Multi-CPU/Multi-GPU Based Framework for Multimedia Processing

Sidi Ahmed Mahmoudi(✉) and Pierre Manneback

University of Mons, Faculty of Engineering, Computer science department
20, Place du Parc. Mons, Belgium
{Sidi.Mahmoudi,Pierre.Manneback}@umons.ac.be

Abstract. Image and video processing algorithms present a necessary tool for various domains related to computer vision such as medical applications, pattern recognition and real time video processing methods. The performance of these algorithms have been severely hampered by their high intensive computation since the new video standards, especially those in high definitions require more resources and memory to achieve their computations. In this paper, we propose a new framework for multimedia (single image, multiple images, multiple videos, video in real time) processing that exploits the full computing power of heterogeneous machines. This framework enables to select firstly the computing units (CPU or/and GPU) for processing, and secondly the methods to be applied depending on the type of media to process and the algorithm complexity. The framework exploits efficient scheduling strategies, and allows to reduce significantly data transfer times thanks to an efficient management of GPU memories and to the overlapping of data copies by kernels executions. Otherwise, the framework includes several GPU-based image and video primitive functions, such as silhouette extraction, corners detection, contours extraction, sparse and dense optical flow estimation. These primitives are exploited in different applications such as vertebra segmentation in X-ray and MR images, videos indexation, event detection and localization in multi-user scenarios. Experimental results have been obtained by applying the framework on different computer vision methods showing a global speedup ranging from 5 to 100, by comparison with sequential CPU implementations.

Keywords: GPU · Heterogeneous architectures · Image and video processing · Medical imaging · Motion tracking

1 Introduction

During the last years, the architecture of central processing units (CPUs) has so evolved that the number of integrated computing units has been multiplied. 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. In this context, image and video processing algorithms are well adapted for acceleration on the GPU by exploiting its processing units in parallel,

A. Amine et al. (Eds.): CIIA 2015, IFIP AICT 456, pp. 54–65, 2015.
DOI: 10.1007/978-3-319-19578-0_5

since they consist mainly of a common computation over many pixels. Several GPU computing approaches have recently been proposed. Although they present a great potential of GPU platform, hardly any is able to process high definition image and video efficiently and accordingly to the type of Medias (single image, multiple image, multiple videos and video in real time). Thus, there was a need to develop a framework capable of addressing the outlined problem.

In literature, one can categorize two types of related works based on the exploitation of parallel and heterogeneous platforms for multimedia processing: one related to image processing on GPU such as presented in [19], [12] which proposed CUDA[1] implementations of classic image processing and medical imaging algorithms. A performance evaluation of GPU-based image processing algorithms is presented in [15]. These implementations offered high improvement of performance thanks to the exploitation of the GPU's computing units in parallel. However, these accelerations are so reduced when processing image databases with different resolutions. Indeed, an efficient exploitation of parallel and heterogeneous (Multi-CPU/Multi-GPU) platforms is required with an effective management of both CPU and GPU memories. Moreover, the treatment of low-resolution images cannot exploit effectively the high power of GPUs since few computations will be launched. This implies an analysis of the spatial and temporal complexities of algorithms before their parallelization.

On the other hand, video processing algorithms require generally a real-time treatment. We may find several methods in this category, such as understanding human behavior, event detection, camera motion estimation, etc. These methods are generally based on motion tracking algorithms that can exploit several techniques such as optical flow estimation [6], block matching technique [20], and scale-invariant feature transform (SIFT) [9] descriptors. In this case also, several GPU implementations have been proposed for sparse [11] and dense [14] optical flow, Kanade-Lucas-Tomasi (KLT) feature tracker and SIFT feature extraction algorithm [17]. Despite their high speedups, none of the above-mentioned implementations can provide real-time processing of high definition videos. Our contribution consists on proposing a new framework that allows an effective and adapted processing of different type of Medias (single image, multiple images, multiple videos, video in real time) exploiting parallel and heterogeneous platforms. This framework offers:

1. Smart selection of resources (CPU or/and GPU) based on the estimated complexity and the type of media to process. In fact, additional computing units are exploited only in case of intensive and parallelizable tasks.
2. Several GPU-based image and video primitive functions ;
3. Efficient scheduling of tasks and management of GPU memories in case of Multi-CPU/Multi-GPU computations ;
4. Acceleration of several real-time image and video processing applications.

The remainder of the paper is organized as follows: section 2 presents our GPU-based image and video processing functions. The third section is devoted

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

to describe the proposed framework for multimedia processing on parallel and heterogeneous platforms. Experimental results are given in section 4. Finally, conclusions and future works are discussed in the last section.

2 GPU-Based Primitive Functions

This section presents our image and video primitive functions that could be exploited by our framework for accelerating several computer vision methods.

2.1 Image Processing Primitive Functions

2.1.1 Noise Elimination we proposed the GPU implementation of noise elimination methods using the smoothing (or blurring) approach. The latter consists on applying a 2-D convolution operator to blur images and remove noise. We developed GPU version of linear, median and Gaussian filtering which represent the most used techniques for noise elimination. This GPU implementation consists of selecting the same number of CUDA threads as the number of image pixels. This allows for each CUDA thread to apply the multiplication of one pixel value with filter values. All the CUDA threads are launched in parallel. More details about this implementation are presented in [12].

2.1.2 Edges detection we proposed a GPU implementation of the recursive contours detection method using Deriche technique [3]. The noise truncature immunity and the reduced number of required operations make this method very efficient. Our GPU implementation of this method is described in [12], based on the parallelization of its four steps on GPU. Fig. 3(c) illustrates an example of edges detection.

2.1.3 Corners detection we developed the GPU implementation of Bouguets corners extraction method [2], based on Harris detector [5]. This method is efficient thanks to its invariance to rotation, scale, brightness, noise, etc. Our GPU implementation of this method is described in [16], based on parallelizing its four steps on GPU. Fig. 3(b) illustrates an example of corners detection.

Moreover, we have integrated the GPU module of the OpenCV [2] library that disposes of many GPU-based image processing algorithms such as FFT, Template Matching, histogram computation and equalization, etc.

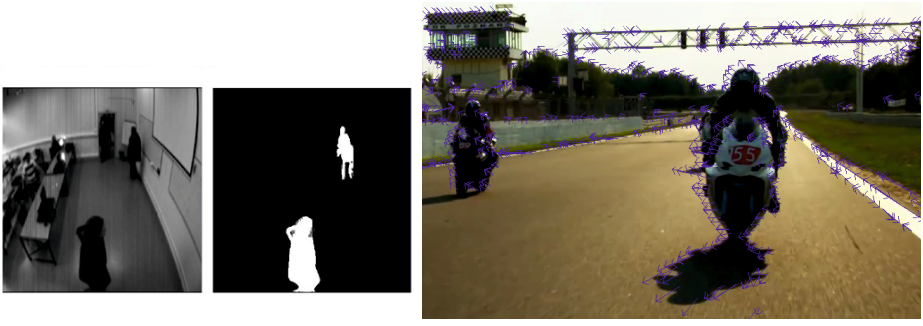
2.2 Video Processing Primitive Functions

2.2.1 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.

[2] OpenCV GPU Module. www.opencv.org

First, we load the two first frames on GPU in order to compute the difference between them within CUDA in parallel. Once the first image displayed, we replace it by the next video frame in order to apply the same treatment. Fig. 1(a) 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.

2.2.2 Sparse optical flow estimation the sparse optical flow method consists of both features detection and tracking algorithms. The first one enables to detect features that are good to track, i.e. corners. To achieve this, we have exploited our corners extraction method (section 2.1.3). The second step enables to track the features 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. Our GPU implementation is detailed in [11]. Fig. 1(b) presents an example of sparse optical flow estimation using a Full HD video frame with characteristic points detected with the Harris corner detector and then tracked with the Lucas-Kanade method. Displacements are marked with arrows. Note that the arrows located on the static objects like trees or a building are there as a result of moving camera.



(a) GPU based silhouette extraction (b) GPU based sparse optical flow estimation

Fig. 1. GPU based video processing primitive functions

2.2.3 Dense optical flow estimation the GPU implementation of dense optical flow is based on the same process of sparse optical flow estimation. The only difference (compared to sparse) is that the tracking step is applied on all frames pixels. Thus, the number of selected CUDA threads is equal to the number of images pixels which requires more computation.

Notice that the image processing primitive functions have been adapted for treating videos also. Moreover, we have integrated the GPU based video processing algorithms of the OpenCV library such as frames interpolation, MOG (Mixture Of Gaussian) model, morphological operations, etc.

3 The Proposed Framework

The presented results and tests within sections 3 and 4 were run with Linux 64 bits on the following hardware:

- CPU: Intel Core (TM) i7, 980 3.33GHz, RAM : 8GB;
- GPU: 4 x NVIDIA GeForce GTX 580, RAM : 1.5GB.

The GPU-based primitive functions are exploited within our framework for processing different types of Medias: single image, multiple images, multiple videos and video in real time. The framework allows to select in an efficient way the adapted resources (CPU or/and GPU) in order to reduce the computation times with an optimal exploitation of computing units.

3.1 Single Image Processing on GPU

This kind of methods is applied on single images, which are displayed on screen at the end of processing. These algorithms are well adapted for GPU parallelization since they consist on common computations over many pixels. However, the use of graphics processing units offers high acceleration when processing high resolution images only. Indeed, performance can be either reduced with GPUs when treating low resolution images since we cannot benefit enough from the GPU. Therefore, we propose a treatment based on the estimated complexity of algorithms. The proposed treatment for single images is summarized in three steps: complexity estimation, resources selection, adapted processing.

3.1.1 Complexity estimation we propose to estimate the algorithm complexity $f_c$using the equation 1.

$$f_c = f \quad \times \quad comp\_pix \quad \times \quad size \tag{1}$$

where :

1. **f (Parallel fraction) :** Amdahl's law [4] proposed an estimation of the theoretical speedup using N processors. This law supposes that f is the part of program that can be parallelized and (1-f) is the part that can't be made in parallel (data transfers, dependent tasks, etc.). Indeed, high values of f can provide better performance and vice versa.

2. **comp_pix (computation per-image):** graphic processors enable to accelerate image processing algorithms thanks to the exploitation of the GPU's computing units in parallel. These accelerations become more significant when we apply intensive treatments since the GPU is specialized for highly parallel computation. The number of operations per pixel presents a relevant factor to estimate the computation intensity.
3. **size :** represents the resolution of input image.

3.1.2 Resources selection based on the estimated complexity f_c, we can have a good guidance for selecting the adapted resource (CPU or GPU) for computation. In fact, we launched for execution several GPU classic image processing (edge detection, corners detection. . .) algorithms using different image resolutions. These experiments allowed to define the value of f_c from which the GPU starts offering better performance than the CPU. This value is called the threshold S. Once the threshold defined, we compare the estimated complexity f_c for each input algorithm with the threshold S.

If $f_c > S$, the treatment is applied on GPU, else the CPU is used for processing. Notice that within our above-mentioned materiel, we have obtained a threshold S of 800000, that correspond to an algorithm with these parameters:

1. parallel fraction: 0.5 ;
2. number of operations per pixel comp_pixel: 10 ;
3. image resolution: 400×400.

We note also that the threshold value can change with other material configurations, since the number of GPUs computing units and the size of memories is not the same. Therefore, we propose to compute the threshold at each changement of material.

3.1.3 Adapted processing after selecting the adapted resource, CPU treatments are launched in case of low intensive algorithms ($fc < S$). The OpenCV library is employed for this aim. Otherwise, in case of high intensive algorithms ($fc > S$), we apply GPU treatment with three steps:

1. **Loading of input images on GPU :** first, the input images are loaded on GPU memory.
2. **CUDA parallel processing :** before launching the parallel processing of the current frame, the number of GPU threads 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.

3. **OpenGL Visualization :** the output image is directly visualized on screen through the video output of GPU. Therefore, we propose to exploit the graphic library OpenGL enabling fast visualization, since it works with buffers already existing on GPU.

3.2 Multi-CPU/Multi-GPU Based Processing of Multiple Images

In case of multiple images treatment, performance can be less improved for two reasons: the first one is the inability to visualize many output images using only one video output that requires a transfer of results from GPU to CPU memory. The second constraint is the high computation intensity due to treatment of large sets of images. In order to overcome these constraints, we propose an implementation exploiting both CPUs and GPUs that offers a faster solution for multiple images processing. This implementation is based on the executive support StarPU [1] which offers a runtime for heterogeneous multicore platforms. For more detail, we refer authors to [8]. The employed scheduling strategy has been improved by taking into account the complexity factor f_c described in section 3.1.1. Indeed, high intensive tasks have higher priority for GPU computation. The low intensive tasks will be affected with a low priority for GPU. This allows to maximize the exploitation of available resources. As result, the repartition of tasks depends mainly on their computational intensity.

3.3 Multi-CPU/Multi-GPU Based Processing of Multiple Videos

This kind in methods is applied on a group of video sequences in order to extract some significant features. The latter can be exploited in several applications such as similarity computation between videos, videos indexation and classification. The real time processing is not required in this case. The treatment of a set of videos can be presented by the treatment of a set of images since a video is always represented by a succession of frames. Therefore, we propose a Multi-CPU/Multi-GPU treatment for multiple videos as shown in section 3.2.

3.4 Real Time Videos Processing on Multiple GPUs

In this case, we propose to exploit GPUs only since the video frames should be processed in order. This excludes the possibility of using heterogeneous platforms, which defines an order based on the employed scheduling strategy. Our approach of video processing on single or multiple GPUs consists of three steps:

1. **GPUs selection :** the program, once launched, first detects the number of GPUs in the system, and 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. **Multi-GPU computation :** in this step, each GPU can apply the required GPU treatment (exp. optical flow computation). The related algorithm can be selected from our GPU primitive functions, or introduced by the framework user. We note also that the number of CUDA threads depends on the number of pixels within each subframe.
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 video frames.

Otherwise, the framework can be used for processing multiple videos simultaneously using multiple GPUs. Indeed, each video stream is loaded and processed with one GPU. At the end of computations for each GPU (actual frame), the result is copied to the GPU which is charged for displaying. Each GPU result is visualized in a separated window in the same screen. Fig. 3.4 summarizes our framework showing the selected resources for each type of media. The figure shows also the primitive functions that could be exploited within the framework for accelerating different computer vision examples that require intensive computations.

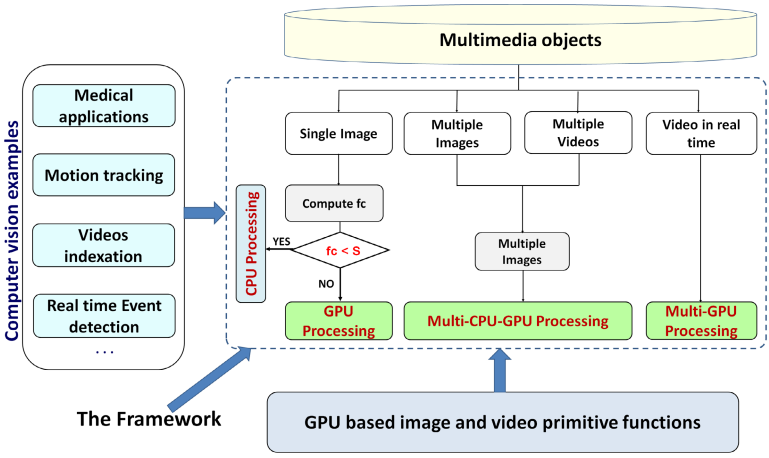


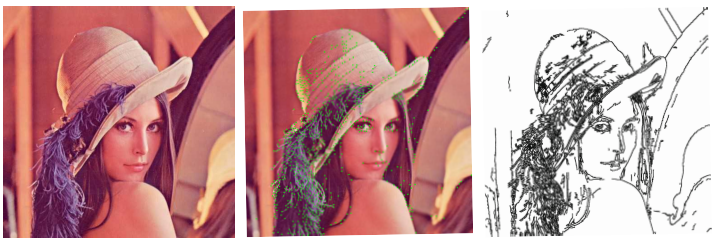
Fig. 2. Multi-CPU/Multi-GPU based Framework for Multimedia Processing

4 Experimental Results

The proposed framework has been exploited in several high intensive applications related to image and video processing such as image pre-processing, vertebra segmentation, videos indexation, event detection and localization.

4.1 CPU/GPU Based Image Pre-processing

Most of image processing methods apply a pre-processing step that allows to prepare the image for treatment. We can cite edges and corners detection methods which are so exploited for this aim. Based on our framework, we propose to accelerate these methods using CPU or GPU since the treatments are applied on single image. As presented in our framework, a complexity estimation is applied to select the convenient resource (CPU or GPU). Table 1 presents the selected resources and performance of corners and edges for different image resolutions. For each one, the complexity is evaluated using the above-mentioned metrics (section 3.1.1). The parallel fraction f presents the percentage of parallelizable computing part relative to total time, while the remaining part (1 - f) is presented by transfer (loading, visualization) operations. The computation per pixel is presented by the average of operations number between the steps of contours and corners detection. As result, the CPU is selected for treating low intensive methods, while the GPU is selected for high intensive ones. This allows to obtain fast results with a reduced energy consumption. In order to validate our results, we have calculated the ratio of acceleration (ACC) with GPU compared to CPU.



(a) Input image (b) Corners (c) Edges

Fig. 3. Edges and corners detection within our framework

Table 1. CPU/GPU based processing of single image processing (edges and corners detection), S = $8.0 * 10^5$

Images	**f**	$comp_{pix}$	f_c	$f_c > S$	**CPU/GPU ?**	**Acc**
256 × 256	0.55	6.1	$2.2 * 10^5$	No	**CPU**	**00.87** ↘
512 × 512	0.81	6.1	$1.3 * 10^6$	Yes	**GPU**	**05.88** ↗
1024 × 1024	0.86	6.1	$5.5 * 10^6$	Yes	**GPU**	**12.01** ↗
3936 × 3936	0.90	6.1	$8.5 * 10^7$	Yes	**GPU**	**19.85** ↗

As shown in Table 1, the GPU is selected only in case of methods that can benefit from the GPU's power. Otherwise, the CPU is selected. Fig. 3 presents an example of edges and corners detection within our framework.

4.2 Multi-CPU/Multi-GPU Based Vertebra Segmentation

The context of this application is the cervical vertebra mobility analysis on X-Ray or MR images. The main objective is to detect vertebra automatically. The computation time presents one of the most important requirements for this application. Based on our framework, we propose a hybrid implementation of the most intensive steps, which have been defined with our complexity factor f_c. Our solution for vertebra detection on Multi-CPU/Multi-GPU platforms is detailed in [8] for X-Ray images, and in [7] for MR images. Fig. 4(a) presents the results of vertebra detection in X-ray images, while Fig. 4(b) is related to present the detected vertebra in MR images. Notice that the use of heterogeneous platforms allowed to improve performance with a speedup of 30 × for vertebra detection within 200 high resolution (1472×1760) X-ray images, and a speedup of 98 × when detecting vertebra in a set of 200 MR images (1024 × 1024).

(a) Vertebra detection in X-ray images (b) Vertebra detection in MR images

Fig. 4. Vertebra detection in X-ray images

4.3 Multi-CPU/Multi-GPU Based Videos Indexation

The aim of this application is to provide a novel browsing environment for multimedia (images, videos) databases. It consists on computing similarities between videos sequences, based on extracting features of images (frames) composing videos [18]. The main disadvantage of this method is the high increase of computing time when enlarging videos sets and resolutions. Based on our framework, we propose a heterogeneous implementation of the most intensive step of features extraction in this application. This step, detected within our complexity estimation equation, is presented by the edge detection algorithm which provides relevant information for detecting motions areas. This implementation is detailed in [13] showing a total gain of 60% (3 min) compared to the total time of the application (about 5 min) treating 800 frames of a video sequence (1080x720).

4.4 Multi-GPU Based Event Detection and Localization in Real Time

This application is used for event detection and localization in real time. It consists of modeling normal behaviors, and then estimating the difference between the normal behavior model and the observed behaviors. These variations can be labeled as emergency events, and the deviations from examples of normal behavior are used to characterize abnormality. Once the event detected, we localize the areas in video frames where motion behavior is surprising compared to the rest of motion in the same frame. Based on our framework, we propose a Multi-GPU implementation of the most intensive steps of the application. The latter are also defined within the above-mentioned complexity factor f_c. This implementation is detailed in [10]. Notice that performed tests show that our application 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. 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..

5 Conclusion

We proposed in this paper a new framework that allows an adapted and effective exploitation of Multi-CPU/Multi-GPU platforms accordingly to the type of multimedia (single image, multiple images, multiple videos, video in real time) objects. The framework enables to select firstly the computing units (CPU or/and GPU) for processing, and secondly the methods to be applied depending on the type of media to process and the algorithm complexity. Experimental results showed different use case applications that have been improved thanks to our framework. Each application has been integrated in an adapted way for exploiting resources in order to reduce both computing time and energy consumption. As future work, we plan to improve our complexity estimation by taking into account more parameters such as tasks dependency, GPU generation, etc. we plan also to include primitive functions related to 3D image processing within our framework. The latter will be exploited for several medical imaging applications that could be applied larger sets of images and videos.

References

1. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 863–874. Springer, Heidelberg (2009)
2. Bouguet, J.Y.: Pyramidal Implementation of the Lucas Kanade Feature Tracker, Description of the algorithm. Intel Corporation Microprocessor Research Labs, 851–862 (2000)

3. Deriche, R., Blaszka, T.: Recovering and characterizing im-age features using an efficient model based approach. In: Proceedings of the Conference on Computer Vision and Pattern Recognition, New York, USA, pp. 530–535 (1993)
4. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing, 2nd edn. Pearson Education Limited (2003)
5. Harris, C.: A combined corner and edge detector. In: Alvey Vision Conference, pp. 147–152 (1988)
6. Horn, B.K.P., Schunk, B.G.: Determining Optical Flow. Artificial Intelligence 2, 185–203 (1981)
7. Larhmam, M.A., et al.: A portable multi-cpu/multi-gpu based vertebra localization in sagittal mr images. In: International Conference on Image Analysis and Recognition, ICIAR 2014, pp. 209–218 (2014)
8. Lecron, F., et al.: Heterogeneous computing for vertebra detection and segmentation in x-ray images. International Journal of Biomedical Imaging: Parallel Computation in Medical Imaging Applications 2011, 1–12 (2011)
9. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision (IJCV) 60(2), 91–110 (2004)
10. Mahmoudi, S.A., et al.: Multi-gpu based event detection and localization using high definition videos. In: International Conference on Multimedia Computing and Systems (ICMCS), pp. 81–86 (2014)
11. Mahmoudi, S.A., Kierzynka, M., Manneback, P., Kurowski, K.: Real-time motion tracking using optical flow on multiple gpus. Bulletin of the Polish Academy of Sciences: Technical Sciences 62, 139–150 (2014)
12. Mahmoudi, S.A., Lecron, F., Manneback, P., Benjelloun, M., Mahmoudi, S.: GPU-Based Segmentation of Cervical Vertebra in X-Ray Images. In: IEEE International Conference on Cluster Computing HPCCE Workshop, pp. 1–8 (2010)
13. Mahmoudi, S.A., Manneback, P.: Efficient exploitation of heterogeneous platforms for images features extraction. In: 3rd International Conference on Image Processing Theory, Tools and Applications (IPTA), pp. 91–96 (2012)
14. Marzat, J., Dumortier, Y., Ducrot, A.: Real-time dense and accurate parallel optical flow using CUDA. In: Proceedings of WSCG, pp. 105–111 (2009)
15. Park, K., Nitin, S., Man, H.L.: Design and Performance Evaluation of Image Processing Algorithms on GPUs. IEEE Transactions on Parallel and Distributed Systems 28, 1–14 (2011)
16. Ricardo Possa, P., Mahmoudi, S.A., Harb, N., Valderrama, C., Manneback, P.: A multi-resolution fpga-based architecture for real-time edge and corner detection. IEEE Transactions on Computers 63, 2376–2388 (2014)
17. Sinha, S.N., Fram, J.-M., Pollefeys, M., Genc, Y.: Gpu-based video feature tracking and matching. In: EDGE, Workshop on Edge Computing Using New Commodity Architectures (2006)
18. Tardieu, D., al.: Video navigation tool: Application to browsing a database of dancers' performances. In: QPSR of the numediart research program, vol. 2(3), pp. 85–90 (2009)
19. Yang, Z., Zhu, Y., Pu, Y.: Parallel Image Processing Based on CUDA. In: International Conference on Computer Science and Software Engineering China, pp. 198–201 (2008)
20. Zhu, S., Ma, K.-K.: A new diamond search algorithm for fast block-matching motion estimation. IEEE Transactions on Image Processing 9(2), 287–290 (2000)