Parallel implementation of Local SVD Binary Pattern for real-time Background Subtraction using CUDA

Alexis Mendoza Villarroel, Crhistian Turpo Apaza, Diego Amable Romero
Escuela de Ciencia de la Computación
Universidad Nacional de San Agustín
Arequipa, Perú

Abstract— **Background subtraction is a technique used in computer vision and image processing which is applied to detect moving objects for further processing. Most of the techniques rely on color and texture feature, being susceptible to illumination and noise. Local SVD Binary Pattern proposes an adaptive background subtraction model invariant to illumination variation, noise and shadows. Although this technique shows effective results, a serial implementation of it makes impossible the task of background subtraction in real-time due to its computational cost.**

A method for the parallelization of this technique is proposed by using GPU threads in CUDA. We implement the parallel version of each of the calculations needed and process the frames in batches to prevent a bottleneck in the use of memory bandwidth. Experimental results achieve up to 28x speedup for the processing of videos of different resolutions, showing that a real-time background subtraction is possible now with this approach.

Keywords—LSBP, Background subtraction, GPU, CUDA, batched SVD.

I. Introduction

Background subtraction is a process for segmenting objects in the foreground of an image. Given its important role on further analysis of content of videos, it has been a topic of interest for many researchers in the last decade. Due to the challenges that this process presents, such as illumination variance, camera movement, and the computational power that it needs, many approaches had been proposed over the years in order to solve these complications.

As a preprocessing step, an accurate subtraction of the background can significantly improve the performance of the entire application. *Local SVD Binary Pattern* (LSBP) is a method proposed in [1] which describes the features of an image while being invariant to illumination changes and noise. Despite the effectiveness of this method, it is natural that there are challenges respect to its computational complexity, as many calculations are performed in the processing of each frame in the video. The use of this effective method in many applications requires a real-time processing capability, for which the serial implementation of the technique is not an option. For this reason, in the present work we propose the parallelization of this approach by using GPU through the CUDA Toolkit framework.

Modern graphic processing units use an architecture *Single Instruction, Multiple Data* (SIMD). To obtain the maximum speedup in GPU, generated memory access requests must be treated carefully, as well as correct mapping of the threads to avoid divergence within the processing units of the device. CUDA (Compute Unified Device Architecture) is an interface that provides many facilities for modern GPUs. We use memory-efficient algorithms for the GPU to obtain more efficient calculations. We optimize the accesses to global memory in order to minimize data-access time, which helps to improve the overall performance significantly.

Section II explains the basis of the techniques to be used for the proposed background subtraction method, being necessary to understand the computational challenges that it presents. Once these principles are introduced, Section III refers to the general scheme used for the CUDA parallelization of the chosen method. In Section IV the results of the proposed implementation for different video resolutions are shown, while comparing it with the execution of the serial implementation. Finally, in Section V the conclusions of the work developed are presented, and in Section VI we talk about the possible future optimizations and paths to take once this progress has been made.

II. Background

Local SVD Binary Pattern (LSBP) is a method proposed in [1] to describe features of an image, with the aim of making easier the background subtraction by an adaptative segmentation model that uses several frames from a video. LSBP uses a combination of concepts of *Local Binary Pattern* (LBP) [2] and the decomposition into singular values (*Singular Value Decomposition - SVD*) to generate an invariant descriptor for the light's intensity, and resistant to noise and shadow variations in the image.

II-A. Singular Value Decomposition (SVD)

SVD is a mathematical concept that can be used to analyze matrices. In [1] the neighbors of each pixel in an image are considered to create a 3×3 dimensional matrix and then perform the decomposition of their singular values. Once obtained, the two lowest values found are normalized respect to the greater of the values for that matrix and added together. This value is calculated for each of the pixels in the image, and roughly describes the significant edges of the image. Inspired by [3], it is proved that this found value is invariant to the illumination, it is an illuminated area, in twilight or in umbra.

$$B(x,y) = U\Sigma V^T \tag{1}$$

$$g(x,y) = \sum_{j=2}^{3} \frac{\lambda_j}{\lambda_1} \tag{2}$$

In the equation (1) we can see the decomposition of a block centered on the pixel (x, y), where Σ is the diagonal matrix that contains the singular values in descending order, U and V are orthogonal matrices, In (2) we find the descriptive coefficient for the pixel (x, y), where λ_j are the elements of the diagonal of the Σ matrix. Knowing this invariant to the illumination, it is used to extend the concept of LBP to LSBP.

II-B. Local SVD Binary Pattern (LSBP)

The principle of LBP is, given a radius, to compare a pixel (x, y) with its neighbors and verify if they are similar or not. Doing this for each neighbors, we can generate a bit string, where 0 will be placed if they are similar, and 1 in the opposite case. While LBP uses the intensity of each pixel for this comparison, the coefficient found by the equation (2) can be used to no longer depend on the illumination changes.

$$LSBP(x,y) = \sum_{p=0}^{p-1} s(i_p, i_c)2^p \tag{3}$$

$$s(i_p, i_c) = \begin{cases} 0 & \text{if } |i_p - i_c| \leq \tau \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

In the equation (3) we find the bit string mentioned above (quantized as an integer), where i_c and i_p are the values found by the equation (2) for the pixel (x, y) and the neighbor p that is being evaluated, respectively. In the equation (4), a τ filter is being used, for which the paper [1] recommend a value of 0,05.

II-C. Modeling Background using Local SVD Binary Pattern

For the segmentation between what belongs to the background or foreground, we use the model proposed in [4] and [5]. The advantage of using this models is that they adapt to the LSBP descriptors. As described in [4], to classify each pixel we use the current frame and the background model $B(x, y)$. In this method, the model $B(x, y)$ will be consist of a N number of recent samples, contemplating a model by the intensity of the pixel $BInt(x, y)$ and a model for the LSBP $BLSBP(x, y)$.

$$BInt(x,y) = BInt_1(x,y), ..., BInt_i(x,y), ..., BInt_N(x,y)$$

$$BLSBP(x,y) = BLSBP_1(x,y), ..., BLSBP_i(x,y), ..., \\ BLSBP_N(x,y)$$

As specified in [1], for the classification of each pixel as part of the background or not, we compare each pixel in the position (x, y) with the N samples of its respective background model. To do this, we use a threshold $R(x, y)$, in addition to a global threshold H_{LSBP}. For more dynamic areas, $R(x, y)$ will have a higher value than for more static regions.

$$\begin{aligned} H(LSBP(x,y), BLSBP_i(x,y)) \leq H_{LSBP} \&\& \\ dist(Int(x,y), BInt_i(x,y)) < R(x,y) \end{aligned} \tag{5}$$

The logical value obtained in the equation (5) indicate whether we obtain a *match* or not, where H is the distance of Hamming and $dist$ refers to a Euclidean distance. Once a minimum number of *matches* $\#min$, is obtained, the comparison with the models is stopped and classified as part of the background. Otherwise, it is considered that it does not belong to the background , but to the foreground. In [1] it is proposed to set the value $\#min = 2$, being a balance between resistance to noise in the image and computational complexity.

To update the models we use a parameter $T(x, y)$ as in [1]. A conservative update update for the models is made (only of the pixels that have been classified as background), using a probability $p = 1/T(x, y)$. An update of the model means that the values of the models $BInt(x, y)$ and $BLSBP(x, y)$ will be replaced by the values of the current pixel $Int(x, y)$ and $LSBP(x, y)$, respectively. Likewise, the probability p is used to evaluate the update of one of the eight neighbors of the pixel (x, y).

The parameters $R(x, y)$ and $T(x, y)$, like models, must be updated as the classification is executed. These changes are given using a value $\bar{d}_{min}(x, y)$, which measure the dynamics of the fund of that pixel. Similar to the model mentioned above $B(x, y)$, we store a $D(x, y)$ model of minimum decision distances:

$$D(x,y) = D_1(x,y), ..., D_i(x,y), ..., D_N(x,y)$$

When an update of $B_i(x, y)$, is executed, the minimum distance $d_{min}(x, y) = min_i dist(Int(x, y), BInt(x, y))$ is calculated, which is saved in the sample $D_i(x, y)$. In this way a history of the minimum distances is created. Then, the calculation of the values $\bar{d}_{min}(x, y)$ is simply the average of the generated history, that is:

$$\bar{d}_{min}(x,y) = \frac{1}{N} \sum_{i=1}^{N} D_i(x,y) \tag{6}$$

Once this value has been calculated for each pixel, we proceed to update the parameters $R(x, y)$ and $T(x, y)$ using the following formulas:

$$R(x,y) = \begin{cases} R(x,y) \cdot (1 - R_{inc/dec}) & \text{if } R(x,y) > \bar{d}_{min}(x,y) \cdot R_{scale} \\ R(x,y) \cdot (1 + R_{inc/dec}) & \text{otherwise} \end{cases} \tag{7}$$

$$T(x,y) = \begin{cases} T(x,y) + \frac{T_{inc}}{d_{min}(x,y)} & \text{if } (x,y) \text{ is foreground} \\ T(x,y) - \frac{T_{dec}}{d_{min}(x,y)} & \text{otherwise} \end{cases} \tag{8}$$

Where the values $R_{inc/dec}$, R_{scale}, T_{inc} and T_{dec} are predefined [4].

III. Parallelization of the method

Due to the limitation of CPU to GPU memory transfer latency, we keep the models per pixel and the parameters per pixel in the global memory of the device. This way we avoid the bottleneck on memory transfer. To keep all the calculations in the device, we have to copy all the frame intensities in the device's global memory. To do this, we will work on batches of frames, so that while we perform all the calculations in one batch, another batch is moved to the global memory on the device by another thread.

The parallelization of the LSBP method leads to the parallel implementation of each of the calculations referred to in Section II, that is, the calculation of the values SVD and LSBP for each pixel of each frame, as well as the update of the models $B(x,y)$, and the parameters $R(x,y)$, $T(x,y)$ and $D(x,y)$. This also involves the initialization of all the mentioned models and parameters, maintaining the highest fidelity to the pseudocode presented in [1].

In order to reduce the bottleneck on memory transfer from CPU to GPU, our method propose the processing of the frames in batches of fixed size, so while a batch is being copied to device's global memory another batch is being processed, this will reduce the limitation given by the memory band width.

In general, in each executed kernel, a thread will be used for each pixel of the current frame, divided by two-dimensional blocks, calculating the number of blocks needed to be able to process the entire frame.

It is necessary to differentiate the sequential and parallelizable parts of the proposed method. Naturally, the calculation of SVD and LSBP values for each pixel is easily parallelizable, however, there is a sequential dependency between them (as described in Section II, the calculation of LSBP values requires the values obtained through SVD). The update of the models of the pixels $B(x,y)$, as well as the parameters $R(x,y)$, $T(x,y)$ and $D(x,y)$ are also parallelizable for each pixel of a frame, however, the update of each of them must be executed in a sequential manner between frames, respecting the order in which they are presented in the video. This is due to the nature of the algorithm for updating the LSBP model, which best differentiates background and foreground over time.

The pseudo-code 1 shows the general scheme of the procedure to process a frame, as well as the update of the $B(x,y)$ models and the $R(x,y)$ parameters , $T(x,y)$ and $D(x,y)$. Before the execution of the code, it is assumed that the array of intensities of the pixels of the corresponding frame is located in d_Int, in the memory of the device. The check in line 3 serves to keep the value of the frame to be processed within the range of the batch. When this exceeds the range, it is assumed that a new batch will already be processed. For the kernels of SVD and LSBP, each thread calculates the values for the corresponding pixel, using information from neighboring pixels. Subsequently, the pixels are classified as foreground or background on line 14, and the results are stored for each pixel in the boolean array d_Class . Given the sequential dependence of these three kernels, we use the $cudaDeviceSynchronize()$ function as a barrier to the threads.

While most of these functions can be easily parallelize there are two main functions that are the heart of this algorithm. These are the SVD and LSBP per pixel. They are not only the most important but also the most computationally expensive. We will briefly explain the solutions proposed to these problems.

```
 1 void BackgroundSubstractor::Step()
 2 {
 3   if(frame >= frames_Int->size())
 4     frame = 0;

 6   SVDKernel <<<dimGrid, dimBlock>>> (d_Int, d_SVD
        , height, width);

 8   cudaDeviceSynchronize();

10   LSBPKernel <<<dimGrid, dimBlock>>> (d_SVD,
        d_LSBP, height, width);

12   cudaDeviceSynchronize();

14   ClassificationKernel <<<dimGrid, dimBlock>>> (
        d_BInt, d_BLSBP, d_Int, d_LSBP, d_Class,
        d_R, height, width, N, min_matches, H);

16   cudaDeviceSynchronize();

18   UpdateBDKernel <<<dimGrid, dimBlock>>> (d_BInt,
        d_BLSBP, d_D, d_d, d_T, d_Int, d_LSBP,
        d_Class, height, width, N);
19   UpdateRKernel <<<dimGrid, dimBlock>>> (d_R, d_d
        , height, width, Rscale, Rincdec, Rlower);
20   UpdateTKernel <<<dimGrid, dimBlock>>> (d_T,
        d_Class, d_d, height, width, Tinc, Tdec,
        Tlower, Tupper);

22   frame++;
23 }
```

Code 1. Code in C++ with CUDA extension for the processing of one frame and the update of the models and parameters.

Finally, the kernels that will update the models and parameters are called. A single kernel is used to update the model $B(x,y)$ and the parameters $D(x,y)$ due to the dependency between them (only $D(x,y) is updated$) if any sample $B_k(x,y)$ has been updated). The use of a barrier for the threads is not necessary due to the independence of the parameter calculations.

III-A. LSBP per Pixel

As mentioned in Section II, we need to perform the local similarity binary pattern for each pixel. In our method we will let each thread perform the LSBP for each thread. We first tried doing it using only global memory, then we tried using the technique of tiling in shared memory. We will see later the results of these two approaches.

III-B. Batched SVD

Most of the problem of parallelizing this method is due to the calculation of the svd per pixel. In order to solve this problem we tried different ways to reduce its time cost. We first try calculating all the SVDs on CPU secuantially by using a library. We also tried to do it on device by using the library cuSolver and its batched functions that relies on the use of streams.We finally tried our own method wich is implementing the SVD algorithm from scratch with device code, having each thread perform the svd per pixel.

IV. Experimental Results

We tested our method with several security cameras feeds available on *Kitware Data*, a cloud storage platform. We first compare the execution times of the proposed approach and the existing serial method, showing a significant improvement on the efficiency of the processing of the frames. Intermediate knowledge in CUDA programming were needed for the development of said improvement and we believe that there is even more room for optimization with more advanced techniques. We also compare the frames per second processing capability of each implementation.

IV-A. *LSBP per thread results*

The results here shows the difference between our three aproachs, we tested with different frame size and we measured time in seconds ,we can see that initial approach of working only with global memory is better than using shared memory. The reason of this is because there is not so much overlap between the data that the threads share.

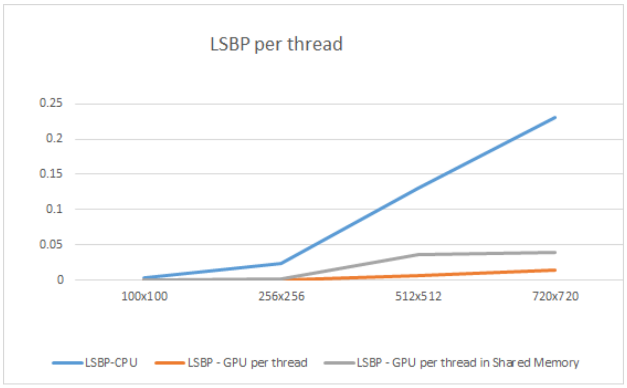

Fig. 1. LSBP per thread

IV-B. *Batched SVD results*

As in LSBP we tested with different frame size and we measured time in seconds. Here our approach of having each thread performing a svd showed better results than using CPU and using cuda libraries.

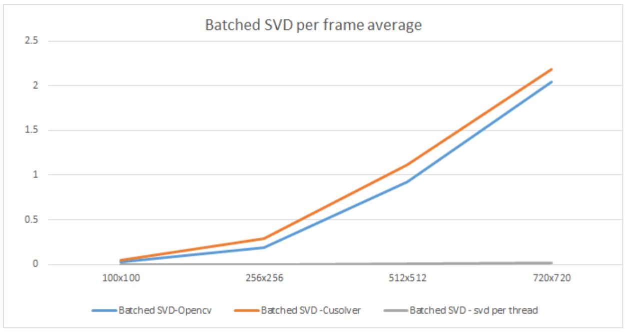

Fig. 2. Batched SVD

IV-C. *Test details*

For the testing, we used an NVidia GeForce GTX 1060 with 6 GBs of memory, 192-bit memory bandwidth, warp size of 32 threads, with a maximum of 2048 threads per multiprocessor, a maximum of 1024 threads per block and 5 Streaming Multiprocessors. With this specifications, we opted using a block size of 16×16 threads for the frame processing, maximizing the use of the multiprocessors. The CPU used was an Intel Core i7-7700HQ with 2.80 GHz of clock rate. Memory size of the RAM was 12 GBs.

The tests were performed with videos of different resolutions (a *resize* operation was made): 720×720, 512×512, 256×256 and 100×100.

IV-D. *Frames per Second Results Table*

Resolution	Serial	Parallel
720x720	0.4364 fps	12.4533 fps
512x512	0.9754 fps	24.1729 fps
256x256	3.7835 fps	93.2849 fps
100x100	25.7645 fps	441.3711 fps

TABLE I
Comparison of frames processed per second for both serial and parallel implementations

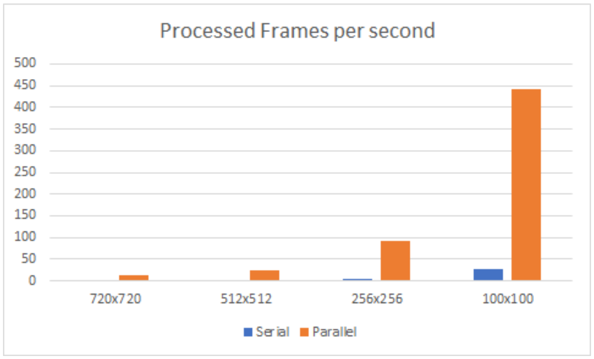

Fig. 3. Frames processed per second for both serial and parallel implementations

As shown in the Table I, the results on the parallelization method give a noticeable improvement on the frame processing, achieving more than 30 frames per second (which is the usual feed rate of a camera) with a video resolution of 512×512. This would allow us to process the feed of a camera in real-time for further analysis on the content of the video.

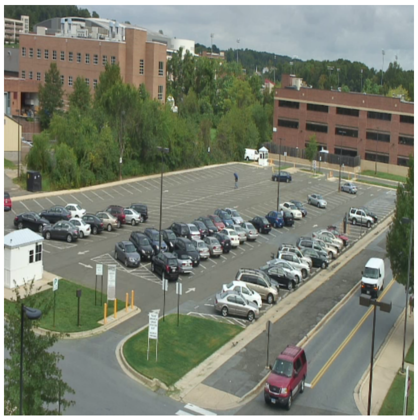

Fig. 4. Original frame

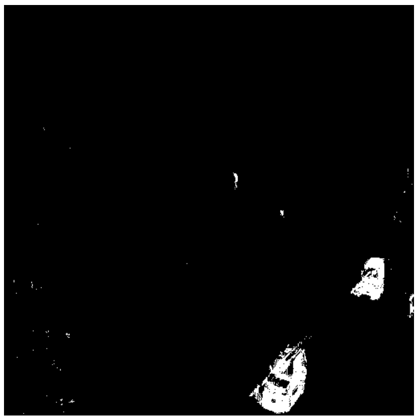

Fig. 5. Background-subtracted frame

V. Conclusions

The nature of the calculations required for background subtraction using the LSBP model lends itself to efficient parallelization through SIMD processors (e.g. GPUs), a factor that is highly usable for use in real applications.

The presented work successfully accomplishes the parallelization of the chosen method of background subtraction using CUDA, significantly improving image processing times by up to 17 times for low resolutions, 24 times for medium resolutions and 28 times for high resolutions. To achieve these improvements, the correct use of the memory bandwidth and the proper mapping of the threads for the processing of each image is a crucial factor.

Although the conditions of hardware were not optimal for the realization of the tests, it is shown that a middle-end equipment is enough for the processing of videos in a close to real-time for images of intermediate resolutions.

The presented progress gives way to future optimizations through more advanced techniques in CUDA, as well as to the beginning development of applications that need the treated bottom subtraction method.

VI. Future Work

The work done has served as motivation to continue with the research of the optimization of the LSBP model with more advanced techniques of CUDA (e.g. analysis of cache misses), with the aim of processing videos of greater resolution closer to real-time. These techniques can be further improved by the use of CPU threads for the handling of batches and the transfer of data between the main memory and the device.

In turn, one of the reasons for the greater optimization of background subtraction method, is to be able to make use of it in future works of video content analysis, such as video summarization and detection of violent actions in real-time for security systems.

References

[1] L. Guo, D. Xu and Z. Qiang. “Background Subtraction Using Local SVD Binary Pattern”. The IEEE Conf. Computer Vision and Pattern Recognition Workshops (CVPRW), 2016.

[2] T. Ojala, M. Pietikainen, T. Maenpaa. “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns”. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 24, Issue 7, pp. 971 – 987, 2002.

[3] W. Kim, S. Suh, W. Hwang, and J.-J. Han. “SVD face: Illumination-invariant face representation”. IEEE Signal Processing Letters, Volume 21, Issue 11, pp. 1336 – 1340, 2014.

[4] M. Hofmann, P. Tiefenbacher, G. Rigoll. “Background segmentation with feedback: The pixel-based adaptive segmenter”. 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2012.

[5] O. Barnich, M. V. Droogenbroeck. “ViBe: A universal background subtraction algorithm for video sequences”. IEEE Transactions on Image Processing, Volume 20, Issue 6, pp. 1709 - 1724, 2011.