GPUs and Multicore CPUs Implementations of a Static Video Summarization

Suellen S. Almeida[1], Edward Cayllahua-Cahuina[1], Arnaldo de A. Araújo[1], Guillermo Cámara-Chávez[2], and David Menotti[2]

[1] Computer Science Department, Federal University of Minas Gerais, Brazil
[2] Computing Department, Federal University of Ouro Preto, Brazil

Abstract. The fast evolution of digital media, in special digital videos, has created an exponential growth of data, increasing the storage and transmission cost and the video content retrieve information complexity. Video summarization has been proposed to circumvent some of these issues and also serves as a pre-processing step in many video applications. In this paper, a static video summarization algorithm is studied and in order to reduce its high execution time, parallelizations using Graphics Processor Units (GPUs) and multicore CPUs are proposed. We also explore a hybrid approach combining both hardware to maximize the performance. The experiments were performed using 120 videos varying frame resolution and video length and the results showed that the hybrid and the multicore CPUs versions reached the best executions times, achieving 4× speedup in average.

Keywords: Video summarization, GPUs, Multicore-CPUs, Parallel algorithms.

1 Introduction

Videos, as a multimedia source, are in continuous generation, e.g., a huge quantity of video is upload to the Internet, security cameras generates hours of videos every day [3]. In order to an efficiently and effectively manage of such amount of video data, video summarization (the process of extracting a concise summary of the content of a video) has been a largely researched subject [13,1,5,11,9].

New methods for video summarization have been proposed in the literature, however they are time consuming [10]. In [3], it is proposed an efficient static video summarization approach, based on temporal video segmentation and visual words obtained by local descriptors. Although this approach generates precise and reliable video summaries, its run time becomes impractical in the processing of high resolution videos with variable duration.

The advances in representations and in the availability of digital videos data generated the need of suitable hardware for their efficiently management, as the new high performance computer architectures, *i. e.,* multicore CPUs and Graphics Processing Units (GPUs). The multicore CPUs allow to implement efficient parallel algorithms in CPU as well as easily perform data parallelism. In contrast,

E. Bayro-Corrochano and E. Hancock (Eds.): CIARP 2014, LNCS 8827, pp. 956–964, 2014.

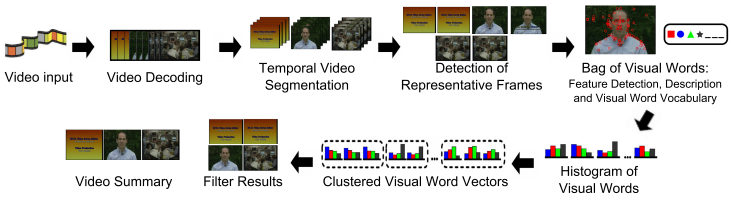

Fig. 1. Static Video Summarization approach

the graphics processors chips started as video processors with fixed function, but they have become increasingly programmable and powerful in terms of computing, being used for general purpose computation.

Lately, different methods of computer vision and image/video processing algorithms using GPUs and multicore CPUs have been presented in literature. Talking about computer vision, several approaches of feature extraction have been parallelized, *e.g.*,, in [6], FAST, HoG and SIFT, three feature extraction algorithms are implemented using an efficient heterogeneous multicore CPUs architecture that achieved great results comparing to GPUs. The SIFT feature extraction is also discussed in [20], but here using GPUs, followed by the KLT feature tracking algorithm, implemented in GPUs as well. Regarding image/video processing, in [12], an approach for parallelization of JPEG compression in GPUs is proposed and the experiments show that the approach works well for 2160p/4k video resolution. In [4], GPUs methods for video encoding and decoding are presented, using as much data parallelism as possible. In [14], it is proposed an interactive retexturing approach for images and videos, also using GPUs, and the experiments achieved high-quality retexturing with real-time feedback.

Having this context in mind, the problem of generating a summary of any video length and resolution in a shorter time can be solved using multicore CPUs and GPUs. In this paper, parallel implementations of the summarization algorithm proposed in [3], using GPUs, multi-core CPUs and an hybrid approach combining these architectures are introduced. The main contributions of this work can be summarized as: 1) The proposition of three parallel algorithms for an automatic video summarization approach. To the best of our knowledge, in the literature, it has not been found any parallelization of video summarization method. 2) The evidence that hybrid multicore CPUs and GPUs algorithms can gather the best of both architectures, significantly improving the runtime.

2 Static Video Summarization Method

The video summarization method used as basis for parallelization in this work is the one proposed in [3]. This method is briefly explained in the following (Fig. 1).

A. ***Video decoding:*** This step can be seen as a quite simply process, however to develop and analysis parallel implementations of video summarization algorithms it is quite important that is detached as a specific step. It basically read/load the input video as a stream and let the frames available for further processing.

B. ***Temporal Video Segmentation:*** The temporal video segmentation is performed by analyzing the cosine dissimilarity measure between RGB color histograms of consecutive frames. High dissimilarity values (threshold 0.4 from [3]) implies that a cut transition has been detected. Dissolve cuts (e.g., fade-in or fade-out transitions) are other type of transitions that are undesirable for video processing. The portions of the video where this effect happens are detected by analyzing the luminance (i.e., gray) variance vector of the video (*i.e.*, the variance of the sequential frames) and then are excluded of any posterior analysis. Dissolve effects are usually located in the valley areas of the variance vector, i.e., the local minimum points, and the procedure adopted in [21,2] is used.

C. ***Detection of Representative Frames:*** Once the video is segmented, the method applies the X-means [18] clustering algorithm to detect the most representative frames for each portion of the video, using the color histograms.

D. ***Bag of Visual Words:*** In order to summarize the videos, the Bag-of-Words (BoW) approach is adopted [22] by considering an image as a document. The "words" are the visual entities present in the image and the objects are described by these visual words. The Bag-of-Visual-Words approach consists of three operations: feature detection, feature description and visual word vocabulary generation. For the first two operations, the summarization method uses the HoG descriptor [7]. This descriptor does the feature detection in blocks of the entire image and then count the occurrences of gradient orientation of them (feature description). A visual word vocabulary, also known as the "codebook" is generated from the feature vectors obtained during the feature detection & description process. To compute the "codewords" all the feature vectors are grouped and classified using a clustering algorithm, such as LBG [15], and the center of the detected cluster define the "codeword".

E. ***Histogram of Visual Words:*** The histogram of visual words is computed by counting the occurrences of the visual words. This is done by using the local image features to find the visual words that occur in the image (representative frame). The occurrences are counted and stored in a vector. As a result, each representative frame will have an associated vector of visual word occurrences.

F. ***Visual Word Vectors Clustering:*** Later, the method utilizes the previously computed visual word vectors and applies the X-means algorithm. Consequently, frames with similar visual entities are clustered together. Afterwards, the nearest frame to the centroid is chosen as the *keyframe* for each cluster. Finally, the detected *keyframes* are sorted according to their time of appearance. At last, these *keyframes* represent the video summary or storyboard.

G. ***Filter Results:*** As an additional step, any possible duplicated *keyframe* is eliminated. This is done by computing a pairwise Manhattan distance between color histograms of consecutive *keyframes*. If a high similarity between two color histograms is detected, the duplicated *keyframe* is deleted.

3 Parallel Versions

In this section, three parallel versions (GPUs, multicore CPUs and hybrid) of the video summarization approach described in Section 2 are presented.

GPUs Version: The parallel GPUs version for the video summarization method is implemented in CUDA [19] version 5.5 and the implementations details of the steps of the method are explained below. Those steps that are not explained here are inexpensive or not high parallelizable, according to our analysis, so they were not parallelized neither discussed.

Video Decoding: The video decoding process is done by the NVCUVID-NVIDIA library [16]. Note that in all three parallel versions, the video decoding step extracts the frames and then write them on the hard disk.

Temporal Video Segmentation: The main idea of the parallelization in this stage is to simultaneously run the operations for the largest possible amount of frames. Thus, if the frames are not in the GPU memory, the GPUs receives frames until their memory capacity is completed and a CUDA three-dimensional (width×height×number of frames) grid of threads performs the computations. In order to compute the color histograms, each thread of the grid accesses one pixel of the frame and increments the correspondent bin at the final histogram. Although computing histograms in CPU is trivial and easy to implement, in GPUs the calculation is not straightforward [19] and multiple threads may simultaneously access the same histogram bin for incrementing, therefore causing writing conflicts among the threads. For solving this issue, atomic increments are used forcing the algorithm to stream the increments. Once the RGB color histograms are computed and stored on GPU memory, each GPU thread computes the cosine dissimilarity of two consecutive histograms. This detection is done by firstly computing the variance vector of the video, which is composed of the luminance (gray) variance of all frames of the video. Taking advantage of the fact that the frames were already in GPUs due to the color histogram calculation, so there is no need to copy data from CPU to GPUs again. The variance computation is done by two CUDA kernels, one to compute the mean and the other to really compute the variance. From the variance vector, a procedure based on the works of [2,21] is applied to conclude the dissolve transition detection. This procedure requires a smoothing spline [8] of the variance vector, which uses sparse matrix multiplication and sparse linear solver. To compute these sparse operations, a library for sparse linear algebra on CUDA called CUSP was evaluated. However, these computations are done very fast on CPU using Eigen library than on GPUs, since there is the overhead of copying the matrices to GPU and an even greater overhead when the matrices are bigger than GPU memory, leading to the need to divide them into small parts. Therefore, the CPU implementation is used to compute those sparse operations.

Detection of Representative Frames: The detection of representative frame is done by the X-means algorithm [18] using the computed RGB color histograms.

Since the X-Means is not the bottleneck of time complexity of the approach and it is not a high parallelizable algorithm for GPUs, the CPU version is used.

Bag of Visual Words: The feature detection and description operations are done in GPU by the OpenCV CUDA's implementation of the HoG descriptor [7]. Initially, the frames are transformed from the RGB color space to the gray one, then the `HOGDescriptor` object is created with standard window, block, and cell sizes and the number of histogram's bins in each cell defined as 9. From the object, the function `getDescriptors` is called and returns the block descriptors computed for the whole image/frame. The generation of the visual word vocabulary is done in CPU by the LBG algorithm. This algorithm was not parallelized because most parts depend on the data calculated on previous iterations as the centers and the distortions.

Histogram of Visual Words: The histogram computation (counting of occurrences of visual words) is done in parallel in a similar fashion to the RGB-histogram explained at the temporal video segmentation step, for all frames.

Parallel Multicore CPUs Version: The multicore CPUs version is implemented using two libraries, OpenMP and the native threads implementation of C++. Below, the same steps implemented on GPUs version but now implemented with multicore CPUs, are explained.

Video Decoding: The OpenCV library is used for simultaneouly decoding the videos using several thread, one for each core of the CPU.

Temporal Video Segmentation: The CPU threads' implementation of this step involves RGB color histograms computation. It is done by dividing the frames between the threads and each one does the computations of a portion of the frames. Once the color histograms are computed, the cosine dissimilarity vector is computed and then refined. Thus, the abrupt changes (shot detection) can be detected. The final stage of video segmentation is to detect and eliminate the dissolve effect of the video and it is done by firstly computing the luminance (gray) variance in each frame of the video. The sparse computations that composes the dissolve detection procedure based on the works of [2,21] is done in parallel by several threads simultaneously using the library Eigen, which is also used in the sequential version (with a single thread performing the operations).

Detection of Representative Frames: The detection of representative frame is done by the X-means algorithm [18] and, in this case, the X-means algorithm is called simultaneously for all the shots detected in the previous step.

Bag of Visual Words: For the feature detector of the representative frames found on the previous step, each thread computes the HOG descriptor for several frames simultaneously. The LBG clustering is used to generate the visual vocabulary and some of its operations are done in parallel with OpenMP, *i.e.,* computing the distance between the visual words and its centers.

Table 1. Parallel version efficiency (speedup-SP) achieved in average of 10 videos for each configuration of resolution (pixels) and length (min). All values are followed by their standard deviation.

Resolution	Length	Sequential CPU	Multicore CPUs		GPUs		Hybrid	
		Runtime	Runtime	SP	Runtime	SP	Runtime	SP
240 × 320	1	77.83 (20.5)	19.02 (2.8)	4.1	42.97 (8.2)	1.8	21.61 (3.5)	3.6
	3	249.41 (35.4)	58.15 (7.4)	4.3	121.39 (20.7)	2.1	69.53 (7.9)	3.6
	5	259.34 (41.3)	95.13 (18.9)	2.7	226.83 (35.0)	1.1	104.76 (16.7)	2.5
	10	529.23 (59.6)	191.12 (26.8)	2.8	490.75 (83.2)	1.1	230.39 (24.6)	2.3
360 × 640	1	111.15 (27.7)	37.46 (10.1)	3.0	63.44 (13.7)	1.8	37.42 (8.8)	3.0
	3	314.38 (55.2)	112.16 (20.6)	2.8	251.38 (46.7)	1.3	116.13 (21.5)	2.7
	5	506.36 (100.2)	197.01 (38.7)	2.6	383.20 (64.6)	1.3	189.89 (35.1)	2.7
	10	945.18 (152.6)	366.64 (56.6)	2.6	710.53 (119.7)	1.3	378.68 (59.4)	2.5
1080 × 1920	1	767.65 (233.5)	262.94 (73.0)	2.9	541.34 (154.2)	1.4	264.50 (72.2)	2.9
	3	2,314.88 (606.4)	779.30 (171.1)	3.0	2,008.80 (348.6)	1.2	776.47 (200.8)	3.0
	5	3,447.93 (557.1)	1,303.16 (280.9)	2.6	2,690.69 (512.9)	1.3	1,328.10 (275.9)	2.6
	10	5,310.85 (759.3)	2,279.94 (583.6)	2.3	4,239.97 (698.1)	1.3	2,381.36 (481.3)	2.2

Histogram of Visual Words: The HoG features that were computed are used to count the occurrences of visual words in the images. In this case, each thread computes the histogram of each visual word. It is important to highlight that this version is faster than the GPUs version on operations of reading and writing files on the hard disk, because the multicore hardware allows us to do these operations simultaneously, what cannot be done in GPU.

Hybrid Multicore CPUs & GPUs Version: This parallel version puts together the best parts (faster runtime) of the multicore CPUs version with the best parts of the GPUs version, resulting in an improved version of parallel automatic video summarization.

As is shown further in Section 4, comparing the video decoding runtime of multicore CPUs and GPUs without copying and writing overheads, the results of both versions are very similar. Therefore, the multicore CPUs version performs well in this case, and as stated earlier, the CPU threads allow to read/write files simultaneously and GPU ones do not.

The computations of histograms, variances and cosine dissimilarities, parts of temporal video segmentation obtained better results with GPUs version for some video configurations and on other cases, the multicore CPUs reached the best results. In this case, the GPUs version is chosen, because if the GPU had more memory, certainly it would perform well for all videos. The detection of the representative frames with multicore CPUs is used in this version.

The local descriptor used for feature detection was the HoG descriptor and the implementations based on different kind of parallelism. On one hand, for multicore CPUs, the threads computed the descriptors for many frames simultaneously. On the other hand, the GPUs version used a parallel algorithm to calculate the HoG descriptors of the frames. Again, the results were similar and the GPUs version was chosen.

Finally, the LBG clustering and the histogram of visual words were computed faster with multicore CPUs and so this version is used for these tasks.

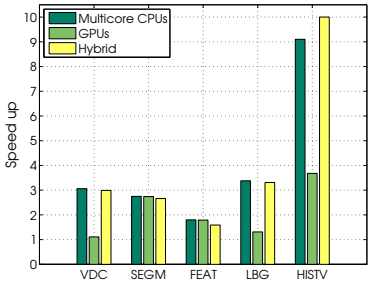

Fig. 2. Average speedup achieved in comparison with sequential version for five video summarization steps: Video Decoding (VDC), Temporal Video Segmentation (SEGM), Features Extraction/Description (FEAT), Visual Word Vocabulary (LBG) and Histogram of Visual Words (HISTV).

4 Experiments and Results

The experiments to measure the efficiency of the proposed parallel implementations were performed using an Intel Core i7 CPU computer, with 16GB RAM, and a NVIDIA GeForce GTX 650, which contains 384 CUDA cores and 1GB of dedicated memory, on Windows 8.1 (64-bits). The video database used in the experiments was created by us from YouTube videos of different genre, with three video resolutions (1920×1080, 640×360, and 320×240 pixels) and four video lengths (1, 3, 5, and 10 minutes). For each combination of resolution and length were chosen ten videos, adding up 120 videos. The entire database and the parallel implementations used in this work are publicly available in [17]. All the execution times were measured in seconds.

Table 1 shows the average of total execution time of the three parallel implementations (Multicore CPUs, GPUs and Hybrid) and also the sequential version. As the video content differs a lot, the execution time of each video also varies. The video runtime is proportional with the shots that the video has. With more shots, we have more representative frames, so the features extraction takes longer and the same occur with the others steps. To realize this variation, we also computed the standard deviation and it is showed in Table 1 (in parenthesis). The execution time is also proportional to video length and resolution, as expected.

The execution time results showed that the Multicore CPUs and Hybrid version achieved the best speedup, reaching 4.3× and 3.6×, respectively. Moreover, the GPUs presented the worst results. This fact occurred because the time measured is for the entire algorithm, including the parts that were not parallelized and the parts that could not run in GPU, *i.e.,* loading and saving files.

The speedup obtained for each parallelized step of the video summarization method is illustrated in the bar graphs in Figure 2. The video decoding with multicore CPUs (≈ 3×) performed better than GPUs (≈ 1.1×), and it is very close to the hybrid version because the same algorithm was used. The parallelizations of temporal video segmentations showed results very similar, ≈ 2.7×, when GPUs and multicore CPUs presented a small advantage and the same occurred with feature description/detection stage (≈ 1.8×). For the creation

of the visual word vocabulary with LBG clustering, only the multicore CPUs (this implementation was also used on hybrid version) was parallelized, so, as expected, the GPUs execution time was the same as sequential and multicore CPUs gained in this experiment ($\approx 3.4\times$). Finally, the computation of histogram of the visual words was faster with multicore CPUs (and hybrid), achieving a speedup $\approx 9.5\times$, while GPUs presented $\approx 3.7\times$ speedup.

5 Conclusions

This paper presented three parallel implementations for a static video summarization approach [3] that takes advantages of temporal video segmentation and bag of words. The first parallel implementation uses the Graphics Processor Units (GPUs), which performed well the video summarization steps that includes a lot of small math computations for big data, *i.e.,* computations of histograms and variances of the video segmentation step and calculation of local features. The second one used multicore CPUs, which presented great results for data parallelism, *i.e.,* loading/saving files and decoding the videos. Finally, the last one combines the best parts of the previous versions, referred as hybrid version. For some videos the speedup achieved is about $5\times$.

Acknowledgments. The authors would like to thank CNPq, CAPES and FAPEMIG, Brazilian funding agencies, for the financial support to this work.

References

1. Almeida, J., Leite, N.J., da, S., Torres, R.: Vison: VIdeo summarization for ONline applications. Pattern Recognition Letters 33(4), 397–409 (2012)
2. Camara Chavez, G., et al.: Shot boundary detection by a hierarchical supervised approach. In: IWSSIP, pp. 197–200 (2007)
3. Cayllahua-Cahuina, E.J.Y., Cámara-Chávez, G.: A new method for static video summarization using local descriptors and video. In: SIBGRAPI (2013)
4. Cheung, N.M., Fan, X., Au, O., Kung, M.C.: Video coding on multicore graphics processors. IEEE Signal Processing Magazine 27(2), 79–89 (2010)
5. Ciocca, G., Schettini, R.: Erratum to: An innovative algorithm for key frame extraction in video summarization. JRTIP 8(2), 225–225 (2013)
6. Clemons, J., et al.: Effex: An embedded processor for computer vision based feature extraction. In: DAC, pp. 1020–1025 (2011)
7. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR, vol. 1, pp. 886–893 (2005)
8. De Boor, C.: A practical guide to splines, vol. 27. Springer, New York (1978)
9. Evangelio, R., et al.: Video indexing and summarization as a tool for privacy protection. In: DSP, pp. 1–6 (2013)
10. Furini, M., et al.: On using clustering algorithms to produce video abstracts for the web scenario. In: CCNC, pp. 1112–1116 (2008)
11. Guan, G., et al.: Video summarization with global and local features. In: ICMEW, pp. 570–575 (2012)

12. Holub, P., et al.: Gpu-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2k, and 4k video. FGCS 29(8) (2013)
13. Kuanar, S.K., et al.: Video key frame extraction through dynamic delaunay clustering with a structural constraint. JVCI 24(7), 1212–1227 (2013)
14. Li, P., et al.: Interactive image/video retexturing using GPU parallelism. Computers & Graphics 36(8), 1048–1059 (2012)
15. Linde, Y., Buzo, A., Gray, R.: An algorithm for vector quantizer design. IEEE Transactions on Communications 28(1), 84–95 (1980)
16. NVIDIA: NVIDIA CUDA Video Decoder. NVIDIA (2010)
17. Omitted: Parallels implementation for temporal video summarization and databases (2014), `https://github.com/omitted/tsumm`
18. Pelleg, D., Moore, A.W.: X-means: Extending k-means with efficient estimation of the number of clusters. In: ICML, pp. 727–734 (2000)
19. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming, 1st edn. Addison-Wesley Professional (2010)
20. Sinha, S.N., et al.: Feature tracking and matching in video using programmable graphics hardware. Machine Vision and Applications 22(1), 207–217 (2011)
21. Won, J.U., et al.: Correlation based video-dissolve detection. In: ITRE, pp. 104–107 (2003)
22. Yang, J., et al.: Evaluating bag-of-visual-words representations in scene classification. In: MIR, pp. 197–206 (2007)