Implementing the Local Binary Patterns with SIMD Instructions of CPU

Roman Juránek, Pavel Zemčík, Adam Herout
{ijuranek,zemcik,herout}@fit.vutbr.cz
Graph@FIT
Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology, Brno, Czech Republic

ABSTRACT

Usage of statistical classifiers, namely AdaBoost and its modifications, is very common in object detection and pattern recognition. Performance of such classifiers strongly depends on low level features they use. This paper presents an experimental implementation of the Local Binary Patterns (LBP) that uses SIMD instructions for acceleration. The experiments shows that the proposed implementation is about six times faster than the plain C implementation (i.e. with no special optimizations) and superior to optimized implementations of features with similar descriptive power.

Keywords: LBP, AdaBoost, Object Detection, Feature Extraction, SIMD, SSE, CUDA

1 INTRODUCTION

Object detection in images and video sequences has wide range of applications. Several object detection methods exist; however, one of the best available methods today is exploitation of statistical classifiers. The statistical classifiers are able to distinguish an object from non-object in a small window. To detect the object in image data or video, it is necessary to scan the image or video frame and apply the classifier to each possible window location within the scanned image. The classifier works with low-level features extracted from the classified image window. The features are usually simple functions of selected pixels from the window. The design of the features significantly affects the performance of the classifier and its speed and thus the speed of detection.

The Local Binary Patterns (LBP) [7] discussed in this paper are widely known to be good features to describe local areas. They are frequently used in texture analysis and segmentation. The recent studies show that they are usable as features for classification as well.

In this paper, the high performance implementation of LBP feature extraction is introduced. The implementation exploits the SIMD instructions (namely SSE) available on contemporary CPUs. The implementation is comparable with similar implementations of other feature types – Local Rank Differences (LRD) [13] and Local Rank Patterns (LRP) [6]. For the experiments WaldBoost (a modification of AdaBoost [11]) training algorithm [9] has been used.

2 RELATED WORK

The interest in fast feature extraction has been earlier pursued by other researchers because the extraction speed influences the overall performance of the object detection. Efficient implementation is thus necessary for practical applications of object detection.

Innovative approach that constituted a breakthrough in real-time object detection was taken by Viola and Jones [11]. They converted the input image into an integral representation which allows for calculation of Haar features in constant time.

There has also been much effort to implement the AdaBoost based object detection on hardware platforms like FPGA or ASIC chips [12, 10, 5]. Typically they use traditional Haar features. Recently there were proposed implementations of object detection with Local Rank Differences image features (LRD) on GPU [8] and CUDA [2] which employs resources of modern graphics cards to accelerate feature extraction. The disadvantage of this approaches is that they still need special hardware.

In our previous work, we introduced high performance implementation of the Local Rank Differences [1] and Local Rank Patterns [4] image features. These implementations exploited the SIMD instructions of Intel CPU. This work shares same framework with the mentioned implementations of LRD and LRP.

3 LOCAL BINARY PATTERNS

Local Binary Patterns [7, 14], in their basic form, capture information about local textural structures through thresholding samples from local neighborhood by its central value. From the thresholded values a pattern code is formed such that each sample is represented by a single bit (Figure 1). The image pixels or response of some sampling function can be used as the samples. In many applications, circular neighborhood with 8 samples is used (8 bit pattern).

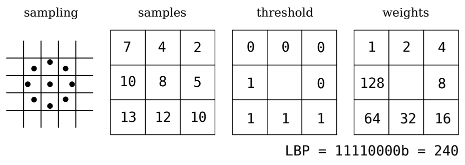

Figure 1: Evaluation of the LBP feature. First the samples are taken from the image. The values are then thresholded by the central value.

In this form, the feature response is dependent on the feature orientation. In some applications, such as texture classification or image segmentation, the rotational invariance is needed. In this case, the feature response is normalized (e.g. by bit shifting).

In our approach, the feature consists from 3×3 regularly spaced rectangular samples. The classifier can hold feature instances of different sizes. However, for practical reasons the size of a sample is constrained to maximum of 2×2 pixels. And therefore four possible samplings exist as shown in the Figure 2. This constraint, as we show in [4], does not introduce any reduction of classifier precision.

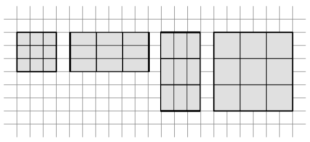

Figure 2: Possible configurations of LBP features.

4 LBP EVALUATION

The traditional way of the feature evaluation is to gather samples from the input image and sequentially construct the LBP code bit by bit. The evaluation in the proposed implementation is optimal in two ways. First, it minimizes number of memory accesses by using precalculated image representation – interleaved convolution image [1, 4], which allows for fast retrieval of necessary data. Second, the evaluation takes advantage of the SIMD instructions by processing all values of the feature in parallel manner.

4.1 The SSE Instruction Set

Before the actual implementation of the LBP feature evaluation is described, let us briefly characterize the SSE instruction set on Intel CPUs. Unlike classic x86 instructions where an operation is executed on one piece of data at the time, the SSE set provides means to execute one operation over multiple data (as shown in the Figure 3). Therefore the main attribute of the SSE instruction set is parallel processing of data.

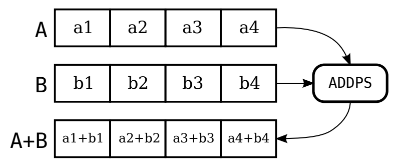

Figure 3: Operation of SSE instruction `ADDPS`. The `A` and `B` are considered to be vectors of four `float` values.

The SSE set was introduced in 1999 as an extension to x86 and MMX instruction set. The instructions works with 128 bit wide registers which can hold a vector of values (e.g. 4× 32 bit float, 16× 8 bit integer, etc.).

4.2 Image Preprocessing

As was previously stated, in our approach we use rectangular samples as an input for feature evaluation. In the preprocessing stage, images convolved with all possible shapes of samples are created. This images are later used as source of samples. Four images are created in our case as the size of the samples is restricted to maximum of 2×2 pixels.

Every convolved image is arranged in a manner that four consequent pixels in the memory (i.e. 32 bit word as we use 8 bit images) correspond to 2×2 adjacent convolution responses. This memory layout requires that each convolved image is logically divided into blocks where each block contains sub-sampled image convolved with same modulo position of convolution kernel. Number of blocks is determined by $w \times h$ where w and h is width and height of the convolution kernel. This is schematically shown in Figure 4 where preprocessing of image with 2×1 pixel kernel is displayed and two memory blocks are formed. Images preprocessed with other kernels are created analogically. This layout ensures that data for each feature are placed in same block and can be obtained by two memory accesses.

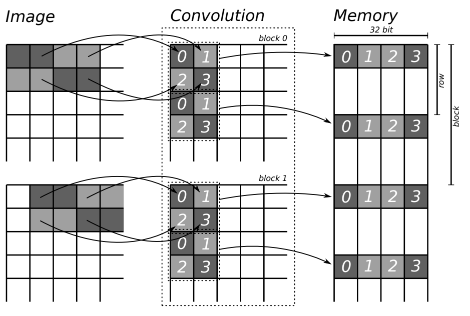

Figure 4: Preprocessing of image with 2×1 pixel kernel.

Note that the feature evaluation itself is independent on the choice of the convolution kernels. The rectangular shape was selected because the implementation of convolution and memory rearrangement can be very highly optimized.

4.3 The Evaluation

The framework of evaluation of the features is displayed in the Figure 5. The input image is first preprocessed – the convolution images are created. Each feature is parametrized by a position in the image and size of samples. The convolution image is selected according to the size of samples and the block and address within the block is determined from the feature position.

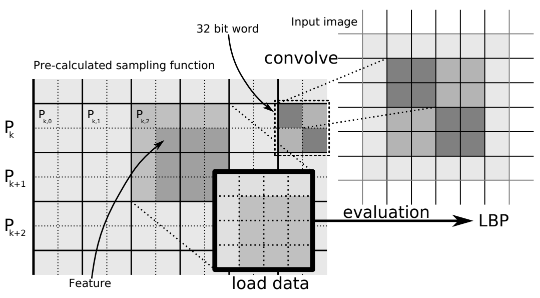

Figure 5: Schematic view of the image preprocessing and LBP evaluation. In this case a feature with 2×2 pixel samples is used.

By loading eight 32 bit aligned pixels from two subsequent rows of a convolution image (i.e. two 64 bit reads), the 4×4 responses of the sampling function are effectively loaded. The feature data is then located in a 3×3 sub-window of this data (feature shift). Note that the feature evaluation code is *independent* on the size of the feature as the data are loaded from precalculated representations.

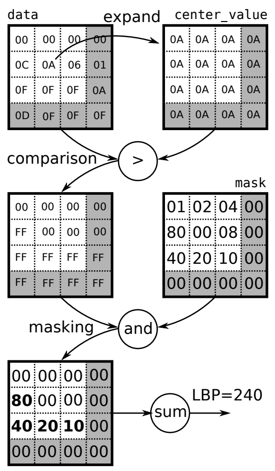

Figure 6: Block diagram of LBP evaluation.

Figure 6 shows the feature evaluation step by step. The corresponding code is displayed in the Figure 7. Every grid represents a 128 bit SIMD register holding sixteen 8 bit values loaded from the pre-calculated sampling function. First, the central value (defined by feature shift) is expanded to the full register width and compared with all other values. The comparison result then serves as a mask for vector of weights (again selected according to the feature shift). The masked weights (i.e. weights for which the comparison resulted in *true*) are then summed up producing the LBP value. Note that, although the comparison can result in *true* on positions that do not belong to the feature, this will not influence the result as the weights are always *zero* for them.

```
union {
  __m128i q;
  signed short ss[8];
} result = {
  _mm_sad_epu8(
    _mm_and_si128(
      lbpWeights[shift].q,
      _mm_cmpgt_epi8(
        data,
        _mm_set1_epi8(*center))),
    zero)
};
int lbp = result.ss[4] + result.ss[0];
```

Figure 7: The actual evaluation code (with Intel intrinsic functions). The `lbpWeight[shift]` selects one of four possible weight vectors, `data` is vector of the data and `center` is the pointer to feature central value.

5 RESULTS

The experiments were conducted on a PC with Intel Core i7 CPU (eight core), 4 GB DDR3 and CUDA capable ASUS NVidia ENGTX 280 graphics card.

The proposed implementation was compared to experimental implementation of LBP on CUDA architecture [3] and to the implementation with no special optimizations (which we refer to as *Plain C*). And also compared to implementations of the LRD and LRP extractors which shares same evaluation framework.

The implementations were tested on the task of multiscale face detection. In the first test, we compare the real detection performance in terms of processed frames per second on a long video (taken from public TV broadcasting).

$\alpha = 0.1$	$560 \times 240px$	$720 \times 576px$	$1280 \times 720px$
SIMD	**61**/58/57	**22**/20/20	**10/10/10**
CUDA	**88**/73/69	**68**/56/54	**27**/24/23
Plain C	**8**/6.4/5.6	3.4/**3.5**/2.8	**1.4**/1.3/1.1

$\alpha = 0.2$	$560 \times 240px$	$720 \times 576px$	$1280 \times 720px$
SIMD	**87**/82/81	**28**/24/24	**13**/11/11
CUDA	**115**/91/89	**82**/63/61	**31**/27/26
Plain C	**12**/10.4/9.6	**4.5**/4/3.7	**1.9**/1.6/1.5

Table 1: Object detection performance in *frames per second* on different architectures with usage of different feature extractors. The values in the table are ordered in following way: LBP/LRP/LRD.

Results in Table 1 display performance of processing of three videos with different resolutions ranging from low resolution to 720p HD video. Two classifiers with different target error rate (α) were used. The experiment shown that the SIMD implementation outperforms the Plain C. On the low resolutions the SIMD almost reaches the performance of the multiprocessor CUDA implementation athought on the high resolutions the CUDA is almost three times faster.

	$560 \times 240px$	$720 \times 576px$	$1280 \times 720px$
SIMD	0.32	0.98	2.1
CUDA	0.29	0.43	0.82
Plain C	2.5	7.8	17.5

Table 2: Preprocessing performance [ms/frame]

The Table 2 shows comparison of preprocessing on the three architectures. The preprocessing includes image scaling to build image pyramid. Additionally, in the SIMD implementation, each pyramid level is convolved according to description in the Section 4.

6 CONCLUSION AND FUTURE WORK

The goal of the work presented in this paper was to efficiently implement the Local Binary Patterns feature extractor in the contemporary CPUs using the advanced SIMD instructions. The goal was fulfilled and high performance feature extractor was implemented and experimentally evaluated.

The extractor needs preprocessing stage which prepares convolution images with precalculated sampling functions. In the evaluation stage, each feature needs only two memory accesses to load necesarry data and the data are then processed by SIMD instructions to calculate the LBP feature response.

The results show that the SIMD extractor is faster than the Plain C extractor by factor of 6.5 and almost reaches to performance of implementation on CUDA architecture. And the speed of LBP evaluation is comparable or better than evaluation of feature types as LRD and LRP. Although the CUDA implementation is faster, it needs special hardware which supports the CUDA architecture. On the other hand, the SIMD instructions are common in all contemporary CPUs which makes the SIMD implementation suitable for larger range of applications. Espetially for embeded systems.

The future work includes optimization of feature evaluation on bulk data, for example evaluation of several independent features in parallel or simultaneous evaluation of consequent features. The future work also includes further dataflow optimizations.

ACKNOWLEDGEMENTS

This work was supported by Czech Ministry of Education, Youth and Sports research project Center of Computer Graphics (LC06008).

REFERENCES

[1] Adam Herout, Michal Hradiš, Roman Juránek, and Pavel Zemčík. Implementation of the "local rank differences" image feature using simd instructions of cpu. In *Proceedings of Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, page 9, 2008.

[2] Adam Herout, Radovan Jošth, Pavel Zemčík, and Michal Hradiš. Gp-gpu implementation of the "local rank differences" image feature. In *Proceedings of International Conference on Computer Vision and Graphics 2008*, Lecture Notes in Computer Science, pages 1–11. Springer Verlag, 2008.

[3] Adam Herout, Radovan Jošth, Pavel Zemčík, and Michal Hradiš. Gp-gpu implementation of the "local rank differences" image feature. In *Proceedings of International Conference on Computer Vision and Graphics 2008*, Lecture Notes in Computer Science, pages 380–390. Springer Verlag, 2008.

[4] Adam Herout, Pavel Zemčík, Michal Hradiš, Roman Juránek, Jiří Havel, Radovan Jošth, and Martin Žádník. *Low-Level Image Features for Real-Time Object Detection*, page 25. IN-TECH Education and Publishing, 2009.

[5] M. Hiromoto, K. Nakahara, H. Sugano, Y. Nakamura, and R. Miyamoto. A specialized processor suitable for adaboost-based detection with haar-like features. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[6] Michal Hradiš, Adam Herout, and Pavel Zemčík. Local rank patterns - novel features for rapid object detection. In *Proceedings of International Conference on Computer Vision and Graphics 2008*, Lecture Notes in Computer Science, pages 1–2, 2008.

[7] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, 2002.

[8] Lukáš Polok, Adam Herout, Pavel Zemčík, Michal Hradiš, Roman Juránek, and Radovan Jošth. "local rank differences" image feature implemented on gpu. In *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Lecture Notes In Computer Science; Vol. 5259, pages 170–181. Springer Verlag, 2008.

[9] Jan Sochman and Jiri Matas. Waldboost – learning for time constrained sequential detection. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 150–156, Washington, DC, USA, 2005. IEEE Computer Society.

[10] T. Theocharides, N. Vijaykrishnan, and M.J. Irwin. A parallel architecture for hardware face detection. In *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, volume 00, pages 2 pp.–, March 2006.

[11] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:511–I–518 vol.1, 2001.

[12] Yu Wei, Xiong Bing, and C. Chareonsak. Fpga implementation of adaboost algorithm for detection of face biometrics. In *Biomedical Circuits and Systems, 2004 IEEE International Workshop on*, pages S1/6–17–20, Dec. 2004.

[13] Pavel Zemčík, Michal Hradiš, and Adam Herout. Local rank differences - novel features for image. In *Proceedings of SCCG 2007*, pages 1–12, 2007.

[14] Lun Zhang, Rufeng Chu, Shiming Xiang, ShengCai Liao, and Stan Z. Li. Face detection based on multi-block lbp representation. In *ICB*, pages 11–18, 2007.