Face Detection CUDA Accelerating

Jaromír Krpec
Department of Computer Science
VŠB – Technical University Ostrava
Ostrava, Czech Republic
krpec.jaromir@seznam.cz

Martin Němec
Department of Computer Science
VŠB – Technical University Ostrava
Ostrava, Czech Republic
martin.nemec@vsb.cz

***Abstract*— Face detection is very useful and important for many different disciplines. Even for our future work, where the face detection will be used, we wanted to determine, whether it is advantageous to use the technology CUDA for detection faces. First, we implemented the Viola and Jones algorithm in the basic one-thread CPU version. Then the basic application is widened to the multi-thread CPU version. Finally, the face detection algorithm is also implemented for the GPU using CUDA technology. At the end, final programs are compared and the results are presented in this paper. For our future plans, the speed-up of face detection is very important. By supporting CUDA technology, the process of the face detection showed considerable speed-up.**

Keywords-CUDA; GPU; Face Detection; Viola and Jones algorithm

I. INTRODUCTION

Face detection in images is quite complicated and a time-consuming problem, which found use in different disciplines, e.g., security, robotics, or advertising. By computer performance, disciplines of image processing, such as face detection, have significantly improved and progressed.

Even on current hardware, face 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 the 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 2D or 3D graphics, but it can also be used for varied, especially parallel computations, which are not connected with the original task of graphic cards-rendering.

Compute Unified Device Architecture (CUDA) [1] technology is used to speed up the process of face detection, therefore we moved the main computation to the graphic card. Then, the final implementation was compared with a similar one-thread CPU and multi-thread CPU programs.

The main targets of this work are

- To implement the Viola and Jones algorithm [2] for the multi-thread CPU application,
- To implement the Viola and Jones algorithm for the GPU,
- To compare the speed of detection of individual programs, especially depending on the input image size, and
- To summarize and discuss results.

II. RELATED WORK

There is much work which describes methods of face detection. There are methods which are based on template matching, skin detection and other techniques.

The algorithm of skin detection looks for areas covered by the skin color. Then, these areas could be marked as a face after fulfillment of other conditions (shape). Y. Chen and Y. Lin [3] widened this method. They added hair detection. If area of the face and area of hair are connected, the face is detected.

Template matching is a method for finding small parts of an image which match a template image containing face. Skin detection and template matching also can be combined into one method [4].

Next group of methods is based on the machine learning algorithms. The Viola Jones algorithm belongs to this group and it was chosen as the algorithm for acceleration in this work. The main reason why this face detection algorithm was chosen is the system of how this algorithm works. Thanks to using detection windows and Haar features, it offers a few of ways, of how to parallelize the detection process. The next reason is that there are many algorithms for face detection based on Viola and Jones.

A few works are also written about acceleration object classification with some good results. For example, in the work by C. Gao and S.L. Lu reached for image size 256 x 192 pixels 37 frames/sec for 1 classifier and 98 frames/sec for 16 classifiers [5].

III. VIOLA AND JONES ALGORITHM

The Viola and Jones algorithm was used for the face detection, which was divided into two parts. The first part is for training a set of classifiers. For this process, MIT CBCL Face Database [6] was used. This database contains 2429 images with face and 4549 no-face images. Based on these

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

samples weak classifiers are created. The second part it is the detection itself, when faces are detected in the input image.

A. Features

Haar features are used for computing feature values during training and detection. A weak classifier is always described by a Haar feature that was chosen during the training process.

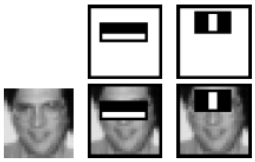

Figure 1. Using Haar features in the input image [2]

Every classifier consists of a black part (marked as B) and a white part (marked as W), and these parts cover some area of the image. Pixels are joined with one of these two parts of features and the final feature value for the current area of the image is calculated as:

$$f(x) = \sum_{w \in W} x(w) - \sum_{b \in B} x(b) \quad (1)$$

That means:

$$II(x,y) = Data(x,y) + II(x-1,y) + II(x,y-1) - II(x-1,y-1) \quad (2)$$

Data represents the original image, II is the integral image and (x,y) is the current position in the image.

B. Integral image

The integral image is an (m+1) x (n+1) multidimensional array created from an input image with m x n dimensions, where every value is counted as a sum of pixels in the interval (0...I - 1) x (0...I - 1). So:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y') \quad (3)$$

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

0	0	0	0	0
0	1	2	3	4
0	2	4	6	8
0	3	6	9	12
0	4	8	12	16

Figure 2. On the left, original image data; on the right, integral image

Now, the sum of pixels for a selected area can be easily quantified with constant speed:

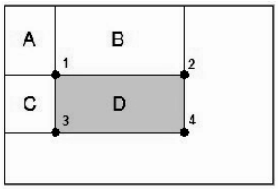

Figure 3. Sum of pixels in integral image

The value in point 1 is made of the sum of pixels in area A, the value in point 2 is A+B. For point 3, the value can be determined as the sum of areas A + C and in point 4 it is equal to the sum of all four areas A+B+C+D. By knowing the area of D, the corner point values could be used to determine the sum of pixels in this area: 4+1-(2+3).

Integral image computation is continual scanning of the full input image. Nevertheless, this operation could be parallelized, especially in case that large data is prepared for the input.

The principle of counting is in divided into two parts. Because results of pixel summation in rows do not influence themselves (and it is same for columns), it is possible to count pixel sums in rows and pixel sums in columns from the new data. During computing more rows or columns will be processed then.

At first, parallel processing of all rows is completed:

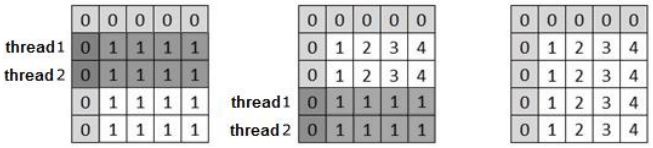

Figure 4. Parallel computing of rows

Then, the final integral image array is gained by the parallel computing of columns:

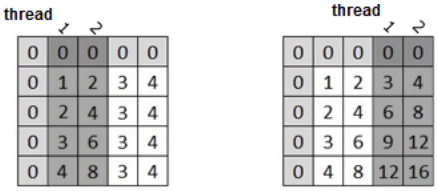

Figure 5. Parallel computing of columns

C. Detection

The detection process has a few steps. At first, the file with trained classifiers is loaded and the input image is also it is loaded, where the application will detect faces. This image is transformed to grayscale and creates the integral image and square integral image. These images are used for computing a standard deviation.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

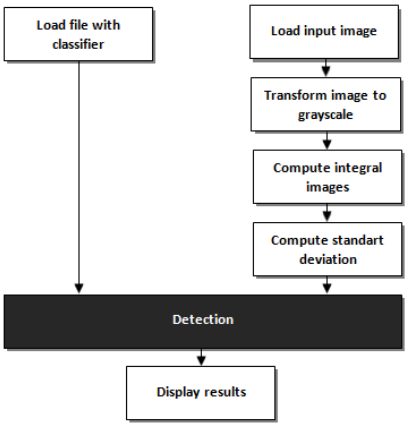

Figure 6. Detection steps

It then runs the main process of detection. During this operation, the parts of the image-detection window are counted. The minimal size of the detection window is the size of the images, which were used during the training.

The detection window is moved through the whole image and tries to determine if the current window could be a face. At the moment, when the window is in the last position in the image, the size of this detection window is enlarged by some rescale coefficient. Then, again, the full image is processed by the transformed detection window from the start position.

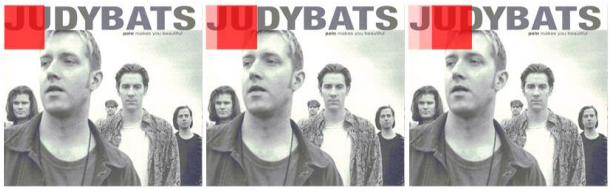

Figure 7. Detection window process

The situation is dependent on the feature value of the detection window, if the window is marked as a face or not. A standard deviation must be computer for every window, which is used for the feature value. Thanks to the integral image, we can count this operation with a constant time and it does not depend on the size of the window.

From the idea of the detection window, we can say that the computation time is affected by the count of trained classifiers, but especially by the image size.

It is obvious that the image size will increase the time calculation, because it must test more detection windows. Thanks to a greater image size, there are also more frequent transmissions of information between the device and host application.

The next table shows how the count of detection windows depends on the input image size (144 x 192 pixels) with the scale coefficient of 1.2.

TABLE I. COUNT OF DETECTION WINDOWS DEPEND ON SCALE

Scale	Detection windows
1,2	20449
1,44	4897
1,72	4480
2,07	4081
2,48	1617
2,98	748
3,58	589
4,29	286
5,16	128
6,19	44
Result	**37319**

D. Cascade algorithm

The cascade algorithm created by Viola and Jones brought some improvements in detection speed. Decreasing the time which is need for detection is based on the condition that there are more areas that do not contain faces. That is why it is not necessary to test all classifiers.

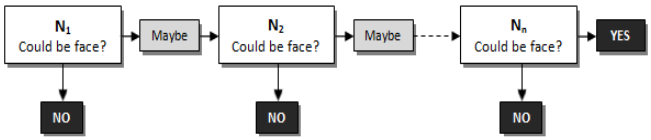

Figure 8. Cascade algorithm

The same MIT CBCL Face Database was used for the training cascade classifiers. The final count of strong classifiers is 15.

Each one of these strong classifiers contains a group of weak classifiers. The total sum of weak classifier is 529. It is significantly more than using one strong classifier. But for most detection windows, the algorithm expects that all weak classifiers will not be tested.

TABLE II. COUNT OF WEAK CLASSIFIERS FOR EVERY STRONG CLASSIFIER IN THE CASCADE

Stage	Number of classifiers	Stage	Number of classifiers
1	2	9	35
2	9	10	40
3	16	11	52
4	21	12	54
5	23	13	57
6	27	14	62
7	27	15	71
8	33	**Total**	**529**

Table 2. shows the sums of weak classifiers for every strong classifier in the cascade.

IV. PARALELL PROCESS

Basically, there are three possibilities how to parallel detection process:

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

A. *Detection windows*

Because the final results of feature values do not depend on the other detection window final results, it is easy to parallelize the detection windows. In the same moment we can test more detection windows. This could be realized on a CPU with more threads, but we can also use the GPU.

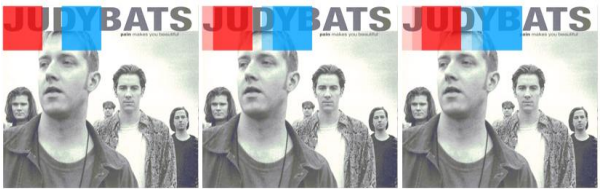

Figure 9. Parallel windows detection process

The problem occurs at the moment, when the detection window is in the last position of the current scale and it is necessary to rescale it. With the rescaling, we also create a different set of features, which must also be rescaled. The problem is when the other threads do not finish their detection and they are still in the old scale. At the moment, when features are rescaling, other threads will use a bad set of features and it could make bad detections.

One of the solutions could be that every thread will have its own copy of the features. However, from the memory view, this solution is not good enough. That is why we implemented a different solution. It uses only one set of features and it is shared by all the threads. To prevent problems, these threads are synchronized in the moment, when they got through the full image. Then the detection window and features are rescaled and threads are executed again with a changed set of features.

This is the method from these three that was chosen for implementation and testing.

B. *Weak Classifier*

The second way is in the parallel processing of Haar features. It signifies that only one detection window in the same moment is tested by more features.

If only one strong classifier is used, then the application runs parallel through a full set of weak classifiers. In case, when it used set of strong classifiers, only weak classifiers in one stage will be parallel processed. After a positive result of the current strong classifier is achieved, a new set of weak classifiers is loaded.

C. *Scale detection window*

If detection windows are in the last image position, windows change size for the next process. But there is no dependence between the scaling of different detection windows.

For example, if a smaller detection window detects a face in some position that does not mean that a larger window will detect the face in the same position. So, it is possible to have in one set of detection windows in one thread. This is not a very good solution for very large images, because for the small sizes, it could create many windows, therefore, making memory demanding for common hardware.

V. IMPLEMENTATION

CUDA is technology developed by the NVidia Company, which can be used for diverse demanding computations on the GPU.

CUDA uses kernels that are executed n times in every thread and the thread is identified by the specific number.

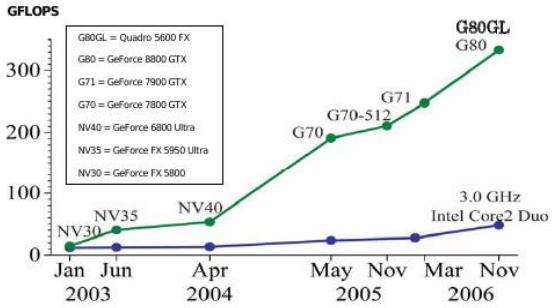

Figure 10. Compare Intel CPU with nVidia GPU [7]

The architecture of CUDA consists of grids, which are divided into smaller units - blocks. The hardware has a group of multiprocessors and it assigns each block to a multiprocessor. And finally, blocks consist of threads. These smallest units can be synchronize in one block.

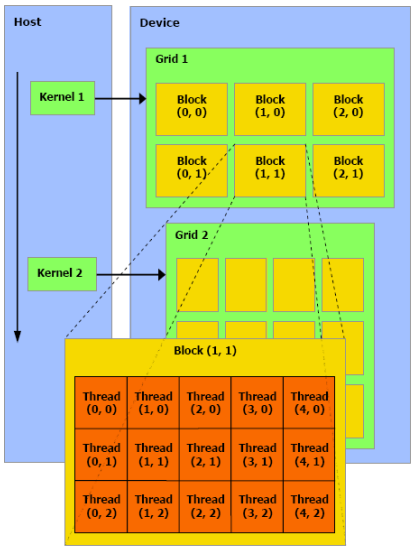

Figure 11. CUDA architecture [7]

In general, the CUDA program starts with memory allocation in the device, while data on the host are prepared. Then, the data are copied from the host to the device. Because the copying from host to device and from the

device to host is a time-consuming process, it is necessary to limit sending data.

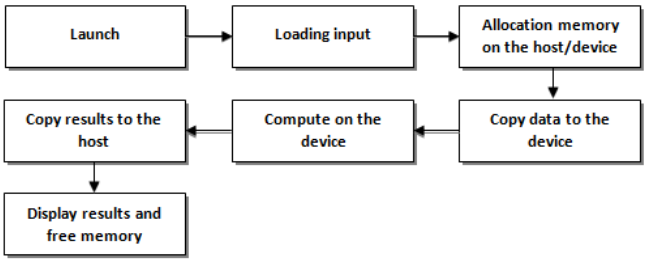

Figure 12. Typicall CUDA program

After the data are ready on the device, it is possible to launch kernels. When the computation is finished, results are copied back to the host.

Finally results can be displayed and the allocated memory is released.

The scheme of the program is the same as Figure 12 shows.

The GPU implementation is similar to CPU multi-thread application. The idea is the same. I copy only what is needed to the device - integral images, trained classifiers and detection windows.

For every detection window's scale, a CUDA kernel is executed. The first version of the program computes positions of detection windows in the client application. A set of windows with the same size is computed. Then it is sent to the device and the detection process for the current windows can start. Of course, the detection window has same size, but the position depends on the index of thread. After that last detection window in the kernel is tested, the results are sent back to the host and the information about new detection windows is prepared on the host. This process is repeated until the scale reaches the final value.

The transmission between client and device is time-consuming, what shows a small adjustment.

For the next version, on the client side a count of detection windows and size is computed. This information is sent to the device. Now, it is possible to compute the position of a current detection window based on the received data from the client and index of thread. This adjustment caused an acceleration of detection an average of 15 times.

In the final implementation, a GPU computation of integral images is also used, which is also described in this paper.

VI. RESULTS

At first, during testing I progressively compare a program with one-strong classifier with a multi-strong classifiers program. For both these implementations, a one-thread and multi-thread variation was created. Then these programs are compared with a GPU program.

A. *Integral image*

Because the integral image is also computed by parallel threads, the following graph presents the results. It shows how the time needed for the computation depends on the image size.

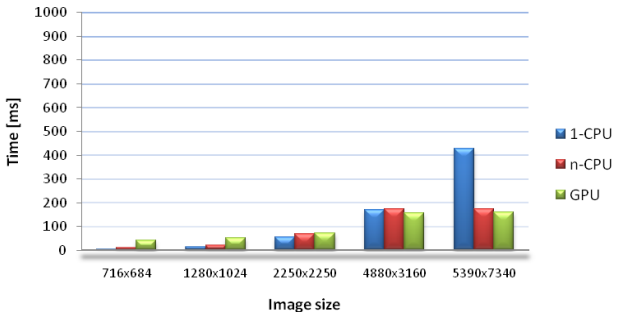

Figure 13. Integral image comparing

From the results we can see, that time computation is lowest for the GPU implementation, while the CPU program is significantly slower.

Nonetheless, the integral image computation is not as time-demanding as detection.

B. *One strong classifier detection*

We then tested a one-strong classifier face detector. The following graph shows how long the process of detection takes for different image sizes. For one-strong classifier, 200 weak classifiers were trained from more than 5000 input samples.

It is not a cascade variation and that is why it is necessary for every detection window to test every weak classifier from the trained set.

For testing, three different image sizes were chose: 716x684, 1280x1024 and 2250x2250 pixels.

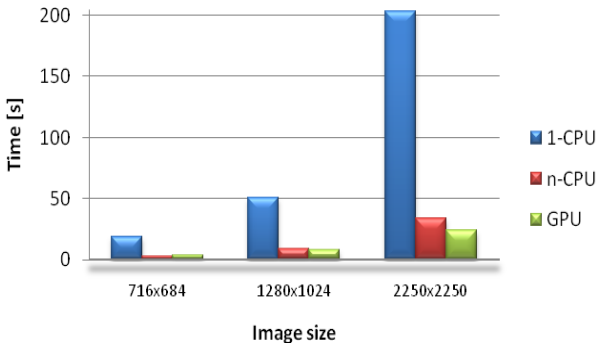

Figure 14. One-strong classifier detection

The results of one-strong classifier detection are 18.44 s for one-Thread for the 716x684 image size, 51.048 s for 1280x1024 pixels and 203.752 s for 2250x2250 pixels.

Eight threads were used for multi-threading testing and the results are 3.128 s, 8.694 s and 33.791 s. For the GPU program it is only 3.495 s, 7.808 s and 24.255 s.

C. Cascade detection

The final comparison is with regards to the cascade variant of the Viola and Jones algorithm. The input file with classifiers contains 15 strong classifiers. Each one of these has a group of weak classifiers.

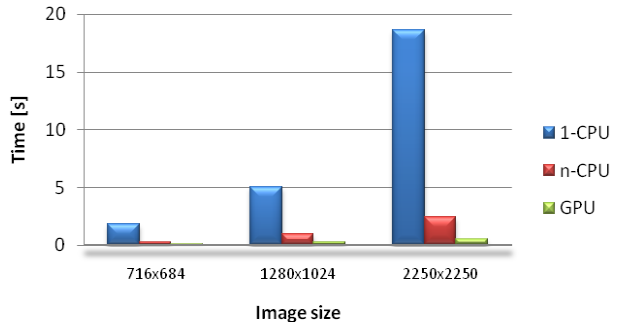

Figure 15. Cascade detection

The result shows that if it is not need test all classifiers set for current detection window, the computation is certainly faster than one strong classifier variation.

In the last graph we can see that one CPU thread is again the slowest from all the implementation modes, but this was obvious from the algorithm principle.

For one CPU thread program it takes only 1.827 s, 5.008 s and 18.644 s. The next testing is with the multi-threading algorithm and results are 0.263 s, 0.989 s and 2.423 s. Finally, the GPU implementation takes only 0.117 s, 0.256 ms and 0.530 s. The final result is the same like the program with one strong classifier, so the GPU detection is the quickest from the presented forms of implementation.

VII. CONCLUSION AND FUTURE WORKS

The possibility of multi-thread and implementation of the Viola and Jones face detection algorithm were presented here. It is not only about CPU thread applications, but especially about GPU.

All programs were tested and it shows that thanks to using threads the face recognition process can be accelerated against the basic one CPU version.

From the graphs, we can see that the detection time depends on the image size; this is the main factor. For the computation of the integral image, GPU implementation is the fastest.

In the next tests, the result is that the one thread CPU variant is obviously slower than the multi-thread CPU and GPU implementation.

From the test results, it is convincing that the GPU detection is usable with reasonable time-consuming results against the CPU variants. It is possible to see that the GPU detection is an average of 35 times faster than one thread CPU detection. In comparison to the multi-thread CPU variant, the results are closer, but the GPU is still quicker.

For the future, it is planned to widen the face detection capabilities for the possibility of recognizing faces with faces stored in a database. The input images will be gained from a camcorder. The real-time detection is the reason why the speed of face detection is very important. The output from the detection will be sent for the comparison with saved faces.

REFERENCES

[1] http://developer.nvidia.com/category/zone/cuda-zone <retrieved: January, 2012>

[2] M. Jones and P. Viola, Robust Real-time Object Detection, International Journal of Computer Vision, vol. 57, no. 2, 2004, pp. 137-154,

[3] Y. Chen and Y. Lin, Simple Face-detection Algorithm Based on Minimum Facial Features, 33rd Annual Conference of the IEEE Industrial Electronic Society(IECON), Taipei, Taiwan, November 2007, pp 455-460

[4] S. Tripathi, V. Sharma, and S. Sharma, Face Detection using Combined Skin Color Detector and Template Matching Method, International Journal of Computer Applications, vol. 26, no. 7, 2011, pp. 5-8,

[5] C. Gao and S. L. Lu, Novel FPGA based haar classifier face detection algorithm acceleration, In Proceedings of International Conference on Field Programmable Logic and Applications, 2008, pp 373-378

[6] http://cbcl.mit.edu/cbcl/software-datasets/FaceData.html <retrieved: January, 2012>

[7] http://www.behardware.com/art/imprimer/659/ <retrieved: January, 2012>

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9