

Acceleration of Pedestrian Detection Algorithm on Novel C2RTL HW/SW Co-design Platform

Yihao Zhu^{*}, Yongpan Liu^{*}, Daming Zhang^{*}, Shuangchen Li^{*}, Pei Zhang⁺, Tedd Hadley⁺

^{*} TNList, EE Dept. Tsinghua University Beijing, 100084, China

⁺ Y Explorations, Inc. San Jose, CA, 95134, U.S.

Abstract—Vehicle-based pedestrian detection systems play a very important role in the field of intelligent transportation. However, existing pedestrian detection algorithms are too computing extensive for traditional vehicle-based processor. Considering the performance and flexibility of hardware/software co-design architecture (CPU+FPGA), it is feasible to implement these algorithms on such platforms. In this paper, we present a popular template-based detection algorithm on such a novel platform which including an ATOM processor and a FPGA-based accelerator, and designed by a novel C2RTL automatic design flow for the platform. Furthermore, we propose performance models for the design and explore configuration parameters of the platform for performance optimization. The experimental results show that our design can achieve nearly 20x speedup for the critical function block compared with the processor-only solution to meet real-time requirement.

I. INTRODUCTION

Pedestrian detection systems are essential to avoid dangerous traffic situations. Driving-assisted systems [1] should warn drivers in time before potential collisions with nearby objects—especially pedestrians. Therefore, detection response latency is very crucial for driving assistance on-board vehicles.

There are plenty of research studies in the pedestrian detection fields. Depending to the detecting mechanisms, they can be categorized as motion-based and shape-based approaches. The motion-based methods [2] recognize the person via a moving pattern. The advantages of these methods are insensitivity to environmental changes, such as color and brightness. However, they suffer from long response time due to the need to process image sequences. Among the shape-based methods, the statistical strategy dominates due to their robustness and relatively low latency. By training the system with enough samples, the system can provide accurate enough results within reasonable time. The typical statistical approaches include SVMs [3], Neuron

Network [4] and Boosting [5] classifiers for object recognition. However, those algorithms still suffer from extensive computing, which impedes their use in realtime applications. Although plenty of effort in algorithm design has been done to reduce detection latency, few have explored the parallel computing acceleration on novel vehicle computing platforms.

Among the existing vehicle computing platforms, FPGA and DSP-based system are the most popular choices. High performance DSP-based systems [7] are power hungry and do not fulfill automotive requirements, such as automotive qualification and power budget. Specific architectures [8] (VLIW, SIMD, and MIMD) provided parallel processing capabilities and decreased processor frequencies, however, with increased software development complexity. FPGA-based approaches [6] show good results, but they are very costly and suffer from long hardware developing time, which is not suitable for automotive applications requiring ultra-low costs.

Considering the good performance but lengthy design time of FPGA platforms, we studied how to efficiently implement the algorithm on an FPGA-based platform. Compared with the long cycle of manual hardware design in previous works our approach takes advantage of a novel C2RTL software – eXCite to transfer C language to FPGA development languages VHDL/verilog automatically and can achieve a better result in experiments. Providing a solution to these problems, eXCite offers:

- 1) High-level synthesis, allowing systems to be described in higher abstractions.
- 2) ANSI C to RTL synthesis, allowing the most popular system development language to be used for hardware design.
- 3) Block design reuse, allowing previously designed intellectual property to be used in new designs.
- 4) Platform-ready results, allowing C-specified hardware/software systems to be emulated on FPGA platforms.

In this paper, we present the HOG-based(histograms of oriented gradients) pedestrian detection algorithm on such a novel platform which includes an ATOM processor and a

This work was supported by the NSFC under grant #60976032, the National Science and Technology Major Project under contract #2010ZX03006-003-01, and the "863" Program under contract #2009AA01Z130.

FPGA-based accelerator. We also demonstrate a novel C2RTL automatic design flow for the platform. Furthermore, we propose the performance models for the design and explore the configuration parameters of the platform for performance optimization. Our contributions are listed as below:

- 1) We first characterize the profiles of the HOG-based pedestrian detection algorithm on general purpose Intel Atom processors and pointed out the performance bottlenecks for further improvements.
- 2) We efficiently partition the detection algorithms to a software part and a hardware part. The hardware part will be executed on a FPGA-based accelerator. After the implementation of the basic FPGA-based accelerator, we give the performance result of our approach compared with processor-only architecture.
- 3) In order to give the evaluation of our FPGA platform, we propose a performance model of the architecture. Furthermore, we give performance results of several different implementations and discuss the tradeoff between the performance and area of the architecture. By doing this, we get a clear idea of how to select the architecture configuration.

This paper is organized as follows: We first introduce the research background and highlight our contributions in Section 1. Section 2 illustrates the pedestrian detection system and HOG-based algorithm flow. Section 3 introduces the architecture of our approach and discusses how to partition the algorithm into a software part and a hardware block. After that, we introduce the design flow of accelerators and propose a performance model to evaluate the performance of the system. Section 4 gives our experimental results and Section 5 concludes the paper.

II. PEDESTRIAN DETECTION SYSTEM

In this section, we first describe a modern driving assistant system, in which a pedestrian detection system is one of the key components. After that, the HOG-based pedestrian detection algorithm [9] is illustrated in detail.

A System Diagram

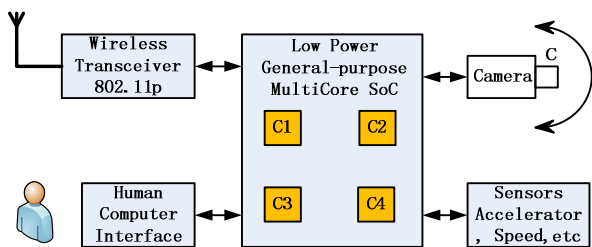


Figure 1. Diagram of driving assistant system

As Figure 1 shows, the modern driving assistant system consists of multiple sensors, vehicle computing system-on-chip (SoC), wireless communication modules and a human computer interface. The camera, speed, acceleration and other sensors collect all kinds of vehicle and environment

information and send them to the vehicle computing system-on-chip (SoC). The general-purpose multi-core processor based vehicle computing platform processes the data and transfers the results to the driver via the human computer interface or to other vehicles as well as the roadside unit if needed.

Among those applications running on the driving assistant system, the pedestrian detection system is one of the key components due to the road safety requirements to avoid collisions without human braking at all.

B Algorithm Flow

In existing pedestrian detection algorithms, HOG-based algorithms provide much lower missing rates and a reasonable computing complexity. Therefore, we study how to map it onto a processor and hardware accelerator. Currently available HOG-based systems [10] are designed heavily using graphic libraries, which makes it very hard to be used on software/hardware co-design platform. Instead, we adopt a dataflow style to redesign the HOG-based pedestrian algorithm as Table 1 shows. By removing the graphic libraries and functioning pointers, our system is more suitable for the partition of software and hardware.

The algorithm detects the pedestrians in a pyramid way. The image is scaled into several pictures with different resolutions. The detection algorithm is deployed from the coarse-grained image to the fine-grained one. For each scaled image, a gamma correction operation (line 4) is used to reduce the effects of brightness and color. In line 5, the gradient magnitudes and orientations of each pixel are computed based on the user-defined template. The histogram is calculated for each 16x16 pixel block in line 6. After that, the histogram is normalized in line 7. Line 8-10 show the progress to determine whether a pedestrian is detected based on a SVM classifier. Finally, a postprocessor is used to highlight the detected persons from the image in line 13.

Table 1 HOG-based pedestrian detection algorithm

Algorithm 1: HOG-based Pedestrian Detection

```

1: pyramid_detection(origin_image);
2: for scale_level = 1 to n do
3:   scaled_image=rescale(origin_image);
4:   nimage=gamma_corr(scaled_image[scale_level]);
5:   (grad_mag, grad_ori)=grad_comp(nimage);
6:   histogram=comp_hist(grad_mag, grad_ori);
7:   normhist=hist_normalizer(histogram);
8:   hist_win=window_detect(normhist);
9:   for win_num = 1 to m do
10:    score=svm_classifier(hist_win[win_num]);
11:   end for
12: end for
13: postprocess(score, location);

```

III. SYSTEM ARCHITECTURE

After the implementation of the pedestrian detection system on a general purpose processor, this section will introduce the architecture of the hardware/software co-design platform. First we give the entire architecture of the system, and then focus on the generation flow of the FPGA-based accelerator. Furthermore, based on the different features of performance between accelerator and the base processor, we propose a performance model to evaluate the performance of the system.

A. Atom + FPGA Accelerator Architecture

Figure 2 shows the architecture of our hardware/software co-design system. This system is composed of two parts: an Intel Atom processor, which is mainly used in netbooks and mobile intelligent devices, and an FPGA-based accelerator block. The Atom processor will execute the whole algorithm while the accelerator is for the blocks which are less efficiently run on the base processor.

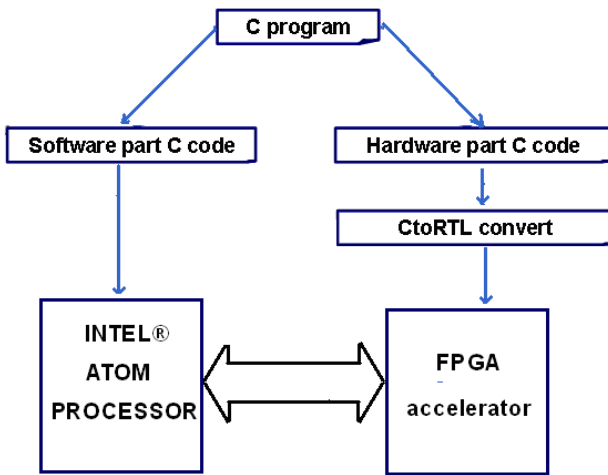


Figure 2. Architecture of hw/sw platform

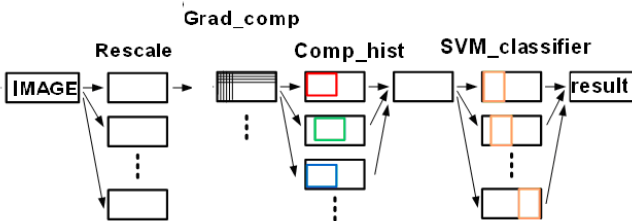


Figure 3. Potential parallelization of the HOG algorithm

Because of the application-specified features of the system, we first profile the whole algorithm and identify the application characterization. As Figure 3 shows, the dataflow of algorithm execution presents parallel execution potential in the rescale, *comp_hist* and SVM_classifier operations. And

from Table 3, we find that the *comp_hist* block takes about 50% of the total running time. The *comp_hist* block is very suitable and reasonable for a hardware implement for acceleration. The *comp_hist* block is used to compute the gradients histogram of a 16 *16 pixels block. Its input is gradients magnitude and orient of 16*16 image block and its output is a 36-dimension histogram vector. Actually, the *comp_hist* is 256 iterations of its two sub-function – *toindex()* and *interpolate_linear()*.

B. Accelerator Design

In contrast to previous works which manually designed the FPGA accelerator, our approach uses a novel C2RTL software to get the result. eXCite is such a software produced by Y Explorations Inc.. It can convert C language code block to HDL language blocks, and satisfy the constraints that users set.

In order to get high quality HDL language based block, parameters are set, including inserting some pragmas in the C language code and setting design constraints such as frequency and area optimization options. By use of this software, we can significantly shorten the design cycle and obtain the required hardware block. Design flow is shown in Figure 4:

1) We first characterized the profiles of HOG-based pedestrian detection algorithm on general purpose processor and point out the performance bottlenecks for further improvements. As discussed in III.A, we choose the *comp_hist* block for our target to accelerate.

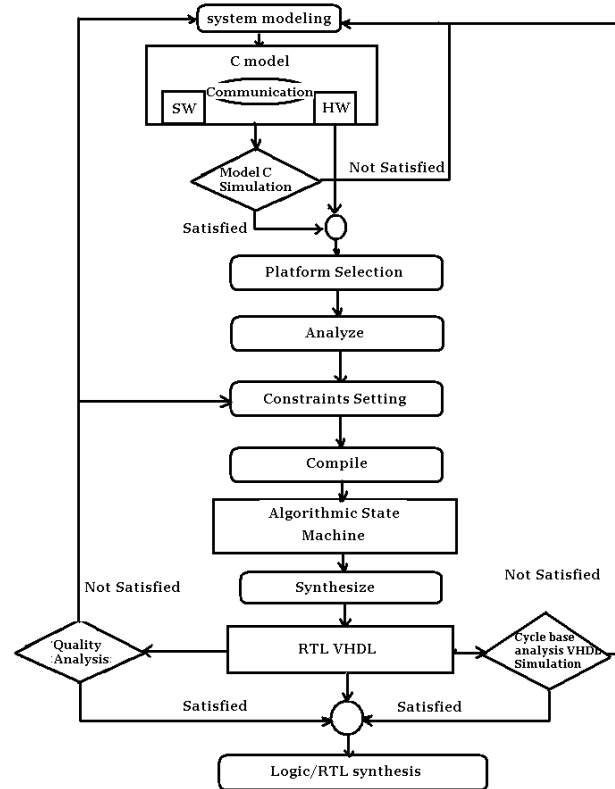


Figure 4. C2RTL design flow

$$T_{c-total} = (n/m) * (T_c * m) \quad (3)$$

2) We extract the hardware part from the whole algorithm and decide input and output channels for the hardware block. In our approach, considering the *comp_hist* block's functionality, its input is gradients magnitude and orientation of a 16*16 image block and its output is a 36-dimension histogram vector.

3) For the purpose of better performance, the C language code can be optimized to make it more suitable for hardware. The eXCite software supports some optimization methods for the hardware. First, if a lengthy loop exists in the code to be converted to hardware, eXCite support an unrolling mechanism to change serial behavior into parallel. Another attractive technique is pipelining. A pipeline architecture can be achieved after setting the latency and throughput of a design or loop. Such optimizations can provide significant improvements to the design.

4) After the initialization and code-optimization of the block, we can use eXCite to test the functionality between software and hardware. In this step, we compile both software part and hardware part into an executable program and then run with them together. The simulation result will be written into a test vector file which will be used in step 6.

5) After the functional verification is finished, we do next is doing automatic hardware synthesis by using the eXCite software. We set clock frequency, channel type and optimization directions (platform whether components should be shared or not and other optimization options). After compilation, we will get an FSM image of behavior and we will get HDL-language based accelerator block as well as testbench files after eXCite synthesis.

6) Finally, we can use simulation tools such as Modelsim to verify the block. This step will use the testbench files generated in step 5 to test the HW block using the testvector file in step 4.

C. Performance Model

With the work in section III.B, we construct a performance model to evaluate the improvement of performance. It is worthy to note that our performance model not only takes into consideration the execution time of accelerator and processor, but also the estimation of the interconnection overhead between hardware and processor. Because the calculated object is the gradients of pixel blocks of the image, actually we can specially allocate memory for this in the FPGA, so our estimation of communication overhead is based on the shared memory model (SMP). According to the assumptions of this model, the execution of accelerators can be done simultaneously but the memory access may suffer from data collision. As a result of this, the performance equation is:

$$T = T_p + T_a * n/m + T_{c-total} \quad .1$$

$$Speedup = T_s / [T_a * n/m + T_{c-total}] \quad (2)$$

where:

- T_p : the time Atom processor takes for the whole algorithm except *comp_hist*() block;
- T_a : the time the accelerator executes once;
- T_s : the time *comp_hist*() takes in the processor;
- T_c : the time *comp_hist*() takes to access data from the memory each time;
- $T_{c-total}$: the total time for data access from the memory in the whole algorithm;
- n : the times the accelerator is called;
- m : the number of the accelerators on the FPGA.

In our experiments, T_p and n are given by the gprof profiling tool, and from Modelsim simulation we can obtain an approximation of T_a and T_c . m is decided by the FPGA capacity and the area of a single accelerator. By means of these equations, we can estimate the running time of total algorithm and the speedup compared with the base processor only.

IV. EXPERIMENT RESULT

This section first presents the experimental setup of the pedestrian detection system. Then the performance profiling of the pedestrian algorithm is provided and analyzed. The hardware accelerator is simulated with simulation tools and the performance result is given. Finally, we propose several strategies for how to improve the system's performance further.

A. Experiment setup

The pedestrian detection algorithm is implemented in C language running on a mobile netbook with an INTEL Atom N270 1.6GHz processor. We adopted the gprof tools to profile the software performance under the Ubuntu8.04 Linux operation system. The eXCite C2RTL software is used for to generate the hardware accelerator block. We obtained the execution time of the accelerator block from Modelsim simulation.

B. Performance profiling

The run time of the algorithm to static image on a single core Atom processor is listed in Table 2. For a normal case in a driving assistant system, an image of 640*480 is big sufficient for pedestrian detection. However, from the detection time for a static image listed below, it's clear that the run time is too long for real-time use.

Table 2 Run time under different resolution by gprof

Image Resolution	Detection Time (s)
640*480	24.38
960*720	48.37
1024*768	86.12

A detailed gprof report is shown in Table 3. The percent of each function's execution time is included in this table. We can easily find that *comp_hist* (including two sub-functions---- *toindex* and *interpolate_linear*) takes about 50% of the total running time. On the other hand, this block is called 43288 times. So there's no doubt that we should choose *comp_hist* for the hardware acceleration. From the called times and run time, we obtain the time that one *comp_hist* function needs is about 0.3ms.

Table 3 Performance profiling of pedestrian detection algorithm

Function Name	Call Times	Run Time (s)	Percents (%)
[0] main	1	24.38	100
[1] comp_hist	43288	13.10	53.7
-toindex	11081728	2.58	10.6
-interpolate_linear	11081728	5.13	21
[2] rescale	23	4.98	20.4
-verticalFilter	22	2.14	11.7
-horizontalFilter	22	2.11	8.7
[3] svm_classifier	24944	1.50	6.2
[4] grad_compute	23	1.46	6
[5] hist_normalizer	43288	0.18	0.7

C. Accelerating Results

The accelerator results include two parts: the synthesis result and the simulation result.

The synthesis result is obtained by the eXCite software. In our design, we set the running frequency at 100MHz, and choose the optimization as best performance with maximum components sharing. This configuration guarantees our design high performance with acceptable area. After doing the automatic C2RTL synthesis, a synthesis report is given and the result listed in Table 4 below

Table 4 synthesise report of hardware block

Item	Value
Requested clock	10.000ns(100MHz)
Estimated delay	4.855ns(206MHz)
Estimated area	1168 LUT

After generation of the HDL-language-based block, the next step is to simulate the block with the testbench and testvectors which was created in functional test above. The simulation is done under Modelsim environment. By means of simulation, we can get information including running time, running cycles and the correctness of the result.

In our design, the maximum simulation frequency is 200MHz and the result is listed in Table 5. In table 5, execution cycle means the number of cycles which is used to compute and interconnect cycles is used to interconnect with memory. From the Modelsim result and the performance model, the total running time of single accelerator block can

be calculated and it is about 0.136ms, which is a 2x speedup compared with the base processor implementation. It is clear that single FPGA accelerator at a low frequency brings us limited improvement of performance, so we will discuss the further improvements in the next section.

Table 5 Modelsim report of accelerator block

Item	Value
Frequency	5.000ns(200MHz)
Execution cycle	26717
Interconnect cycle	548

D. Performance profiling and Further Improvement

Through performance profiling and the accelerator's synthesis and simulation, we collect enough information about the system and can obtain a performance evaluation by using the performance model. In this part, we focus on four different kinds of implementations of the system and compare their performances in calculating the *comp_hist* block. These four kinds of implementations are listed in Table 6 below:

Table 6 Performance profiling of different implementations

Implementation type	Run Time /block (ms)
Atom processor	0.3
Single FPGA accelerator	0.136
ASIC accelerator	0.017
Multi FPGA accelerator(10)	0.016
Multi FPGA accelerator(100)	0.004

The first is an implementation on the ATOM processor directly. Through the performance model, we know that each histogram block's computation takes about 0.3ms (13.1/43288). On the other hand, if we assign a single FPGA *comp_hist* accelerator, it will take about 0.136ms to run (simulation in section IV. C).

Due to the low speed of the FPGA device we used, our FPGA implementations don't run very fast. If we use an ASIC accelerator with frequency (1.6GHz) similar to the ATOM processor, a histogram block will take about 0.017ms..

Considering that area of a single HW accelerator block is not large compared with the total area of the FPGA device, we could adopt a multi-core like parallel method to solve the problem. In addition, as was researched in [11], the *comp_hist* function is responsible for the repeated computation of histograms in overlapping 16*16 pixel blocks. After mapping the computation sequence of the block, we can remove the data dependence during the parallel process. This means that we can load several accelerator blocks into the FPGA and achieve more parallelism. In this experiment, the size of the image is 640*480, and 30*40

histogram blocks can be calculated simultaneously at most. On the other hand, the area of a block is 1168 and a Virtex-5 FPGA board can have 2000000 LUTs. It's possible to load 100 blocks in a board. However, under this configuration, the interconnection overhead, while constant, begins to dominate the execution time as the main performance bottleneck. For example, for an implementation of 10 blocks in FPGA, the total interconnection overhead in one parallel execution is about 2.74us while the execution time is 134us, which means running time per block is about $134/10+2.74us=16us$. But for an implementation of 100 blocks, the running time per block is about $134/100+2.74us=4us$, and the interconnection overhead plays a leading role. For multiple accelerator blocks in the FPGA implementation, a tradeoff of the performance and area should decide the number of the accelerators, so the selection of proper number of accelerators is also an important issue.

In conclusion, for the architecture of 10 accelerators on FPGA device, according to the synthesis report (206MHz maximum frequency) and the simulation result, we can achieve a performance of 15 us per block (including execution and interconnect overhead). Contrasting this with the running time of a base processor (300us), we obtain a speedup of about 20 times. This is a significant improvement.

V. CONCLUSION

This paper illustrated an acceleration method of a pedestrian detection algorithm on a hardware/software co-design platform. By using eXCite, a novel C2RTL software, we obtain hardware acceleration in a more efficient way. By profiling the performance, we proposed a multi-accelerator architecture to improve performance and discussed tradeoffs between performance and area for number of accelerators. Experimental results show that our design can achieve nearly

20x speedup for the critical function blocks of the HOG algorithm and thus reduce the total time by about 50%.

REFERENCES

- [1] U.S. Department of transportation, national highway traffic safety administration: Task 3 final report: Identify Intelligent Vehicle Safety Applications by DSRC, Public Document, 2004
- [2] Ran Yang, Zheng Qin-Fen, Weiss I, Davis L S, AbdAlmageed W, Zhao Liang. "Pedestrian classification from moving platforms using cyclic motion pattern". In: Proc of Int Conference on Image Processing. Genoa, Italy, Page(s):II-854-7, 2005
- [3] Cheng Hong, Zheng Nan-Ning, Qin Jun-Jie. Pedestrian detection using sparse Gabor filter and support vector machine. In Proc. of Intelligent Vehicles Symposium. Vienna, Austria, Page(s): 583 – 587, 2005
- [4] Szaras M, Yoshizawa A, Yamamoto M, Ogata J. Pedestrian detection with convolutional neural networks. In: Proc. of IEEE Intelligent Vehicles Symposium. Las Vegas, Nevada, Page(s): 224 – 229, 2005
- [5] Viola P, Jones M. "Rapid object detection using a boosted cascade of simple features". In Proc on Computer Vision and Pattern Recognition, Hawaii, Page(s): I.511– 518, 2001
- [6] D. Han, D.-H. Hwang, "A Novel Stereo Matching Method for Wide Disparity Range Detection", Page(s):643 – 650, ICIAR 2005
- [7] <http://www.ti.com>
- [8] Axel Techmer, "Application Development of Camera-based Driver Assistance Systems on a Programmable Multi-Processor Architecture", Proc. of the IEEE Intelligent Vehicles Symposium Istanbul, Turkey, Page(s): 1211 – 1216, June 13-15, 2007
- [9] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In Proc. of Computer Vision and Pattern Recognition. San Diego. IEEE, Page(s): 886 –893, 2005.
- [10] Navneet Dalal. Finding People in Images and Videos .PhD Thesis. Institut National Polytechnique de Grenoble / INRIA Grenoble , Grenoble, July 2006.
- [11] Yongpan Liu, Yihao Zhu, Sunny Zhang, Senjie Zhang, Huazhong Yang, Acceleration of pedestrian detection algorithm on multi-core vehicle computing platform, Beijing, Page(s): 208– 211, ICT2009