GPU-based pedestrian detection for autonomous driving

V. Campmany[1,2], S. Silva[1,2], A. Espinosa[1], J.C. Moure[1],
D. Vázquez[2], and A.M. López[2]

[1] Universitat Autonoma de Barcelona, Spain.
[2] Computer Vision Centre (CVC), Spain.

Abstract
We propose a real-time pedestrian detection system for the embedded Nvidia Tegra X1 GPU-CPU hybrid platform. The detection pipeline is composed by the following state-of-the-art algorithms: features extracted from the input image are Histograms of Local Binary Patterns (LBP) and Histograms of Oriented Gradients (HOG); candidate generation using Pyramidal Sliding Window technique; and classification with Support Vector Machine (SVM). Experimental results show that the Tegra ARM platform is two times more energy efficient than a desktop GPU and at least 8 times faster than a desktop multicore CPU.

Keywords: Autonomous Driving, Pedestrian detection, Computer Vision, CUDA, Low consumption

1 Introduction

Autonomous driving requires perceiving and understanding the vehicle environment using sensors, providing self-localization, controlling the vehicle, and planning the routes. A pedestrian detector locating humans on a digital image is a key module that requires real-time response. The wide variation by which humans appear with different poses, clothes, illuminations and backgrounds makes the problem very hard, and the source of an active research during the last twenty years [1, 2, 3]. The recent appearance of embedded GPU-accelerated systems based on the Nvidia's Tegra X1 ARM processor, like the Jetson TX1 and DrivePX platforms, pave the way for low-cost, reduced space, and low-consumption real-time pedestrian detection.

We have designed a complete GPU-accelerated pedestrian detection pipeline [1] based on [3]. We have explored alternative parallelization schemes and data layouts and selected the most general and scalable solutions providing high performance and detection accuracy. Evaluation results proved that: (1) real-time can be reached on an embedded GPU-accelerated system with state-of-the-art accuracy (20 images of 1242×375 pixels per second); (2) GPU-acceleration provides between 8x and 60x performance speedup with respect to a baseline multi-core CPU implementation; and (3) the Tegra X1 processor at least doubles the performance per Watt of the system accelerated by a GTX 960 GPU.

[1]This research has been supported by the MICINN and MEC under contract numbers TIN2014-53234-C2-1-R and TRA2014-57088-C2-1-R, and by the spanish DGT and Generalitat de Catalunya projects SPIP2014-01352, 2014-SGR-1506 and 2014-SGR1562. We thank Nvidia for the donation of the GPUs used in this work

2 Pedestrian detection and Related work

A pedestrian detector based on computer vision is composed by four core modules: the candidate generation, the feature extraction, the classification and the refinement. We use the Pyramidal Sliding Window method for candidate generation to provide rectangular image windows which eventually contain pedestrians. We use Local Binary Patterns (LBP) [4] and Histograms of Oriented Gradients (HOG) [5] as distinctive patches or features that describe each image window. The classification stage labels the windows using a Support Vector Machine (SVM) [6] learned model accordingly to its features. Finally, a refinement stage using the Non-maximum Suppression algorithm [7] selects a unique window for each detected pedestrian.

Since the appearance of GPGPU computing, several object detection algorithms have been ported to the GPU [8, 9, 10, 11], most of them using the well-known HOG-SVM approach. They were executed on desktop GPUs and clearly outperformed a highly tuned CPU version [12]. Although FPGA designs like [13] also obtained outstanding results, the lower development costs of the CUDA programming environment and affordability of GPU cards make them more suitable for testing new algorithms. In this work, we propose a real-time pedestrian detector running on a low-consumption GPU device. We also present for the first time a GPU implementation of the HOGLBP-SVM detection pipeline [14].

3 Design and Analysis of Massively-Parallel Algorithms

In this section we present three detection pipelines combining the basic algorithms mentioned in section 2, and describe the decisions behind their massively-parallel implementations on a CUDA architecture. The detection pipelines, ordered from lower to higher accuracy and computational complexity, are LBP-SVM, HOG-SVM and HOGLBP-SVM, representing three approaches to trade off functionality with processing rate. They use different feature extraction methods, being the HOGLBP a concatenation of the single HOG and LBP feature vectors. The hybrid processing pipeline (1) copies the captured images from the Host (CPU) memory space to the Device (GPU); (2) creates the scaled-pyramid of images; (3) extracts features from each pyramid layer; (4) segments and classifies windows from each layer; and (5) copies detection results to the Host memory to refine them using Non-maximum Suppression.

3.1 Histograms of Local Binary Patterns (LBP)

LBP features give information of the texture on a small block of the image [4]. Figure 1 shows the stages needed to compute the LBP features. First, the LBP image is generated using a 2-dimensional stencil algorithmic pattern, where each pixel value is compared to its nearest neighbors, and generates a 0 or a 1 depending on the comparison result, producing an 8-bit result. Second, histograms of blocks of 16×16 pixels are generated over the LBP image. Blocks have a 50% overlap in the X and Y axis. We avoid computing 4 times the same data by calculating the histograms of smaller cells of 8×8 pixels, which are then reduced in groups of four to generate the output block histograms.

The CUDA implementation of the 2D Stencil pattern maps each thread to one output pixel, avoiding data dependencies and assuring coalesced memory accesses. The computation of LBP cell histograms uses a cooperative scatter pattern that aims for both efficient memory accesses and data reuse: each thread is mapped to an input pixel and uses atomic operations to add to its corresponding cell histogram. Block Histograms are computed using a warp-level reduction pattern. We have verified that performance scales gracefully for different image sizes.

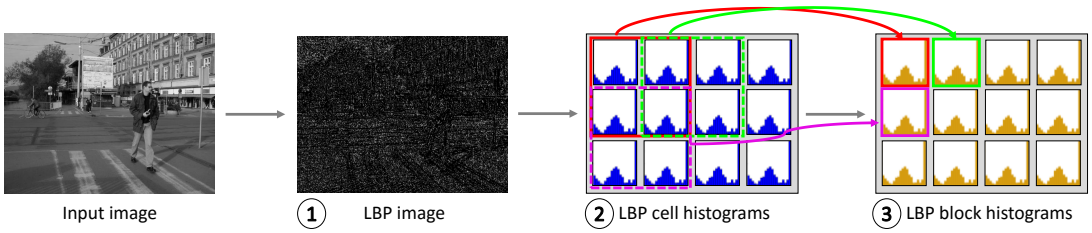

Figure 1: Generation of the LBP feature vectors

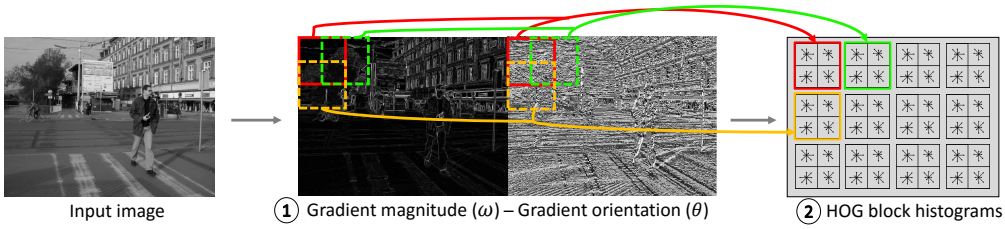

Figure 2: Generation of HOG feature vectors

3.2 Histogram of Oriented Gradients (HOG)

Histograms of Oriented Gradients count the occurrence of gradient orientation on a block of the image [5]. Figure 2 shows the stages needed to compute the HOG features. Gradient computation applies a 2D stencil pattern to measure the directional change of color in the image. The gradient of a pixel has two components: the orientation (θ), or the directional change of color, and the magnitude (ω), or the intensity of the change. Histograms are also computed on overlapped blocks of 16×16 pixels into the Gradient image, but using trilinear interpolation in order to avoid sudden changes due to aliasing effect [5]. Then, Block Histograms are composed by four concatenated 8×8-pixel Cell Histograms, instead of being a reduction of the Cell Histograms. Different bins of the Block Histogram receive a weighted value of the orientation (θ) multiplied by the magnitude (ω). Depending on the pixel coordinates, each input value can affect two, four or eight bins of the Block Histogram.

Gradient computation in CUDA maps threads to output pixels (stencil pattern), so that each thread performs coalesced memory accesses. The scattered pattern for generating the histograms involves non-coalesced accesses. After checking and discarding several cooperative strategies, all of them limited by atomic memory operations and thread divergence, we decided to map one thread to the task of computing one block histogram. Allocating histograms on the limited on-chip shared memory, even with only 25% of the GPU occupancy, avoids the cache contention of using local or global memory, and provides the best performance.

3.3 Pyramidal Sliding Window & Support Vector Machine (SVM)

A pyramid of several down-scaled copies of the input image is used to detect pedestrians of various sizes and at different distances. Every layer is split into highly overlapped regions, or *windows*, of 128×64 pixels, described with a feature vector ($\vec{x}$), which is composed by the concatenation of the HOG and LBP histograms enclosed in the given region. Then, every vector is evaluated to predict if the region contains a pedestrian or not.

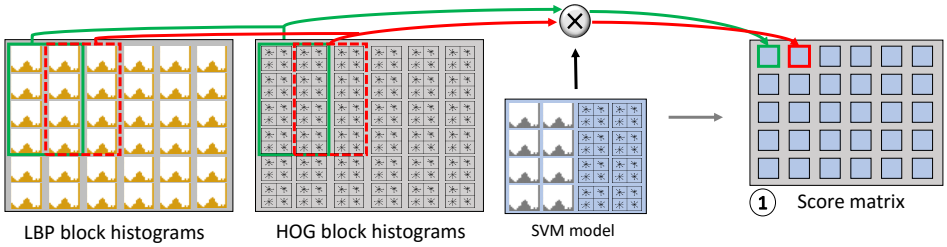

Figure 3: Sliding Window and SVM inference of the HOG and LBP features

Figure 3 illustrates the evaluation of a window using both HOG and LBP features as the image descriptors. SVM is a supervised learning method that requires an offline training to produce a model (an n-dimensional plane) that discriminates two categories, in our case pedestrians from background [6]. The distance of an input feature vector ($\vec{x}$) and the model hyper-plane ($\vec{\omega}$) is computed using the dot product operation: the window is classified as pedestrian if the distance is greater than a given threshold.

The dot product is the most time-consuming part of the pipeline. The CUDA design uses warp-level cooperation, with each warp responsible of a window ($\vec{x}$) of the transformed image. Threads in the warp compute the dot product of different vector segments and then the partial results are reduced, using register shuffle for communication. The warp-level approach avoids the overhead of explicit thread synchronization, and also allows full utilization of the memory bandwidth with coalesced memory accesses.

4 Experiments & Results

Figure 4 shows the performance (measured in frames processed per second or FPS) of a multi-threaded CPU baseline version (Intel i7-5930K) and two GPU-accelerated versions (Nvidia GTX 960 and Tegra X1) of the whole detection pipelines (LBP-SVM, HOG-SVM and HOGLBP-SVM), using 12 pyramid layers and for a video sequence with an image size of 1242×375 pixels. The performance under the low-consumption ARM platform, between 20 and 40 FPS, can be considered real-time.

Figure 5 illustrates the detection miss rate (non-detected cases) as a function of the false positives per image or FPPI (windows wrongly classified as pedestrians). As the FPPI increases, the miss rate decreases, leading to a more tolerant system. The area below the curves is shown as legend values: the lower, the more reliable is the detector. The HOGLBP-SVM pipeline achieves state-of-the-art accuracy, while the simpler pipelines achieve slightly lower accuracy but, on the other hand, demand less computational power to achieve real-time performance, which makes them suitable for less powerful GPUs.

The GTX 960 is a desktop GPU designed to provide very high performance with good power efficiency, while the Tegra X1 embedded system, though, is intended to operate in constrained environments and power consumption is a high concern. The power efficiency has been measured in $FPS/Watt$, assuming the Watt consumption is the Thermal Design Power (TDP) provided by the manufacturer company. Using the HOGLBP-SVM pipeline, the most accurate, the Tegra X1 platform reaches 2 $FPS/Watt$, which outperforms the CPU-only implementation by $200\times$ and doubles the efficiency of the GTX 960 desktop GPU.

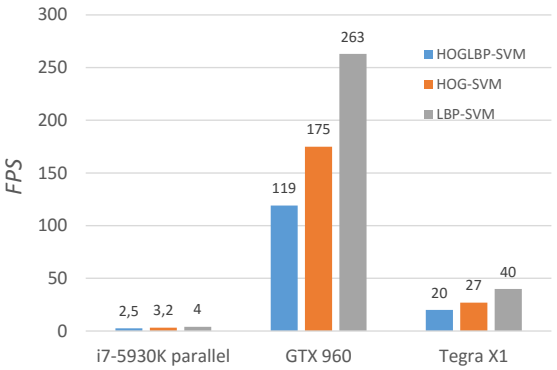

Figure 4: Detection pipelines performance

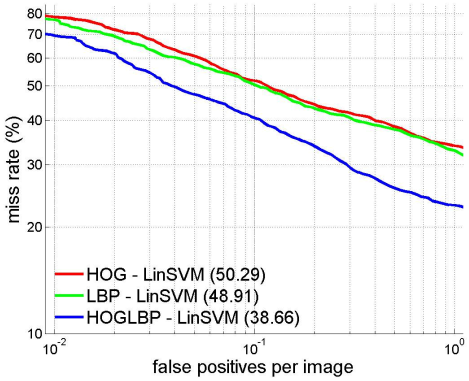

Figure 5: Objective term (lower is better)

5 Conclusions

We present a massively parallel implementation of a pedestrian detector that achieves the real-time requirements of autonomous driving using the Nvidia DrivePX platform. Algorithms must be adapted to the GPU architecture: smart work distribution and thread collaboration are key factors to attain significant performance improvements, which become even more critical when the target is the low consumption Tegra X1 processor. Experimental results show that the Tegra ARM platform is two times more energy efficient than a desktop GPU when running our massively parallel algorithms.

References

[1] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf. Survey of pedestrian detection for advanced driver assistance systems. In *PAMI*, 2010.

[2] D. M. Gavrila. The visual analysis of human movement: A survey. In *CVIU*, 1999.

[3] J. Marin, D. Vazquez, A. M. Lopez, J. Amores, and B. Leibe. Random forests of local experts for pedestrian detection. In *ICCV*, 2013.

[4] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. In *PAMI*, 2002.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[6] C. Cortes and V. Vapnik. Support-vector networks. In *Machine learning*, 1995.

[7] Laptev. Improving object detection with boosted histograms. In *Image and Vision Comp.*, 2009.

[8] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Pedestrian detection at 100 frames per second. In *CVPR*, 2012.

[9] C. Wojek, G. Dork, A. Schulz, and B. Schiele. Sliding-windows for rapid object class localization: A parallel technique. In *Pattern Recognition*, 2008.

[10] Zhang and Nevatia. Efficient scan-window based object detection using GPGPU. In *CVPR*, 2008.

[11] V. A. Prisacariu and I. Reid. fastHOG - a real-time GPU implementation of HOG. In *Technical Report*, 2009.

[12] P. Dollar, Belongie, and Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010.

[13] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll. FPGA-based real-time pedestrian detection on high-resolution images. In *CVPR*, 2013.

[14] X. Wang, T. X. Han, and S. Yan. An HOG-LBP human detector with partial occlusion handling. In *ICCV*, 2009.