EAI Endorsed Transactions
on Creative Technologies

Research Article ICST.ORG

Multi-GPU based framework for real-time motion analysis and tracking in multi-user scenarios

Sidi Ahmed Mahmoudi[1,*]

[1]University of Mons, Faculty of Engineering, Computer Science Department. Place du Parc, 20. 7000, Mons, Belgium

Abstract

Video processing algorithms present a necessary tool for various domains related to computer vision such as motion tracking, event detection and localization in multi-user scenarios (crowd videos, mobile camera, scenes with noise, etc.). However, the new video standards, especially those in high definitions require more computation since their treatment is applied on large video frames. As result, the current implementations, even running on modern hardware, cannot provide a real-time processing (25 frames per second, fps). Several solutions have been proposed to overcome this constraint, by exploiting graphic processing units (GPUs). Although they exploit GPU platforms, they are not able to provide a real-time processing of high definition video sequences. In this work, we propose a new framework that enables an efficient exploitation of single and multiple GPUs, in order to achieve real-time processing of Full HD or even 4K video standards. Moreover, the framework includes several GPU based primitive functions related to motion analysis and tracking methods, such as silhouette extraction, contours extraction, corners detection and tracking using optical flow estimation. Based on this framework, we developed several real-time and GPU based video processing applications such as motion detection using moving camera, event detection and event localization

Keywords: Multi-GPU computing, camera motion estimation, event detection and event localization

Received on 08 May 2014, accepted on 29 September 2014, published on 27 February 2015

doi: 10.4108/ct.2.2e4

1. Introduction

Recently, the CPU power has been capped, essentially for thermal reasons, to less than 4 GHz. A limitation that has been circumvented by the change of internal architecture, with multiplying the number of integrated computing units. This evolution is reflected in both general (CPU) and graphic (GPU) processors which present a large number of computing units, their power has far exceeded the CPUs ones.

Video processing and more particularly motion estimation algorithms present a very active research topic in computer vision domain. They can be used, for example, in surveillance systems tracking humans in public places, such as metro or airports, to identify possible abnormal behaviors and threats [**?**]. Motion estimation algorithms serve therefore as a common building block of some more complex routines and systems. However, these algorithms are hampered by their high consumption of both computing power and memory. The exploitation of graphic processors can present an efficient solution for their acceleration.

Unlike algorithms requiring a high dependency of computation between the input data and hence a complicated parallelization, most of image and video processing algorithms consist of similar computations over many pixels. This fact makes them well adapted for acceleration on GPU by exploiting its processing units in parallel. Nevertheless, the new standards, especially those in high resolutions require more computation since their treatment is applied on large video frames. Thus, the current implementations, even running on modern hardware, cannot provide a real-time processing. Moreover, the treatment of TV broadcast images, which cannot be down sampled, require an accelerated object detection and recognition. Therefore, a fast processing of videos is needed to ensure the treatment of 25 high definition frames per second (25 fps). To overcome these constraints, several GPU computing approaches have recently been proposed. Although they exploit GPU platforms, they are not able to provide a real-time processing of high definition video sequences. Moreover, they are not well adapted for exploiting multiple GPUs. Indeed, clusters and computers that contain multiple GPUs are becoming commonplace nowadays. However, their exploitation requires an efficient sharing of data and

*Corresponding author. Email: sidi.mahmoudi@umons.ac.be

computations between the available GPUs. Otherwise, the computation time (with many GPUs) can be even more elevated, compared to the case using one single GPU due to data transfers and memory allocation times.

In this paper, we propose a new framework enabling an effective exploitation of single and multiple GPUs for accelerating video processing algorithms, and hence achieving real-time treatment of high definition videos. This framework allows an efficient management of single and multiple GPUs memories and a fast visualization of results. The proposed framework includes several CUDA[1] based primitive functions related to motion analysis and tracking methods, such as silhouette extraction, contours extraction, corners detection and tracking using optical flow estimation. With this framework, we developed several real-time video processing applications such as motion detection using mobile camera, event detection and event localization.

The remainder of the paper is organized as follows: related works are discussed in the second section. Section 3 presents the proposed framework for real-time video processing using multiple GPUs. In sections 4, 5, 6, we present three Multi-GPU based use cases of our framework : motion detection using mobile camera, event detection and event localization, respectively. Section 8 compares and evaluates the related CPU, GPU and multi-GPU implementations. Finally, conclusions and future works are described in the last Section.

2. Related work

In general, event detection and localization methods consist of analyzing and modeling normal behaviors, and then detecting the difference between the normal behavior model and the observed behaviors. These variations can be labeled as abnormal or emergency events. In this category, Bilinski et al. [**?**] extract hog descriptors in order to calculate predefined models (i.e. crowd scenarios) that allow to recognize crowd events. Authors in [**?**] propose a context-aware method that allows to detect anomalies by tracking all moving objects in the video. There are also some work in [**?**] which addresses the problem of analyzing video events in crowded scenes. A novel manifold learning method was developed to achieve an effective modeling of video events in a low dimensional space. The general idea of these methods consist on estimating the displacement and velocity of features in a given video frame with respect to the previous one. In this work, we are more focused on optical flow methods since they present a promising solution for tracking even in noisy and crowded scenes or in case of small motions. Moreover, most of camera motion estimation techniques are based on optical flow methods such as shown in [**?**]. The latter presents a method that creates prototypes of optical flows before performing a linear decomposition of motion vectors, which enable to estimate the camera parameters.

Otherwise, several GPU based motion tracking methods have been proposed such as in [**?**], which developed a GPU version of motion detection technique from a moving platform. As result, they can build the background model and detect motion regions at around 18 fps on 320 × 240 video captured from a moving camera. Authors in [**?**] proposed an eye blink detector which can be used in dry eye prevention system. Their solution, exploiting GPU, is based on histogram back projection and optical flow techniques.

In case of optical flow-based motion tracking algorithms, one can distinguish two categories of related works. The first presents so called dense optical flow which tracks all frame pixels without selecting any features. In this context, Marzat et al. [**?**] proposed a GPU implementation of the Lucas-Kanade method for the optical flow estimation. The software was programmed using the CUDA library (Compute Unified Device Architecture) to compute dense and accurate velocity field at about 15 fps with 640x480 video resolution. Authors in [**?**] presented the CUDA implementation of the Horn-Shunck optical flow, that offered a real-time processing of 316×252 video resolution. Gwosdek et al. [**?**] developed a GPU implementation of the Euler-Lagrange (EL) framework for solving variational optical flow methods using sequences with 640x480 pixels in near-real-time.

The second category includes software tools tracking selected image features only. Sinha et al. [**?**] developed a GPU implementation of the popular KLT feature tracker [**?**] and the SIFT feature extraction algorithm [**?**]. This was developed with the OpenGL/Cg libraries allowing to extract about 800 features from 640x480 video at 10 fps which is approximately 10 times faster than the corresponding CPU implementation. There is also a work in [**?**] proposing a GPU based block matching technique using OpenGL. This implementation offered a real-time processing of 640x480 video. Sundaram et al. [**?**] developed a method for computing point trajectories based on a fast GPU implementation of the optical flow algorithm that tolerates fast motion. This parallel implementation runs at about 22 fps, which is 78 times faster than its CPU version. However, despite their great speedups, none of the abovementioned GPU based software tools can provide real-time processing of high definition videos. Moreover, they are not well adapted for exploiting multiple GPUs simultaneously.

Our contribution consists on proposing a new framework for video processing on single or multiple

[1] CUDA. https://developer.nvidia.com/cuda-zone

GPUs. This framework contributes within three main factors:

1. Efficient and adapted exploitation of GPU in case of video processing applications. Indeed, we apply several optimization techniques such as: the adapted selection of the number of GPU threads within blocks, the exploitation of GPU shared and texture memories, the overlapping of data transfers by kernels executions that allows reducing data transfer times, the OpenGL fast visualization of results.

2. Scalable exploitation of multiple GPUs that allows to benefit from the full power of each graphic cards. The number of selected GPUs depends directly from the computational intensity of the input application. As result, the framework allows to reduce significantly the computation time and the energy consumption.

3. Several primitive GPU based image and video processing functions are developed such as: silhouette extraction, contours and corners detection, features tracking, etc. These functions can be selected easily within our framework for accelerating video processing applications. As example, we developed three real-time HD/Full HD or even 4K video processing applications:

    - Camera motion estimation using optical flow vectors.
    - Event detection in crowd videos
    - Event localization in multi-user scenarios.

3. Multi-GPU based framework for real-time video processing

This section is presented in two parts: the first one describes our framework for video processing on single or multiple GPUs. The second part presents the related GPU based primitive functions : silhouette extraction, features detection and tracking, based on optical flow estimation. These primitive functions can be directly selected within our framework.

3.1. The proposed framework

Our framework is based upon CUDA for parallel constructs and OpenGL for visualization, using three steps: multiple GPUs selection, CUDA parallel processing and OpenGL visualization (Fig. **??**).

1. **Multiple GPUs selection :** Video processing algorithms are known by their high intensity computation, and their well adaptation for parallel calculation. Therefore, apart from implementing our video processing algorithms on a single GPU, we propose a version taking advantage of multiple GPU systems that nowadays are becoming commonplace. The default setup of our framework selects a number of one GPU only. During the first 20 frames , we test the performance of our application. If the condition of real-time processing is achieved, the number of 1 GPU is maintained, else we increase the number of GPUs until achieving a treatment of 25 fps. This allows to select one GPU only in case of processing low resolution videos or applying low intensive treatments. As result, accelerated treatments are obtained with a reduced energy consumption. Once the number of GPUs selected, the program initializes all of them. Then, the input image frame is first uploaded to each GPU. This frame is virtually divided into equally sized subframes along y dimension and once the image data is available, each GPU is responsible for treating its part of the frame (subframe).

2. **CUDA parallel processing :** Before launching the parallel treatments of the current subframes. The number of CUDA threads, within each GPU, in the so called blocks and grid has to be defined, so that each thread can perform its processing on one or a group of pixels in parallel. This enables the program to process the image pixels in parallel. Note that the number of threads depends on the number of pixels. Once the number and the layout of threads is defined, different CUDA functions (kernels) are executed sequentially, but each of them in parallel using multiple CUDA threads. In our case, we selected a number of CUDA threads equal to the number of pixels, (within each subframe) which enables for each CUDA thread to treat its corresponding pixel.

    For a better exploitation of GPU, we propose to load the input image on GPU texture memory for a fast access to pixels. We have also loaded each pixel neighbors on GPU shared memory for a fast processing of pixels using their neighbors values. This optimization is so useful in case of applying image convolutions, the related filters are loaded in the shared memory. Moreover, we employ the streaming technique within multiple GPUs so that each GPU can overlap effectively data transfers by kernels executions.

3. **OpenGL visualization :** At the end of computations for each frame (the subframes). The results can be displayed on screen using the OpenGL graphics library that allows for fast visualization, as it can operate on the already existing buffers on GPU, and thus requires less data transfer between host and device memories. In case of Multi-GPU

treatments, each GPU result (subframe) need to be copied to the GPU which is charged of displaying. This, however, is a fast operation since contiguous memory space is always transferred. Once the visualization of the current image is completed, the program goes back to the first step to load and process the next frames of the video.

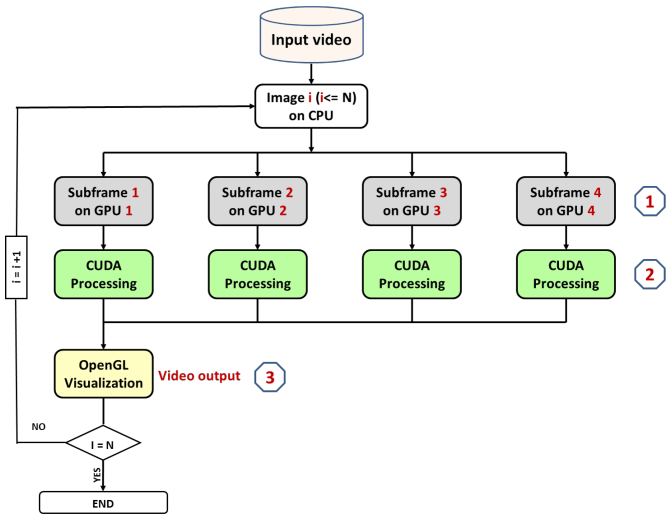


Figure 1. Multi-GPU based framework for real-time video processing (N : video frames number)

This framework could provide a unified way to implement advanced methods accordingly to each given scenario. These methods can exploit easily the primitive functions (described in Section 3.2). Otherwise, the framework user could provide the new required algorithms before integrating them into the processing primitives. Fig. **??** summarizes the method of exploitation of our framework. Notice that the selection of primitive functions depends mainly from the video processing application.

3.2. GPU-based primitive functions

In this section, we propose GPU (CUDA) based primitive functions that can be used within our framework as shown in Fig. **??**. These primitives correspond to the functions of silhouette extraction, features (corners and contours) detection and tracking methods.

1. **GPU based silhouette extraction :** the computation of difference between frames presents a simple and efficient method for detecting the silhouettes of moving objects. We propose a GPU implementation of this method using three steps. First, we load the two first frames on GPU in order to compute the difference between them during the CUDA parallel processing step. Once the first image displayed, we replace it by the next

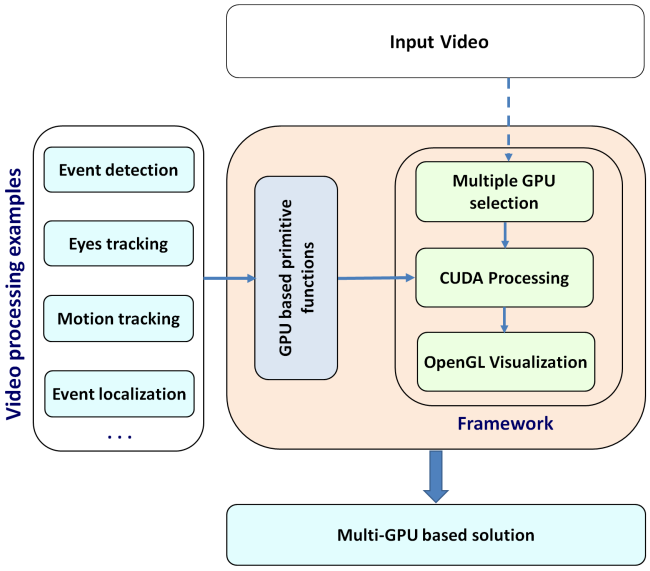


Figure 2. Exploitation process of our framework

video frame in order to apply the same treatment. Fig **??**.(b) presents the obtained result of silhouette extraction. This figure shows two silhouettes extracted, that present two moving persons. In order to improve the quality of results, a threshold of 200 was used for noise elimination.

2. **GPU based features detection and tracking :** in this section, we propose the GPU implementation of both features detection and tracking methods. The first one enables to detect the good features to track, i.e. corners. To achieve this, we have exploited the Bouguet's corners extraction technique [**?**], which is based on the principle of Harris detector [**?**]. Moreover, we developed a CUDA version of Deriche-Canny edge detector. These two GPU implementations are detailed in [**?**].

   The second step enables to track the features (corners) previously detected using the optical flow method, which presents a distribution of apparent velocities of movement of brightness pattern in an image. It enables to compute the spatial displacements of images pixels based on the assumption of constant light hypothesis which supposes that the properties of consecutive images are similar in a small region. For more detail about optical flow computation, we refer readers to [**?**] and [**?**]. In this work, we propose the GPU implementation of the Lucas-Kanade algorithm, which is well known for its high efficiency, accuracy and robustness. This algorithm disposes of six steps:

(a) **Step 1: Pyramid construction :** in the first step, the algorithm computes a pyramid representation of images I and J which represent two consecutive images from the video. The other pyramid levels are built in a recursive fashion by applying a Gaussian filter. Once the pyramid is constructed, a loop is launched that starts from the smallest image (the highest pyramid level) and ends with the original image (level 0). Its goal is to propagate the displacement vector between the pyramid levels.

(b) **Step 2: Pixels matching over levels :** for each pyramid level (described in the previous step), the new coordinates of pixels (or corners) are calculated.

(c) **Step 3: Local gradient computation :** in this step, the matrix of spatial gradient G is computed for each pixel (or corner) of the image I. This matrix of four elements (2×2) is calculated with the horizontal and vertical spatial derivatives. The computation of the gradient matrix takes into account the area (window) of pixels which are centered on the point to track.

(d) **Step 4: Iterative loop launch and temporal derivative computation :** a loop is launched and iterated until the difference between the two successive optical flow measures (calculated in the next step), or iterations, is higher than a defined threshold. Once the loop is launched, the computation of the temporal derivatives is performed using the image J (second image). This derivative is obtained by the subtraction of each pixel (or corner) of the image I (first image) and its corresponding corner in the image J (second image). This enables to estimate the displacement estimations which is then propagated between successive pyramid levels.

(e) **Step 5: Optical flow computation :** the optical flow measure $\bar{g}$ is calculated using the gradient matrix G and the sum of temporal derivatives presented by shift vector $\bar{b}$. The measure of optical flow is calculated by multiplying the inverse of the gradient matrix G by the shift vector $\bar{b}$.

(f) **Step 6: Result propagation and end of the pyramid loop :** the current results are propagated to the lower level. Once the algorithm reaches the lowest pyramid level (the original image), the pyramid loop (launched in the first step) is stopped. The vector $\bar{g}$ presents the final optical flow value of the analyzed corner.

Upon matching and tracking pixels (corners) between frames, the result is a set of vectors as shown in Equation (**??**):

$$\Omega = \{\omega_1 \dots \omega_n \mid \omega_i = (x_i, y_i, v_i, \alpha_i)\} \quad (1)$$

where:

- x_i, y_i are x a y coordinates of the feature i;
- v_i represents the velocity of the feature i;
- α_i denotes motion direction of the feature i.

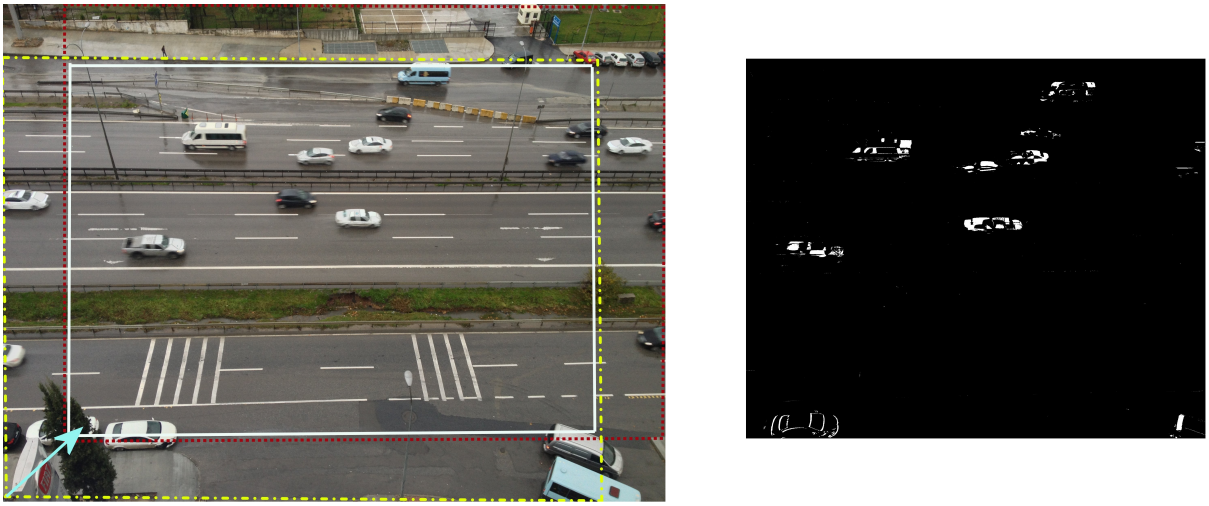
We propose a GPU implementation of the Lucas-Kanade optical flow method by parallelizing its steps on GPU. These steps are executed in parallel using CUDA such that each GPU thread applies its instructions (among the six steps) on one pixel or corner. Therefore, the number of GPU threads is equal to the number of pixels or corners. Since the algorithm looks at the neighboring pixels, for a given pixel, the images, or pyramid levels are kept in the texture memory. This allows a faster access within the 2-dimensional spatial data. Other data, e.g. the arrays with computed displacements, are kept in the global memory, and are cached in the shared memory if needed.

4. Multi-GPU based motion detection using mobile camera :

The above-mentioned GPU implementations are exploited in an application that consists of real-time motion detection within moving camera. In this category, motion detection algorithms are generally based on background subtraction which presents a widely used technique in computer vision domain. Typically, a fixed background is given to the application and new frames are subtracted from this background to detect the motion. The difference will give the objects or motion when the frame is subtracted from the fixed background. This difference in resulting binary image is called foreground objects. However, some scenarios present a dynamic background which can change due to the movement of cameras. In this context, we propose an application for real-time background subtraction, which enables to detect automatically background and foreground using a moving camera. This application can be summarized in four steps:

1. **Corners detection :** the Harris corner detector [**?**] is applied to extract good features to track and examine for camera motion.

2. **Optical flow computation :** the Lukas-Kanade optical flow method [**?**] is applied to track the corners, detected previously.



(a) (b)

Figure 3. (a). Camera motion estimation (b). Motion detection

3. **Camera motion inhibition :** the camera motion is estimated by computing the dominant values of optical flow vectors. This enables to extract the common area between each two consecutive images and focus only on motions related to objects in the scene.

4. **Motion detection :** this step consists of detecting movements based on computing the difference between each two consecutive frames.

In order to achieve a real-time treatment of high definition videos, the method steps are ported on GPU. Their GPU implementation is described in section 3. Fig. **??**.(a) shows a scene of camera motion. Dotted and dashed line presents the first image, dotted line presents the second frame and solid line shows the joint area of two frames. Once, the camera motion is estimated. The joint area between 2 consecutive frames is determined by cropping the incoming and outgoing areas as seen in the white area of Fig. **??**.(a). Fig. **??**.(b) shows the resulting image of background subtraction. White areas represent the difference around moving objects.

This method steps are integrated within our framework for providing an accelerated Multi-GPU based solution that offers an efficient and scalable exploitation of multiple GPUs.

5. Multi-GPU based event detection

Before presenting the GPU implementation, we start by describing our CPU based implementation of event detection method that consists of four steps: points of interest detection, points of interest tracking, clustering and event detection.

1. **Points of interest detection :** the Harris [**?**] corner detector is applied to extract good features, which will be tracked during the next step.

2. **Points on interest tracking :** the Lucas-Kanade [**?**] optical flow method is applied to track the previously detected corners. As a result, this method provides, for each corner, a related velocity and an angular direction of motion. The features (points on interest) that have a velocity close to zero are considered as static features. Noise features are isolated points that have a velocity or direction so different compared to their neighbors.

3. **Clustering :** this step consists of applying the K-means method to get clusters. The features which are used within the clustering process are the spatial coordinates (x; y), the velocity and the direction. The K-means method is a well-known geometric clustering algorithm. This classification allows to have in each class individuals moving with similar velocities and directions in the same region.

4. **Event detection :** this step consists of computing the distance between each cluster and its corresponding one in the next frame. If we have N clusters within each frame, the global distance between frames is represented by the sum of the N distances between clusters.

As compared to normal motion, the positions and sizes of the clusters (rectangles) of abnormal motion are significantly different between two consecutive frames. This is due to the sudden changes of the values of velocity and direction of optical flow fields. Fig. **??** presents the result of the steps listed above

using a crowd video. For more detail about this implementation, we refer readers to [**?**].

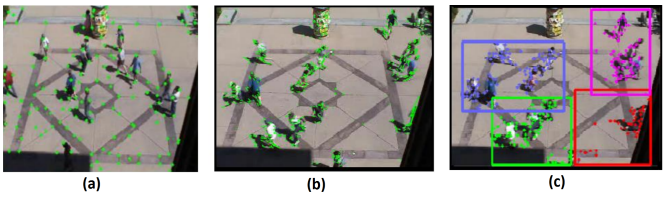


Figure 4. (a). Corners detection (b). Corners tracking (c). Clustering

For a better exploitation of graphic cards, we developed a GPU implementation of the above-mentioned event detection method by parallelizing its most intensive steps on graphic processing unit: corners detection and tracking. The related GPU versions are described in Section 3.2.2. Otherwise, these steps are integrated within our framework for providing an accelerated Multi-GPU based solution that offers an efficient and scalable exploitation of multiple GPUs.

To conduct the experiment, we used as data set videos from different outdoor places, comprising both normal and abnormal motions. The experimental results corresponding to the approach of event detection based on clustering optical flow fields, applied on one of the videos have been presented in Fig. **??**. Abnormal motion includes a sudden situation when a group of people start running. From frame 1 to 550 the people's motion is normal. People tend to run since the frame number 551. More precisely, the assigned distance of frame 551 will be higher than any other assigned distances among frames. In Fig **??**, the blue colored curve is the output of the proposed approach. The Gaussian like curve represents the abnormal motion when the group of people is trying to leave the place with very quick motion.

Fig. **??** presents another case of event detection, applied on a video that consists of four persons only. In frame 231, one person starts running and the assigned distance will be higher. Therefore, an abnormal event is detected in this frame. Moreover, the high values of optical flow vectors (compared to previous values) enabled to localize the persons which present an abnormal motion. The latter are presented by red optical flow vectors, as shown in Fig. **??**.

6. Multi-GPU based event localization

Event localization allows to provide, in real-time, areas in video frames where motion behavior is surprising compared to the rest of motion in the same frame. We propose in this section an approach that takes into account the spatial occupation of the moving objects. It is based on the extraction of scene occupation model from video sequences. Once this model is acquired

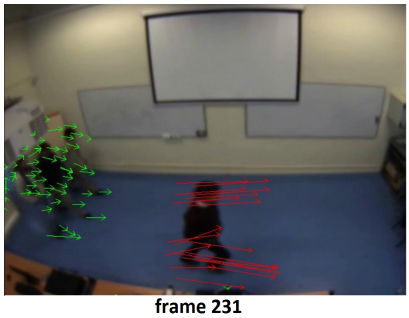


Figure 6. Event detection: video with several moving persons

(Fig. **??**.(c)), motions close to this model are inhibited while only different motion (in other spatial locations) is highlighted. The event localization using scene occupation model follows three principal steps: frame difference, frame accumulation and motion inhibition.

1. **Frame difference :** we use the simplest motion detection technique which consists in the difference between every two consecutive frames in video. This method is very fast to implement. It needs no background modeling which can be a difficult task in some situations. Fig. **??**.(b) shows the result given by this step.

2. **Frame accumulation :** the frame difference enables to detect moving individuals, so every frame represents individuals in movement. The video frames will be accumulated using a threshold accumulator in order to have as result a top-down model representing dominant regions. The threshold accumulator value depends of the computing time of model extraction. This extraction can be longer if one has a bigger value of this threshold. Fig. **??**.(c) shows the model extracted after 1000 frames using a threshold of 100. Indeed, the white area presents the dominant regions on which the major individuals' movements have taken place.

3. **Motion inhibition :** in this step we apply a subtraction between the frames of moving objects (frame difference) and the model extracted. This subtraction allows to inhibit similar motions and focus only on abnormal motions.

On the one hand, we developed a GPU version of the above-mentioned event localization method. This parallel implementation is based on parallelizing these steps on GPU: frame difference, frame accumulation and motion inhibition. The GPU based frame difference is described in Section 3.2.1, while the GPU based frame accumulation consists on exploiting the result of frame difference step for each image of the video. Theses resulting frames will be accumulated (added),

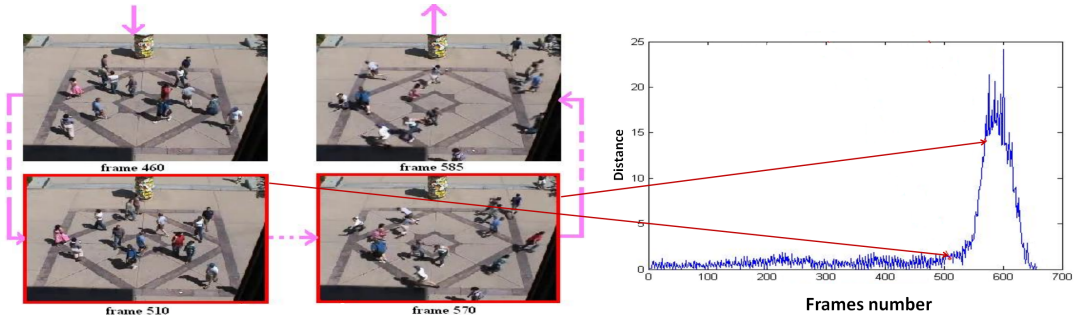


Figure 5. Event detection in crowd videos

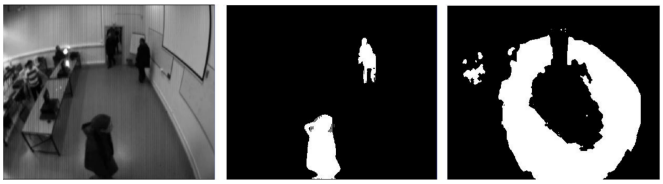


Figure 7. (a). Input Video (b). Frame difference (c). Frame accumulation

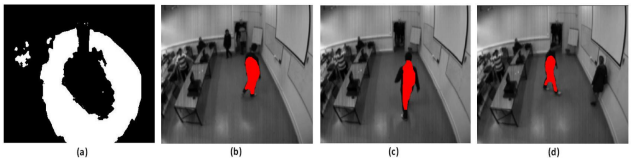


Figure 8. (a). Scene Occupation Model (b).(c).(d). Abnormal events localized

in a GPU buffer, using a threshold accumulator in order to have as result a top-down model representing dominant regions. The GPU implementation of this step consists for each frame of applying an addition and a comparison with threshold. The last step applies a subtraction on GPU between the frames of moving objects (frame difference) and the model extracted in previous step. This enables to inhibit similar motions and focus only on abnormal motions. On the other hand, these GPU based steps are included within our framework for providing an accelerated Multi-GPU based solution that offers an efficient and scalable exploitation of multiple GPUs. Experimentation are conducted with a video of 3800 frames representing normal and abnormal motions. The model extraction (accumulation) needed 1000 frames, the event localization can therefore begin at the same time. At the beginning, when the model is not finished all the moving objects will be salient, but after 1000 frames, some motion is no more interesting as it corresponds to the model. This approach is close to human perception when recurrent motions become annoying after a given time. Fig. **??** shows abnormal events detected due to the motion of a person out of the model representing dominant regions of movement at the frames number 1900, 2100 and 3150, this person is tracked using the red color.

The above-mentioned GPU implementations can be also exploited for accelerating an eye tracker application called "CVC Eye-Tracker" [**?**], presented by an open source eye tracking software. Indeed, this application consists mainly on selecting the 8 facial points (related to eyes, nose and frontal face), followed their tracking. These steps are well adapted for a Multi-GPU acceleration using our framework.

7. Performance

On the one hand, we can say that the quality of the above-mentioned applications of camera motion estimation, event detection and event localization remains identical since the procedure has not changed. Only the architecture and the implementation did. On the other hand, the exploitation of single or multiple GPUs enabled to accelerate the computation time. This acceleration allowed to obtain a real-time processing of high definition videos, even of 4K standards.

Table **??** presents a comparison between CPU, GPU and Multi-GPU performances of our approach of motion detection within mobile camera (Section 4). Notice that the exploitation of GPUs enabled a real-time processing of Full HD videos (1920×1080) and even 4K standards (3840×2160), with an acceleration ranging from 16 × to 47 ×.

Otherwise, Table **??** presents a comparison, in terms of the number of frames per second (fps), between sequential (CPU), parallel (GPU) and Multi-GPU implementations of event detection and localization methods respectively. Notice that use of GPUs enabled to achieve a real-time processing of high definition videos. This is due to the parallel treatment of the pixels within video frames, exploiting the large number of computing units in GPUs. We note also that we obtain reduced acceleration when processing low resolution

Resolution	2 CPU	1 GPU		2 GPU		4 GPU	
		fps	Acc (x)	fps	Acc (x)	fps	Acc (x)
512×512	5 fps	79	16.8 ×	89	17.8 ×	98	19.6 ×
1280×720	2,9 fps	51	17.6 ×	69	23.8 ×	95	32.8 ×
1920×1080	1,9 fps	45	23.7 ×	58	30.5 ×	84	44.2 ×
3840×2160	1,7 fps	35	20.6 ×	54	31.8 ×	80	47.1 ×

Table 1. GPU performances of motion detection using mobile camera

videos since we can't exploit sufficiently the high power of GPUs. As result, the use of multiple GPUs in this case is not so beneficial in terms of performance. Notice also that the number of exploited GPUs should take into account the treatment intensity. Indeed, Table **??** shows that using 2 or 4 GPUs can be significantly advantageous, in terms of performance, when treating Full HD or 4K video. This is due to an efficient exploitation of GPUs since the graphic cards can be fully exploited. In case of event localization, we present single GPU performances only since this method doesn't require a high intensive treatment. Thus, the use of more than one GPU doesn't improve performance.

A demonstration of GPU based features detection, features tracking, and event detection in crowd video is shown in this video sequence: https://www.youtube.com/watch?v=PwJRUTdQWg8.

In order to have a fair evaluation, we compare our GPU performance with the very recent and up to our best knowledge the fastest GPU based image processing library . *i.e.* OpenCV 2.4.9. The latter provides both CPU and GPU versions of algorithms. Table **??** presents a comparison between our GPU performances and those obtained with the GPU module of OpenCV. Notice that our implementations offers better performances thanks to the efficient exploitation of GPU (texture & shared) memories, the overlapping (streaming) of data transfers by CUDA executions and the fast OpenGL visualization.

Notice that the tests were run on the following hardware:

- CPU: Intel Core 2 Quad Q8200, 2.33GHz,
- GPU: 4 x NVIDIA GeForce GTX 580 with 1.5GB of RAM,
- RAM: 8GB,
- OS: 64-bit Linux.

8. Conclusion

We proposed in this paper a new framework for real-time video processing using multiple GPUs, and particularly in case of treating high definition videos such as Full HD formats or even 4K standards. This framework includes several GPU based primitive functions related to motion analysis and tracking methods, such as silhouette extraction, contours extraction, corners detection and tracking using optical flow estimation. Based on this framework, we developed three real-time video processing applications: motion detection using mobile camera, event detection and localization in multi-user scenarios. Performed tests show that our applications can turn in multi-user scenarios, and in real-time even when processing high definition videos such as Full HD or 4K standards. Moreover, the scalability of our results is achieved thanks to the efficient exploitation of multiple graphic cards. As future work, we plan to extend our framework in order to facilitate the implementation of new advanced monitoring and control systems exploiting parallel and heterogeneous platforms, with reduced energy consumption. We plan also to exploit the SDI capture cards[2] that allow for direct video stream capture into the GPU memory without any use of the CPU memory, which enables the software to decrease the amount of PCI-E bandwidth used for the video transmission.

References

[1] A. Fonseca, L. Mayron, D. Socek and O. Marques (2008) "Design and implementation of an optical flow-based autonomous video surveillance system", *Proc. IASTED* 1, p. 209-214.

[2] P. Bilinski, F. Bremond and M. B. Kaaniche (2009) "Multiple object tracking with occlusions using hog descriptors and multi resolution images", *3rd International Conference on Crime Detection and Prevention*, p. 1-6.

[3] F. Jiang, J. Yuan, S. A. Tsaftaris and A. K. Katsaggelos (2011) "Anomalous video event detection using spatiotemporal context", *Computer Vision and Image Understanding*, vol. 115, p. 323-333.

[4] M. Thida, H.-L. Eng, M. Dorothy and P. Remagnino (2011) "Learning video manifold for segmenting crowd events and abnormality detection", *10th Asian Conference on Computer Vision*, vol. 6492, p. 439-449.

[2] NVIDIA Quadro SDI Capture: http://www.nvidia.com/object/product quadro sdi capture us.html

Table 2. Performance of event detection and localization on GPUs: CPU Intel i5 and GPU GTX 580

Videos resolution	Event detection							Event localization		
	2 CPU	1 GPU		2 GPU		4 GPU		2 CPU	1 GPU	
	fps	fps	Acc (x)	fps	Acc (x)	fps	Acc (x)	fps	fps	Acc (x)
320×240	18	82	4.6 ×	74	04.1 ×	69	03.8 ×	20	61	3.1 ×
1280×720	7	44	6.3 ×	54	07.7 ×	66	09.4 ×	11	57	5.2 ×
1980×1080	4	32	8.0 ×	51	12.7 ×	65	16.3 ×	5	55	11 ×
3840×2160	1.6	21	13.1 ×	38	23.7 ×	63	39.4 ×	2	50	25 ×

Table 3. Performance of eye-tracker on GPU: CPU Intel i5 and GPU GTX 580

Videos resolution	Silhouette extraction		Edges & corners detection		Corners tracking	
	OpenCV GPU	GPU (ours)	OpenCV GPU	GPU (ours)	OpenCV GPU	GPU (ours)
1280×720	67 fps	90 fps	62 fps	79 fps	25.2 fps	30 fps
1920×1080	49 fps	62.7 fps	33 fps	39.9 fps	28.8 fps	38.8 fps
3840×2160	33.5 fps	40.1 fps	28.7 fps	34 fps	13.4 fps	20.1 fps

[5] J. Almeida, R. Minetto, T. A. Almeida, R. Torres and N. Leite (2009) "Robust Estimation of Camera Motion Using Optical Flow Models", *5th International Symposium, ISVC*, vol. 5875, p. 435-446.

[6] J. Marzat, Y. Dumortier and A. Ducrot (2009) "Real-time dense and accurate parallel optical flow using CUDA", *Proc. WSCG 1*, p. 105-111.

[7] Y. Mizukami and K. Tadamura (2007) "Optical flow computation on Compute Unified Device Architecture", *Proc. 14th International Conf. on Image Analysis and Processing*, 1, p. 179-184.

[8] Gwosdek. Pascal, Zimmer. Henning, Grewenig. Sven, Bruhn. AndrÃŽs and Weickert. Joachim (2012) "A Highly Efficient GPU Implementation for Variational Optic Flow Based on the Euler-Lagrange Framework", *Trends and Topics in Computer Vision*, vol. 6554, p. 372-383.

[9] S. Sinha, J.-M. Fram, M. Pollefeys and Y. Genc(2006) "GPU-based video feature tracking and matching", *EDGE, Workshop on Edge Computing Using New Commodity Architectures 1*.

[10] C. Tomasi and T. Kanade (1991) "Detection and tracking of point features", *Technical Report CMU-CS-91-132, Carnegie Mellon University, Pittsburgh*, p. 383-394.

[11] D. Lowe (2004) "Distinctive image features from scale-invariant key-points", *Int. J. Computer Vision (IJCV)*, 60(2), p. 91-110.

[12] J.M. Ready and C.N. Taylor (2007) "GPU acceleration of real-time feature based algorithms", *Proc. IEEE Workshop on Motion and Video Computing, WMVC'07*, vol. 1, p. 8-9.

[13] N. Sundaram, T. Brox and K. Keutzer (2010) "Dense point trajec-tories by GPU-accelerated large displacement optical flow", *In the 11th European Conference in Computer Vision, ECCV*.

[14] J. Y. Bouguet (2000) "Pyramidal Implementation of the Lucas Kanade Feature Tracker", *Intel Corporation Microprocessor Research Labs*.

[15] C. Harris and M. Stephens (1988) "A combined corner and edge detector", *in Proceedings of the 4th Alvey vision conference*, vol. 15, p. 147-151.

[16] S. A. Mahmoudi and P. Manneback (2012) "Efficient Exploitation of Heterogeneous Platforms for Images Features Extraction", *International Conference on Image Process-ing Theory, Tools and Applications, IPTA*, p. 91-96.

[17] B. D. Lucas and T. Kanade (1981) "An iterative image registration technique with an application to stereo vision", *Imaging Understanding Workshop*, p. 121.

[18] S.A. Mahmoudi, M. Kierzynka, P. Manneback and K. Kurowski(2014) "Real-time motion tracking using optical flow on multiple GPUs", *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 62, Issue. 1, p. 139-150.

[19] S. A. Mahmoudi, H. Sharif, N. Ihaddadene and C. Djeraba(2008) "Abnormal event detection in real-time video", *First International Workshop on Multimodal Interactions Analysis of Users in a Controlled Environment, ICMI*.

[20] Q. Y, G. Medioni (2008) " A GPU-based implementation of motion detection from a moving platform ", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops CVPRW'08*, p. 1-6.

[21] P. Polatsek (2008), "Eye Blink Detection", *Research report. IIT.SRC, Bratislava, Slovakia*, p. 1-8.

[22] O. Ferhat, F. Vilariño and F. J. Sanchez, (2014) " cheap portable eye-tracker solution for common setups ", *Journal of Eye Movement Research*, vol. 7, p. 1-10.