FLOATING POINT HOG IMPLEMENTATION FOR REAL-TIME MULTIPLE OBJECT DETECTION

Mateusz Komorkiewicz, Maciej Kluczewski, Marek Gorgon

AGH University of Science and Technology
al. A. Mickiewicza 30, 30-059 Krakow
email: { komorkie,kluczews,mago}@agh.edu.pl

ABSTRACT

Object detection and localization in a video stream is an important requirement for almost all vision systems. In the article a design embedded into a reconfigurable device which is using the Histogram of Oriented Gradients for feature extraction and SVM classification for detecting multiple objects is presented. Superior accuracy is achieved by making all computations using single precision 32-bit floating point values in all stages of image processing. The resulting implementation is fully pipelined and there is no need for external memory. Finally a working system able to detect and localize three different classes of objects in color images with resolution 640x480 @ 60fps is presented with a computational performance above 9 GFLOPS.

1. INTRODUCTION

Real time object detection and localization in a video stream is important for many vision systems. Car, signs or pedestrian detection is crucial for an advanced driver assistance system. In automated video surveillance detecting human, left luggage or possible dangerous objects is also important.

Histograms of Oriented Gradients [1] is a widely used algorithm for object detection (especially pedestrians). It uses histograms of oriented gradients for feature generation and a SVM (Support Vector Machine) classifier for classification. Unfortunately it has a quite high computational complexity and it is not possible to run it on CPU in real time.

In the past there have been several implementations of this algorithm in FPGA.

Cao et al.[2] in the article from 2008 presented a system based on the reduced HOG and simplified classifier to detect stop signs which was able to process 752x480 @60fps. In the next year (2009) Kadota et al.[3] presented an efficient architecture for computing HOG features, but without classification (30 fps for a 640x480 image). Another group [4] presented an architecture using HOG and an interesting scheme of predicting SVM result which greatly reduce the resource usage (17fps reported for a image of 640x480). In [5] a mixed platform system performing SVM pedestrian detection using FPGA, CPU and GPU was presented. The HOG features were computed using FPGA, then were normalized using CPU. The GPU was used to compute the SVM. They reported more than 10 fps for a 800x600 pixels image. In 2011 a work [6] using HOG and AdaBoost classifiers to detect humans was presented. Authors reported 62.5 fps for a VGA frame (112 fps theoretically possible). In their work a comparison was presented which showed, that due to simplification the hardware versus software implementation had suffered about 16 % drop in a detection accuracy.

Since the previous research proved that implementing the HOG using fixed point arithmetic is reducing the detection rate, in our implementation we decided to use single precision floating point representation. Although using floating point is not common for an FPGA based systems, there have been some interesting implementations of other algorithms done previously [7], [8] with the highest reported data throughput at the level of 160 GFLOPS [9] in radar applications. This results prove that the FPGA devices have the potential to run the original algorithms without any simplification and are able to compete in floating point computing performance with CPUs and GPUs. We present a working system able to classify three different classes of objects (human, head, bicycle silhouettes), able to process VGA stream with 60 fps, running the original state-of-the-art HOG algorithm without any simplification that could affect its accuracy.

2. ALGORITHM DESCRIPTION

The Histogram of Oriented Gradients algorithm was first described in [1]. Available C++ implementation from OpenCV library [10] was used as a reference for porting to FPGA. A short overview of the HOG algorithm, needed to understand the hardware realisation, is presented in the rest of this section.

The work presented in this paper was supported by AGH-UST project 11.11.120.612.

The first step is to apply the gamma correction to all three RGB colour channels of the image. Then a Sobel gradient operator is computed in both x and y direction. Once both components are obtained (g_x, g_y respectively), a magnitude (1) as well as its orientation angle (2) is computed.

$$m = \sqrt{g_x^2 + g_y^2} \tag{1}$$

$$\alpha = arctg\left(\frac{g_y}{g_x}\right) \tag{2}$$

To integrate information from all three colour components into a single magnitude and rotation map, only the values from a channel with the maximum magnitude at each location is preserved.

In order to compute the gradients orientation histogram, the vectors are divided into 9 bins (40 degrees each) according to their direction. Because basic thresholding may cause that a small angle change can place two similar vectors in a different bins, a weighted histogram approach is used. The magnitude value is divided between two closest bins with a ratio of its distance (in degrees) to the centre of the bins.

Once the information about magnitudes and corresponding bins is computed, the whole image is divided into cells and blocks using the square grid of a fixed size. Both cells and blocks overlap in order to compensate a possible object displacement within a grid. In order to accumulate the magnitude values in all cells the module consisting of 16 submodules for computing the single cell histogram was designed (Figure 2).

In each cell a histogram of gradients is accumulated according to their angle value. A voting is also applied (by multiplying magnitudes by corresponding values from 2D Gaussian bell curve) as the central pixels in each cell are more important than the one from boundary. The block is consisting of histograms from four cells.

The values of histograms in a block are normalised using the L2-Hys norm (described in [1]) by executing two times L2 normalisation (3):

$$v = \frac{v}{\sqrt{\|v\|_2^2 + \varepsilon^2}} \tag{3}$$

with limiting each element maximum value to 0.2 between two normalisations.

When the block histograms are computed, they are divided into windows of fixed size, and checked by the pretrained SVM classifier to analyse if an object bounded by the window is similar to the one we are looking for. Therefore checking all possible locations, by sliding the window through the whole image, require many SVM iterations.

The Support Vector Machines theory was presented in the work [11]. The SVM in this form is based on dividing the data into hyperplanes and is able to do a binary classification (negative/positive). It is given by the following equation:

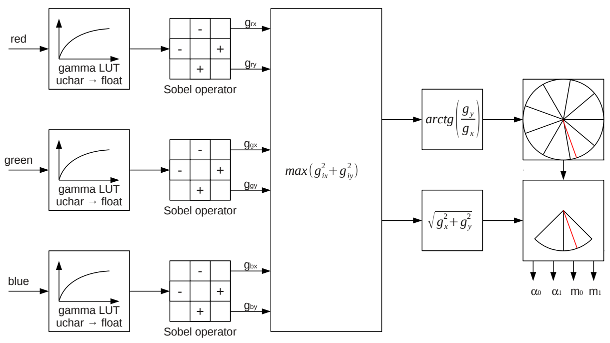

Fig. 1. Hardware gradients computation module

$$r = sign(w \cdot x + b) \tag{4}$$

where $w \cdot x = \sum_i w_i x_i$ and x_i is the i-th feature in the feature vector and w_i is its corresponding weight, b is a bias term.

The values of w_i and b are obtained during the training process which is out of scope of this study. Once trained, SVM classifier becomes tolerant to shape variations and viewpoint changes. Unfortunately, it is not very tolerant to scale differences. The problem is solved either by scaling the input image and repeating the classification or by executing the SVM classification with different window sizes. There are also systems (e.g. video surveillance) where the camera is fixed to one particular location and the problem with the perspective can be overcome by proper camera positioning.

3. PIPELINED HARDWARE REALISATION

In order to achieve the best accuracy, it was decided not to simplify anything from the reference OpenCV 2.1 HOG implementation.

On the one hand, FPGA devices are not consider to be a good platform for implementing the floating point operations. On the other hand, the significant reduction in detection accuracy caused by simplification (described in [1] and [6]), had a decisive influence on the use of floating point in this design.

The FPGA vendors are providing ready to use IP Cores for floating point arithmetic (addition, subtraction, multiplication, division and square root). In Table 3 the resource usage and a maximum operating frequency for Xilinx single precision adder and multiplier IP Core (Virtex 6 familly) are presented. Two conclusions can be drawn regarding the floating point operations using FPGA. The first one is related to latency, when it is equal to one, the operating frequency is quite low comparing with the fixed point multipliers, but with the maximum latency (about 12 clock cycles for a multiplier and 8 cycles for adder) the frequencies are

Operation	Latency	Clock (MHz)	LUTs
ADD	1	77	414
ADD	12 (max)	516	447
MUL	1	104	842
MUL	8 (max)	516	668

Table 1. Xilinx single precision IP Cores specification

almost the same. The other observation is the resource usage. For an adder (32-bit), the fixed point uses only from 32 to 64 LUTs or one DSP48 block. The floating point core is consuming almost 10 times more resources. Comparing the multipliers gives a different result. The fixed point parallel multiplier consumes almost 1088 LUTS or 4 DSP blocks (32-bit operands, 64-bit result) and the floating point multiplier uses almost two times less LUTs.

Unfortunately the floating point cores are not well supported by design tools, as for example there is no possibility to display the signal in floating point radix in simulation programs. The main challenge is to design the system us-

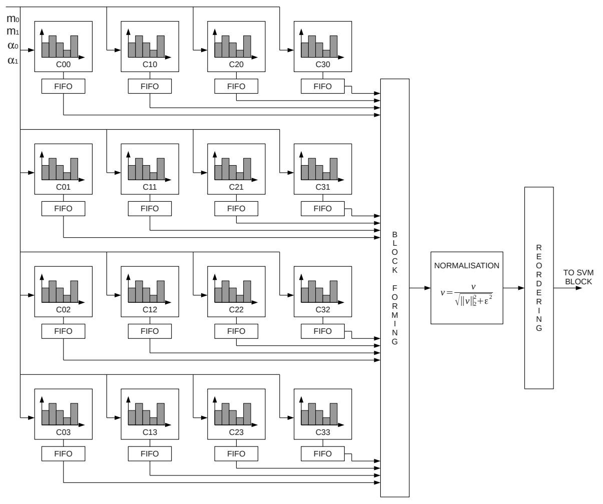

Fig. 2. Histograms computation module

ing the IP Cores with longest possible latency and to do it in pipelined fashion in such a way that no external memory storage is needed (it gives lower cost and power consumption of the final design).

4. MULTIPLE OBJECT DETECTION

Because SVM does a binary classification, different instances has to be used for detecting multiple objects. The proposed multiple object detection system is based on modules described in Chapter 3. The block schematic is presented in Figure 3. The gradient and histogram computation modules are used to process the image, divide it into cells and blocks and compute the feature vector for each block. The data is then streamed to multiple instances of SVM classifiers. We

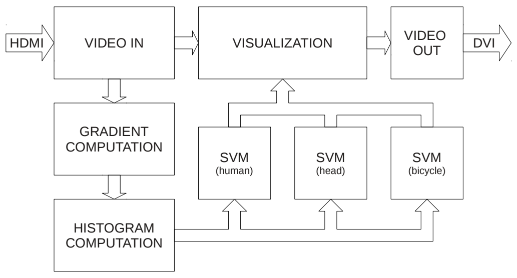

Fig. 3. Classification system

decided to use the system for video surveillance task. In the first block, a human detection is performed. Yet sometimes the whole silhouette can not be seen (occlusion), so in the second block heads are detected. In the third block a bicycles are classified. When an object is detected in SVM, the information is sent to the visualisation module, which is used to draw the rectangles of a specified colour and size on the screen (to mark object type and location).

5. RESULTS

Described implementation was tested using Xilinx ML605 board with Virtex 6 XC6VLX240T device and Avnet DVI I/O FMC expansion card for video input and the results were compliant with the original software implementation. The result obtained by processing the video sequence from [12] is presented in Figure 4. The resource utilisation is pre-

Resource	Used	Available	Percentage
FF	75071	301440	24 %
LUT 6	113359	150720	75 %
SLICE	32428	37680	86 %
BRAM	119	416	28 %
DSP48	72	768	9 %

Table 2. Resource utilisation

sented in Table 5. The design is using almost all slices of FPGA device, but it was our intent to test the limits and instantiate as many SVM classification modules as possible.

The computational performance is presented in Table 5. The number of floating points IP Cores in each module is presented. Most operations in the design are add-multiply, with the simple one (comparisons) balanced by square root or division, so the FLOP unit is quite meaningful in this context. The whole design is working with 25 MHz clock,

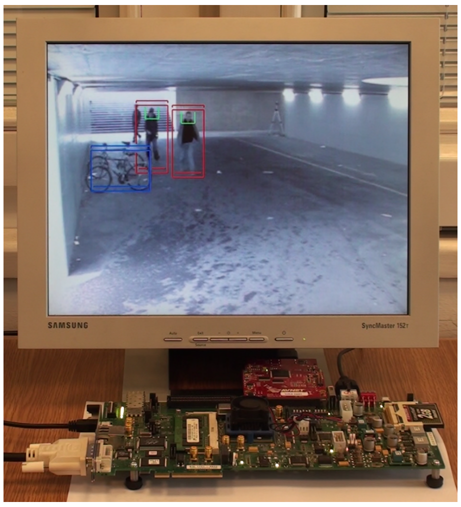

Fig. 4. Working system

with separate clock domains for SVM modules. The computational performance for all blocks reach the level of 9.47 GFLOPS. Implemented system works with 60fps and process a single 640x480 frame in about 16.6 ms. For comparison, the original OpenCV implementation in C++ running on Core i7 2600 (3.4 GHz) process the same image in single scale with an average processing time of 50 ms when MMX and SSE instructions are used and 130 ms when they are not used by the compiler.

6. CONCLUSION

In the paper a working system is presented for multiple object detection using floating point HOG implementation. The original state-of-the-art algorithm was implemented at the cost of higher resource consumption. It proves that FPGA device can be a good platform for executing floating point algorithms and even compete in this area with CPU. The final design has a computational performance above 9 GFLOPS and is able to process 640x480 images @60 fps in real-time.

Module	IP Cores	clock	GFLOPS
Gradient	31	25MHz	0.775
Histogram	77	25MHz	1.925
SVM0	15	237MHz	3.555
SVM1	7	50MHz	0.350
SVM2	27	106MHz	2.862
		Sum	9.47

Table 3. System parallel performance

7. REFERENCES

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, June 2005, pp. 886 –893 vol. 1.

[2] T. P. Cao, G. Deng, and D. Mulligan, "Implementation of real-time pedestrian detection on FPGA," in *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, Nov. 2008, pp. 1 –6.

[3] R. Kadota, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto, and Y. Nakamura, "Hardware architecture for HOG feature extraction," in *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on*, Sept. 2009, pp. 1330 –1333.

[4] M. Hiromoto and R. Miyamoto, "Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, 27 2009-Oct. 4 2009, pp. 894 –899.

[5] S. Bauer, S. Kohler, K. Doll, and U. Brunsmann, "FPGA-GPU architecture for kernel svm pedestrian detection," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, June 2010, pp. 61 –68.

[6] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," in *Field-Programmable Technology (FPT), 2011 International Conference on*, Dec. 2011, pp. 1 –8.

[7] A. Lopes, G. Constantinides, and E. Kerrigan, "A floating-point solver for band structured linear equations," in *ICECE Technology, 2008. FPT 2008. International Conference on*, Dec. 2008, pp. 353 –356.

[8] M. Parker, "High-performance floating-point implementation using FPGAs," in *Military Communications Conference, 2009. MILCOM 2009. IEEE*, Oct. 2009, pp. 1 –5.

[9] W. Chapman, S. Ranka, S. Sahni, M. Schmalz, and U. Majumder, "Parallel processing techniques for the processing of synthetic aperture radar data on FPGAs," in *Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on*, Dec. 2010, pp. 17 –22.

[10] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. Cambridge, MA: O'Reilly, 2008.

[11] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[12] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua, "Multiple object tracking using k-shortest paths optimization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 9, pp. 1806 –1819, Sept. 2011.