Pedestrian Detection Over 100 fps with C4 Algorithm

Fei Wang[1(✉)], Caifang Lin[1], and Qian Huang[2]

[1] Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518000, People's Republic of China
1392135844@qq.com

[2] Wright State University, Dayton, OH 45324, USA

Abstract. In this paper a novel pedestrian detection algorithm on GPU is presented, which takes advantage of features of census transform histogram (CENTRIST), rather than common HOG feature. The proposed algorithm uses NVIDIA CUDA framework, and can process VGA images at a speed of 108 fps on a low cost notebook computer with a GPU, while without using any other auxiliary technique. Our Implementation enables a factor 17 speedup over original CENTRIST detector while without compromising any accuracy.

Keywords: Pedestrian detection · GPU · CENTRIST · CUDA · Real-time

1 Introduction

Pedestrian detection plays an important role in many applications, for example video surveillance, mobile robots, driving assistance systems and so on. In some applications like robot, real-time detection is critical [1]. There are many popular descriptors for pedestrian detection, for example HOG [2], LBP [3], ChnFtrs [4] and so on, but algorithms based on these descriptors are generally slow for real time applications.

Graphics Processing Units (GPUs) can run massive threads in parallel, and can accelerate a variety of image processing tasks. Quite some works have been conducted on pedestrian detection. There have been many studies of using the GPU to accelerate HOG-based pedestrian detection. Wojek et al. [5] achieved 30 times speedup on INRIA person test set. Prisacariu and Reid [6] achieved a 67 speedup in color mode and a 95 speedup in grayscale mode. Lillywhite et al. [7] presented a real-time implementation which can process VGA images at 38 fps. Benenson et al. [8] presented a very fast pedestrian detection algorithm which can run at 100 fps. This work was the fastest algorithm before our work described in this paper. Big differences between our proposed algorithm and Benenson's algorithm are: (1) Benenson's algorithm is based on ChnFtrs descriptors [4], ours is based on CENTRIST descriptors (2) Besides GPU parallel computing, Benenson's algorithm takes advantage of other auxiliary algorithms like stixel and ground plane estimation, therefore detections are conducted on reduced area, our algorithm explores GPU's parallel computing capability only. Most of all, our detection speed outperforms Benenson's using the same hardware. More detail analysis can be seen in Sect. 4.

The critical information in features can lead to efficient detection architecture. Wu et al. propose Census Transform Histogram (CENTRIST) descriptors, which can characterize critical contour information for pedestrian detection [9]. It is remarkable that the algorithm (C4) can detect pedestrians in real time with high accuracy. As an example, human detection based on C4 is illustrated in Fig. 1.

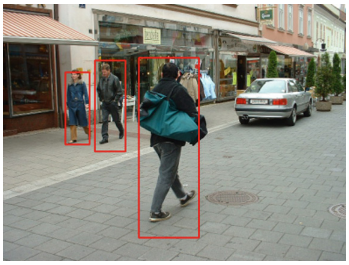

Fig. 1. An example of C4 for pedestrians

In this paper a GPU version of C4 algorithm is presented, which can detect pedestrians at 108 fps for VGA images. Section 1 gives a survey on similar works now days. Section 2 briefly describes the CENTRIST visual descriptor. Section 3 introduces the parallel version of the C4 algorithm in detail. Section 4 summarizes the performance of the parallel C4 algorithm, and Sect. 5 concludes this paper.

2 CENTRIST Descriptor

Global contour is believed to be the most useful information to characterize a pedestrian [9], and signs of comparisons among neighboring pixels represent this information. CENTRIST visual descriptor encodes this sign information, and does not require pre-processing (padding image and gamma normalization).

Building CENTRIST visual descriptors begins from computing Sobel gradient image. Sobel image can smooth high frequency local texture information, and can hold the contour information. And then Census Transform (CT) is conducted, which can be illustrated as Eq. 1. The pixel in the center is compared with its eight neighboring pixels. A neighbor pixel value will be replaced by a bit 1 (0) if the gray level of the central pixel is bigger (smaller) than that of the neighbor pixel. The resulting eight bits are collected from left to right and from top to bottom. The eight bits are converted to a radix-10 number in [0 255].

$$\begin{array}{ccc} 32 & 32 & 90 \\ 82 & 82 & 96 \\ 64 & 64 & 98 \end{array} \Rightarrow \begin{array}{ccc} 1 & 1 & 0 \\ 1 & & 0 \\ 1 & 1 & 0 \end{array} \Rightarrow (11010110)_2 \Rightarrow \mathrm{CT} = 214 \tag{1}$$

This number is named as the CT value of the central pixel. Histogram based on CT values is named as CENTRIST descriptor [10] (Fig. 2).

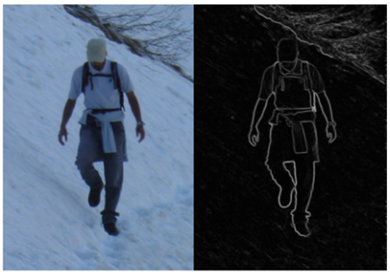

Fig. 2. An example of C4 for pedestrians

A window is slid over the image, and CENTRIST features are evaluated at the current window position for pedestrian detection. The detection window size is 108 × 36. The window is split into 9 × 4 cells, so each cell contains 12 × 9 = 108 pixels. Any adjacent 2 × 2 cells can be combined into a block, so the number of blocks in a window is 8 × 3 = 24. As the feature vector for each block has 256 dimensions, a window has a feature vector with a size of 256 × 24 = 6144 dimensions. Then this feature vector is fed into a two stage cascaded classifier to decide whether current window contains any pedestrian. The linear classifier guarantees fast testing speed, and HIK SVM classifier achieves high detection accuracy [11, 12].

3 Parallel C4 Algorithm

Only a small portion of our parallel implementation is performed on the host processor. The image and cascade classifier are loaded from the host. Final results, i.e., window locations on all image scales containing a pedestrian, are returned to the host for non-maximal suppression (NMS). While all the other computations are performed on the GPU.

3.1 Calculating CENTRIST Features

As a pedestrian in the image may be bigger than the window size, the pedestrian can be detected after the image is shrunk until the pedestrian can fit into the window. In our implementation, the image is shrunk seven times, and a pyramid with seven levels is created, which is stored in the global memory. At each level, CENTRIST features are calculated at every sliding detection window location.

Sobel is an edge detection operator, which can be created after two successive 3 × 3 convolutions are applied on the gray level image. An example code named Sobel Filter from the NVIDIA CUDA SDK is utilized. On the Sobel gradient image, CT transform is conducted except those boundary pixels, as no padding is applied. Signs of comparisons between neighbor pixels are encoded. All above computation take the same execution configuration: the thread block dimensions is set as 16 × 16, and the thread grid dimensions is set $\frac{h_r}{16} \times \frac{w_r}{16}$ (h_r and w_r mean the height and width of the image at current scale after being shrunk), with each thread computing one pixel. As long as all CENTRIST features are available, they are fed into the linear classifier, where windows which do not contain any pedestrian are dropped.

3.2 Linear Classifier Evaluation

In traditional approach like SVM, to judge whether a window contains a pedestrian, the following formulation is evaluated, where f denotes the feature vector, and ω denotes a linear classifier trained offline. Both of them may be broken into chunks to fit the block size:

$$\omega^T f = \sum_{i=1}^{8} \sum_{j=1}^{3} \omega_{i,j}^T f_{i,j}. \tag{2}$$

If $\omega^T f \geq 0$, then a pedestrian is detected, otherwise not. Unlike single SVM, Wu et al. proposed a two-stage classifier [9], i.e., a linear classifier followed by a HIK SVM classifier. Using the linear classifier, windows which do not contain any pedestrian, i.e. $\omega^T f < 0$, can be excluded quickly. However, any feature vector which can pass the first-stage linear classifier, i.e. $\omega^T f \geq 0$, does not guarantee the existence of any pedestrian in current window. Further evaluation is conducted at the second-stage classifier, i.e., the HIK SVM classifier.

Suppose a cell has a size of $h_s \times w_s$, and a block has a size of $2h_s \times 2w_s$, and (t, l) is the coordinate of the upper left corner of the detection window, $\omega_{i,j}^k$ is the k-th component of $\omega_{i,j}^k$, and $C(x, y)$ means a pixel in CT image, then $\omega^T f$ in Eq. 2 can be converted into the following form:

$$\sum_{i=1}^{8} \sum_{j=1}^{3} \sum_{x=2}^{2h_s-1} \sum_{y=2}^{2w_s-1} \omega_{i,j}^{C(t+(i-1)h_s+x,l+(j-1)w_s+y)} \tag{3}$$

To accelerate the computation of Eq. 3, Wu et al. calculate an auxiliary image A instead.

$$A(x, y) = \sum_{i=1}^{8} \sum_{j=1}^{3} \omega_{i,j}^{C((i-1)h_s+x,(j-1)w_s+y)} \tag{4}$$

Then Eq. 3 can be computed as

$$\omega^T \mathrm{f} = \sum_{x=2}^{2h_s-1} \sum_{y=2}^{2w_s-1} A(t+x, l+y) \tag{5}$$

Therefore an integral image can be build to calculate $\omega^T f$ while without explicit acquiring f or ω. A CUDA kernel function is dedicated to compute the auxiliary image A. Again each thread block is set as 16×16 threads, with each thread processing one pixel.

Building an integral image on GPU can be decomposed into three steps, (1) conduct inclusive scan along the horizontal direction (2) transpose the scan results (3) conduct the inclusive scan along the horizontal direction, and then transpose results again. Taking advantage shared memory, both scan and matrix transpose (therefore integral image) can be computed efficiently [14, 15].

When conducting inclusive scan, each row of the image is broken into several segments, with 256 pixels contained in each segment, and with 128 threads in one thread block. The thread grid size is $n \times h$, where n is the number of segments in each row, and h is the height (or width) of the image.

When doing transpose, results are dumped into the global memory, and un-coalesced memory accessing may happen, which may degrade computing efficiency. Special attentions have to be paid to avoid this. Again shared memory is utilized. By dedicated organizing, data in non-contiguous locations in shared memory are dumped into contiguous locations in the global memory. In addition, in order to avoid bank conflicts in the shared memory, one column is padded [15].

When the integral image is ready, the score for each detection window (contain 108×36 pixels) can be calculated efficiently. Execution configuration is taken as that described in the Sobel gradient computation section.

If the score of a detection window is than less 0, then this window does not contain any pedestrian; otherwise this window possibly contains a pedestrian, and following HIK SVM classifier is applied to make for sure.

3.3 HIK SVM Classifier Evaluation

Suppose the sliding step size of the detection window is 2 pixels, for a VGA resolution (640×480) image, there are more than ten million windows. Out of these windows, only a small portion of them have non-negative scores. Using an approach known as stream reduction [16], these windows can be picked out quickly, while other windows will be dropped off.

In the HOG-based detection algorithm by Prisacariu and Reid [6], the sliding step size of detection window is exactly equal to the width or height of cell (four pixels), so histogram over each cell can be calculated once, and can be reused by each related block. In contrast, in our implementation of CENTRIST-based detection algorithm, the sliding step size of the detection window is usually smaller than the width or height of a cell, e.g., 2 pixels. Therefore histogram over a cell or block cannot be reused again when the detection window moves to the next position.

A kernel is designed for computing block histograms for those windows, which can pass the first-stage linear classifier. In terms of CUDA execution configuration, the gird size is equal to the number of blocks in all detection windows, with each thread block containing Nwarp number of warps in charge of computing the histogram of a block in the detection window. Following sample code from NVIDIA CUDA SDK, histogram calculation kernel function is designed. After trying different values, in each thread block, shared memory with a size of $256 \times 2 \times 4$ bytes is allocated. Two sub-histograms are produced in a block. The sub-histogram arrays are then combined to form the histogram over a block.

Two kernels are constructed to compute the score of the detection window. The first kernel is used to compute the score of each block in the detection window, with each thread block computing the score of a block. Using this histogram value as an index, corresponding score can be looked up from HIK SVM classifier which stays in the texture memory. Summarizing scores corresponding to those pixels which have valid census transform, the score corresponding to the block can be gotten, where parallel reduction is conducted. To maximize efficiency, shared memory is utilized, where branch divergence and bank conflict are avoided [18]. The second kernel function is in charge of computing the final score of the window. As each window contains 24 blocks, the final score can be gotten by summarizing scores of these 24 blocks. If the final score of any window is greater than 0, i.e., this window does contain a pedestrian, the coordinates of the window is stored in the global memory. As coordinates of these windows in global memory are not close each other, stream reduction is conducted again, coordinates of windows containing any pedestrian are compacted together in the global memory, and then transferred back to host PC. At the host PC, non-maximal suppression is conducted, because the same pedestrian may be detected multiple times in separated but closely nearby windows at different image scale.

4 Experiment Results

To verify the efficiency of our design, experiments are conducted using INRIA dataset [2] and BAHNHOF video sequence [19]. INRIA data set is composed of the training set and test set. Our implementation is run on a notebook machine with Intel Core i5-4200H CPU and NVIDIA GeForce GTX 950 M GPU. As our implementation exactly repeats Wu's code in terms of calculating accuracy, therefore we take over all Wu's analysis about detection accuracy of C4 algorithm, while emphasizing speed comparison. Currently C4's accuracy is slightly lower than some methods,while comparable or higher than many other methods [9].

For the purpose of comparison, the original serial version C4 code is run on the same platform. Speed comparisons over different image size are summarized in Table 1. Further tests are conducted using the BAHNHOF video sequence. The algorithm by Benenson et al. is the fastest algorithm before ours [8], so it is used as a benchmark for comparison. Benenson uses a combination of variety of techniques. One of the key features of his algorithm is the Very Fast detector. N/κ classifiers trained offline can be transferred into N classifiers by dedicated approximation technique, while image resizing computation is saved. In our implementation, images are rescaled 7 levels with a ratio of 1.25 on 640×480 image.

Table 1. Example running times for three image sizes

Implementation	320 × 240	640 × 480	1280 × 960
C4 on CPU	31.6 ms	133.4 ms	642.4 ms
C4 on GPU	2.3 ms	8.4 ms	39.0 ms

In addition, Benenson's algorithm uses scene geometry information like ground plane [20] and stixel world model (stixel ≈ sticks above the ground in the image) [21] as prior knowledge for object detection. By assuming pedestrian always stands on the ground, window searching area is largely reduced [20, 22].

Benenson's algorithm is re-run on our machine platform, and results are summarized in Table 2. Without using any prior knowledge, our implementation search pedestrians with brute force, and can reach a speed of 108 fps. Furthermore our implementation is 17 times faster than original CENTRIST implementation.

Table 2. Processing speed for BAHNHOF sequence

Implementation	Speed
ChnFtrs (baseline)	20 fps
VeryFast + ground plane	38 fps
VeryFast + stixels	98 fps
CPU C4	6 fps
Our GPU C4	108 fps

5 Conclusions

A parallel implementation of pedestrian detection algorithm is presented, which is based on census transform histogram algorithm. Our implementation achieves 17 times speedup over the original C4 implementation, i.e., 108 fps speed on 640 × 480 images. To our knowledge, our implementation outperforms Benenson's implementation in speed, therefore is the most efficient one by far. Further improvement and rigorous test of our C4 implementation are undergoing.

References

1. Gerónimo, D., López, A.M., Sappa, A.D., Graf, T.: Survey of pedestrian detection for advanced driver assistance systems. IEEE Trans. Pattern Anal. Mach. Intell. **32**(7), 1239–1258 (2010)
2. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: Proceedings of the Conference on Computer Vision and Pattern Recognition, CVPR 2005, San Diego, CA, vol. 1, pp. 886–893, June 2005
3. Wang, X., Han, T.X., Yan, S.: An HOG-LBP human detector with partial occlusion handling. In: Proceedings of the International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, pp. 32–39 (2009)
4. Dollár, P., Tu, Z., Perona, P., Belongie, S.: Integral channel features. In: Proceedings of the British Machine Vision Conference, BMVC 2009, London, United Kingdom, pp. 1–11 (2009)

5. Wojek, C., Dorko, G., Schulz, A., Schiele, B.: Sliding-windows for rapid object class localization: a parallel technique. In: Proceedings of 30st Annual Symposium of the Deutsche Arbeitsgemeinschaft fur Mustererkennung, DAGM 2008, Munich, Germany, pp. 71–81 (2008)
6. Prisacariu, V., Reid, I.: Fast HOG- a real-time GPU implementation of HOG. Technical report 2310/09, Department of Engineering Science, Oxford University (2009)
7. Lillywhite, K., Dah-Jye, L., Dong, Z.: Real-time human detection using histograms of oriented gradients on a GPU. In: Proceedings of the Workshop on Applications of Computer Vision, WACV 2009, Snowbird, Utah, pp. 1–6, December 2009
8. Benenson, R., Mathias, M., Timofte, R., Van Gool, L.: Pedestrian detection at 100 frames per second. In: Proceedings of the Conference on Computer Vision and Pattern Recognition, CVPR 2012, Rhode Island, USA, pp. 2903–2910 (2012)
9. Wu, J., Geyer, C., Rehg, J.M.: Real-time human detection using contour cues. In: Proceedings of the International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, pp. 860–867 (2011)
10. Wu, J., Rehg, J.M.: CENTRIST: a visual descriptor for scene categorization. IEEE Trans. Pattern Anal. Mach. Intell. **33**(8), 1489–1501 (2011)
11. Wu, J., Rehg, J.M.: Beyond the Euclidean distance: creating effective visual codebooks using the histogram intersection kernel. In: Proceedings of the IEEE International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, pp. 630–637 (2009). doi:10.1109/ICCV.2009.5459178
12. Maji, S., Berg, A.: Max-margin additive classifiers for detection. In: Proceedings of the IEEE International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, pp. 40–47 (2009). doi:10.1109/ICCV.2009.5459203
13. Bilgic, B., Horn, B.K.P., Masaki, I.: Efficient integral image computation on the GPU. In: Proceedings of the 2010 IEEE Intelligent Vehicles Symposium, IV 2010, La Jolla, CA, USA, pp. 528–533 (2010). doi:10.1109/IVS.2010.5548142
14. Harris, M., Sengupta, S., Owens, J.D.: Parallel prefix sum (scan) with CUDA. GPU Gems **3** (39), 851–876 (2007)
15. Ruetsch, G., Micikevicius, P.: Optimizing matrix transpose in CUDA. NVIDIA GPU Computing SDK (2009)
16. Horn, D.: Stream reduction operations for GPGPU applications. GPU Gems **2**(36), 573–589 (2005)
17. Billeter, M., Olsson, O., Assarsson, U.: Efficient stream compaction on wide SIMD many-core architectures. In: High Performance Graphics, pp. 159–166 (2009)
18. Harris, M.: Optimizing parallel reduction in CUDA (2010). http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/reduction/doc/reduction.pdf
19. Ess, A., Leibe, B., Schindler, K., Gool, L.V.: Robust multiperson tracking from a mobile platform. IEEE Trans. Pattern Anal. Mach. Intell. **31**(10), 1831–1846 (2009)
20. Sudowe, P., Leibe, B.: Efficient use of geometric constraints for sliding-window object detection in video. In: Proceedings of the 8th International Conference on Computer Vision Systems, ICVS 2011, Sophia Antipolis, France, pp. 11–20 (2011). doi:10.1007/978-3-642-23968-7_2
21. Badino, H., Franke, U., Pfeiffer, D.: The stixel world - a compact medium level representation of the 3D-world. In: Proceedings of the 31st Annual Symposium of the Deutsche Arbeitsgemeinschaft fur Mustererkennung, DAGM 2009, Jena, Germany, pp. 51–60 (2009). doi:10.1007/978-3-642-03798-6_6
22. Benenson, R., Timofte, R., Gool, L.V.: Stixels estimation without depth map computation. In: Proceedings of the 2011 IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, pp. 2010–2017 (2011). doi:10.1109/ICCVW.2011.6130495