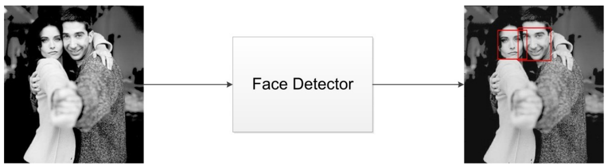

Figure 2: Input and output of face detection system

As shown in Figure 1, there are three main components involved in this face detection system: Patch Extraction, ILBP Feature Extraction, and Face Classification. In the Patch Extraction stage, patches in various positions and of varying sizes are extracted from the image. During the ILBP Feature Extraction stage, the ILBP feature is calculated for each of the patches. In the final stage, each patch is classified according to the extracted features as either containing a face or not, using a pre-trained face classification model. The face classification model has been previously trained using AdaBoost, a machine learning algorithm.

The bulk of the processing in this scheme is spent during the first two stages, and the third stage involves significantly more complex computations than the first two. Therefore, in order to maximize speedup, only the first two stages will be considered for the rest of this paper.

1.2.1 Patch Extraction

In this stage, square patches in various positions of varying sizes within the input image are considered, as depicted in Figure 3. Since the face classifier was trained using images measuring 24×24 pixels, each extracted patch is resized to 24×24 pixels using nearest neighbour interpolation. The nearest neighbour algorithm was chosen so as to reduce the computation complexity. However, other interpolation methods, such as linear interpolation and bi-cubical interpolation, can also be applied for potentially higher accuracy.

For a 300×300 pixel image, with typical translation and resizing step sizes, there are

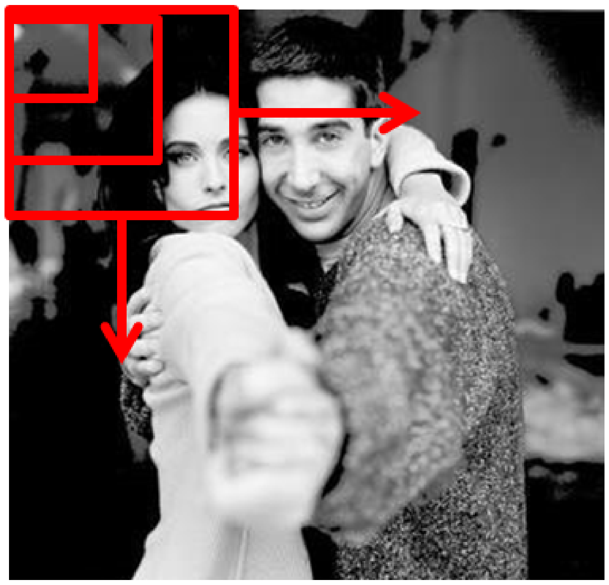

Figure 3: Patch extraction

approximately $65,000$ patches to be extracted. The same operations (extracting and resizing) are applied to each of these patches.

1.2.2 ILBP Feature Extraction

Feature extraction is applied to reduce the dimensionality of the input data and to extract unique characteristics of face images for classification. In particular, the Improved Local Binary Pattern (ILBP) feature is considered for this project due to its promising performance and relatively simple calculations. [1]

The ILBP feature, shown in Figure 4, is calculated as follows: for a given patch, a centre location is first determined. The centre location can be any possible pixel in the patch (except for at the edges). Around every centre pixel, a 3x3 neighbourhood is extracted. The average value of this neighbourhood is calculated, and a 3x3 binary map is generated, in which each element has either a value of 1 if its corresponding grayscale value is greater than the average, and 0 otherwise. The binary map is then multiplied by a previously learned weighting matrix and thresholded again, and then converted into an integer by concatenating the elements of the map together, starting from the top left element and moving clockwise.

For every 24×24 window, there are typically about 200 ILBP features (200 pixel locations) to be extracted for face classification. In total, there are about 13 million feature extraction operations.

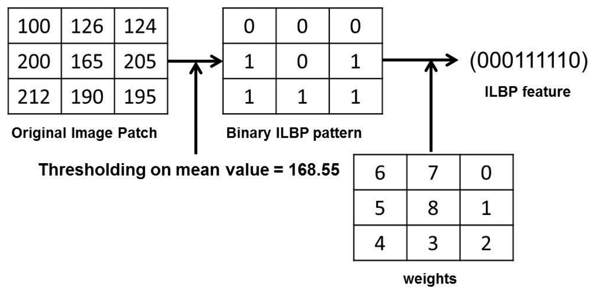

Figure 4: ILBP feature extraction

1.3 Objective

For typical face detection on a 300×300 pixel image, there are about $65,000$ patches to be extracted and resized, and around 200 ILBP features to be extracted from each patch. In total, there are about 10^10 calculations that must be performed. The goal of this project is to decrease the end-to-end execution time of patch extraction and feature extraction by implementing the corresponding algorithms in CUDA.

The performance (execution time) is evaluated and compared between the CUDA and ANSI C implementations of patch extraction and feature extraction. In addition, the performance is also evaluated on different image sizes and for different numbers of features.

2 Methodology

In this project, the patch extraction and feature extraction stages were implemented and tested independently, and then combined in order to test the overall system performance. Both stages were implemented in both CUDA and ANSI C with a Matlab interface.

2.1 Patch Extraction

In the patch extraction phase, patches are extracted by sequentially iterating over all of the different patch sizes, where a single kernel launch extracts all patches of a given size. The CUDA grid size is $M \times N$ blocks, where M and N are the number of patches along the horizontal and vertical dimensions of the image, respectively (see Figure 5). Each block is composed of 24×24 threads, where each thread is responsible for saving a single pixel of the resized patch using nearest neighbour interpolation. These patches are saved into global memory, and remain there for the subsequent feature extraction stage. Since every thread is responsible for exactly one pixel, and each pixel is saved sequentially in memory, this scheme does not suffer from any control flow divergence or bank conflicts.

This scheme is extensible to features trained on patches of up to 32×32 pixels, due to CUDA's maximum of 1024 threads per block.

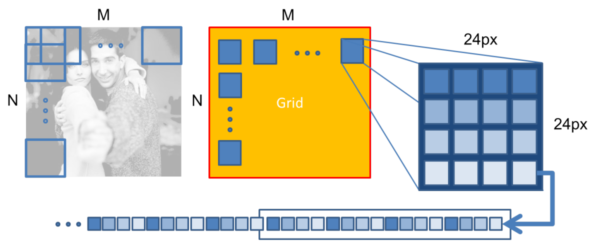

Figure 5: Patch extraction kernel behaviour

The nearest neighbour interpolation is achieved as follows: given an input image of dimensions $N \times N$, and a desired output dimension of size 24×24, the column and row indexes into the output image are given by the values of `threadIdx.x` and `threadIdx.y` respectively, while the corresponding indexes into the input image are calculated as follows:

$$y_{resized} = y_{original}\frac{24}{patchSize}$$
$$x_{resized} = x_{original}\frac{24}{patchSize}$$

The "nearest-neighbour" behaviour is achieved by the compiler rounding the division to an integral value. If the input image is greater than 24×24, extraneous pixels are simply discarded, whereas if it is smaller than 24×24, pixels in non-integral positions are simply assigned the value of their nearest neighbour (see Figure 6).

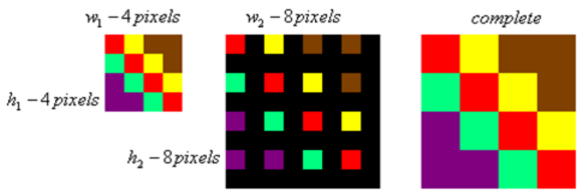

Figure 6: Nearest neighbour interpolation

2.2 Feature Extraction

In the Feature Extraction stage, the partitioning of threads is assigned as follows. Due to the nature of ILBP feature extraction problem, each 1D block is assigned to extract an ILBP feature of every 3x3 neighbourhood. Therefore, each 1D block requires 9 active threads. However, in the interest of memory alignment during the reduction step, 16 threads are assigned to each block, with 7 additional threads for zero-padding. The 2D grid is structured such that each row corresponds to a single patch and each column corresponds to an ILBP feature at a particular centre location. The advantage of this arrangement is that the extracted patches can be processed directly without additional memory mapping. However, the disadvantage is that the total number of patches is limited due to the limitation on the grid size ($65,535$ blocks), thereby limiting the size of the input image to approximately 300×300 pixels. Nevertheless, this limitation is not a problem for the images of interest in this project.

Since the 24×24 patches extracted from the previous step were saved in global memory, no memory transfer from host to device is required in this step. Furthermore, shared memory is used to further decrease calculation latencies: prior to calculation, each 3×3 neighbourhood is first extracted and saved in shared memory. Since each 24×24 patch is saved in the same order, a hard-coded mapping (Figure 7) is applied to ensure values are copied from the correct location.

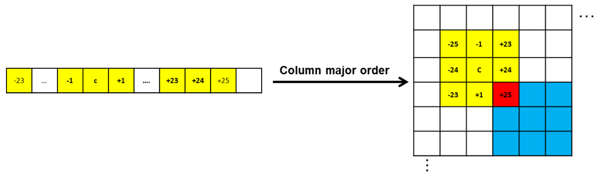

Figure 7: ILBP 3×3 neighbourhood extraction mapping

According to the ILBP feature extraction shown in Figure 4, three different arithmetic operations are required for every 3×3 neighbourhood. The first operation is to calculate the mean of the pixels within the neighbourhood. The second operation is to generate the binary map by applying the mean value threshold each pixel within the neighbourhood. Lastly, the binary map is weighted and summed together to produce the ILBP code.

The first and last operations can be replaced by a summation operation, which is achieved through a reduction operation. The reduction algorithm is implemented to avoid high divergence and bank conflicts: high divergence is avoided by stride indexing, and bank conflicts are avoided by partitioning the entire sequence into two halves. Thresholding is achieved by a comparison operation.

Finally, the calculated ILBP code is copied from shared memory to global memory, and eventually copied from device memory back to host memory.

2.3 Evaluation

The evaluation of the CUDA versus ANSI C implementation of the face detection system is as follows. While the original size of the input image is 300×300 pixels, downscaling of the original image is applied to test the performance of the algorithm with different image sizes. Patch extraction is performed with patches starting at 24×24 pixels large, and increasing until only a single patch can fit within the image, with increments of 1.125%. For each patch size, the horizontal and vertical translations of the patch are 5% and 10% of the patch size, respectively.

For each configuration, the execution time is recorded using `Win32::kernel32::QueryPerformanceCounter()` and `Win32::kernel32::QueryPerformanceFrequency()`. The two stages (patch extraction and feature extraction) are evaluated independently, as well as together as a system. The test results are presented in the following section.

The specifications of the hardware used for testing are listed in Table 1. Both the CPU and GPU were high-end devices purchased simultaneously approximately one year ago, thereby providing a roughly equal comparison.

3 Results & Analysis

The results of the Patch Extraction, ILBP Feature Extraction, and of the overall system are presented in the following sub-sections, respectively.

3.1 Patch Extraction

For this test, the GPU and CPU implementations of the patch extraction algorithm are evaluated as a function of the input image size.

During the Patch Extraction stage, the execution time on the CPU increases exponentially as a function of the image size (see Figure 8). This is because for a larger image size, not only are there more patches of a given size, but there are also a greater number of different patch sizes that can fit into the image.

In comparison, the execution time on the GPU grows at a markedly slower rate. In fact, from the collected data, the relationship between GPU execution time and image size is almost linear. This linear relationship between execution time and image size is attributable to parallelism.

Upon closer inspection, the execution time appears to increase in steps. This corresponds with the discrete increases in the grid size that would be necessary for continuously increasing image dimensions, while the block size remains unchanged.

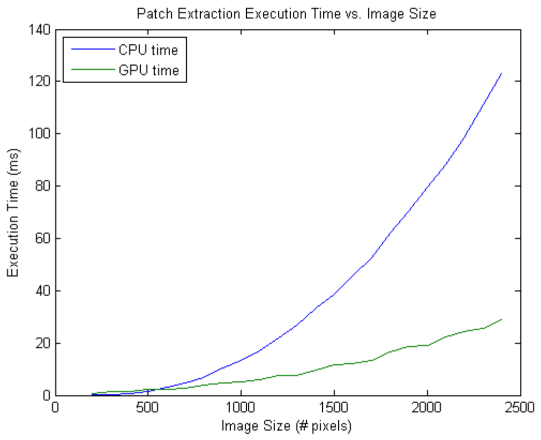

Figure 8: Patch Extraction execution time as a function of image size.

3.2 ILBP Feature Extraction

For this test, three different ILBP feature extraction implementations are compared: ANSI C (CPUtime), CUDA with global memory (GPUtime) and CUDA with shared memory (GPUtime shared). The execution times of

Table 1: Testbed specifications

	CPU	GPU
Processor	AMD Phenom II	NVIDIA GeForce GTX470
Clock Speed	2.8 GHz	1.215 GHz
# Cores	6	448
Memory	4 GB	1.25 GB

the three different implementations are plotted against the number of ILBP features extracted.

As expected, the plot in Figure 9 shows a linear relationship between the execution time and the number of features. The CUDA implementations show improved performance over the ANSI C implementation in both cases. Furthermore, using shared memory demonstrates significantly better performance than using global memory only. In fact, the execution time difference between shared memory and global memory is larger than the difference between the global memory and C implementations.

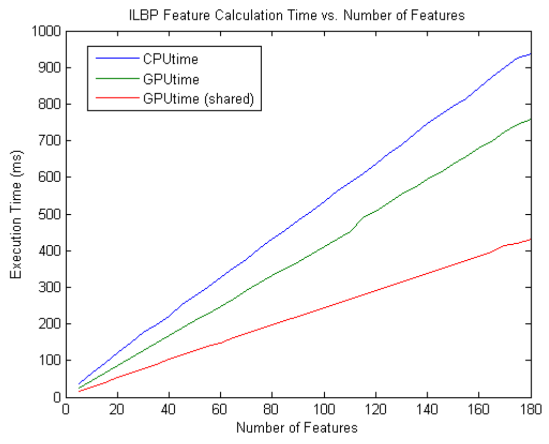

Figure 9: Feature Calculation execution time as a function of the number of features.

3.3 Overall System

The overall system test evaluates the performance of the combination of both stages. The plot in Figure 10 shows the execution time for the CUDA implementation with shared memory (GPU time) and the ANSI C implementation (CPU time) against different image sizes, while holding the number of features constant. An exponential relationship is expected since, as previously noted, increasing the image size increases the number of patches exponentially.

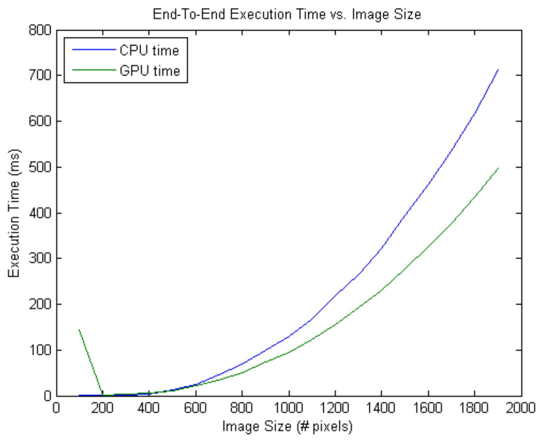

Figure 10: Overall execution time as a function of image size.

The CUDA implementation shows better overall performance than the ANSI C implementation. However, unlike the patch extraction alone, the overall system performance for the CUDA implementation also shows an exponential relationship with the image size. While the exact cause of this is unclear, the CUDA implementation of the feature extraction clearly requires more processing time than the detection window extraction. Therefore, the overall execution time is limited by the Feature Extraction stage.

The overall system execution time is also

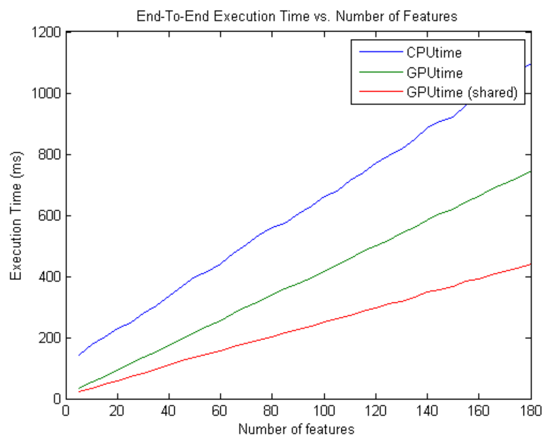

Figure 11: Overall execution time as a function of the number of features.

evaluated against different number of features extracted (see Figure 11). Since the Feature Extraction stage is the bottleneck of the system, this plot shows a similar trend as the Feature Extraction results alone. Nevertheless, the CUDA implementations show faster execution time than the ANSI C implementation, and shared memory shows better performance than global memory only.

4 Conclusion

In this paper, we implemented two stages of a face detection system using improved local binary patterns in CUDA. By varying the input image size and the number of features, we demonstrated that this system outperforms a similar algorithm implemented on a CPU. Our results indicate that the main bottleneck in calculations is the feature calculation, which has an exponential complexity in the number of pixels.

A potential area for future improvement is the utilization of texture memory for linear interpolation, which may lead to increased performance, or the use of bicubic or other interpolation for increased accuracy. It may also be possible to further increase throughput by adding support for concurrent patch extraction and feature extraction.

References

[1] T. Ahonen, A. Hadid, and M. Pietikainen. Face description with local binary patterns: Application to face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2037 –2041, dec. 2006.

[2] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004. 10.1023/B:VISI.0000013087.49260.fb.