PARALLEL MOTION ESTIMATION IMPLEMENTATION FOR DIFFERENT BLOCK MATCHING ALGORITHMS ONTO GPGPU

Eduarda Monteiro, Marilena Maule, Felipe Sampaio, Cláudio Diniz, Bruno Zatt, Sergio Bampi

{ermonteiro, mmaule, fmsampaio, cmdiniz, bzatt, bampi}@inf.ufrgs.br

Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

Abstract

This work presents an efficient method to map Motion Estimation (ME) algorithms onto General Purpose Graphic Processing Unit (GPGPU) architectures using CUDA programming model. Our method jointly exploits the massive parallelism available in current GPGPU devices and the parallelization potential of ME algorithms: Full Search (FS) and Diamond Search (DS). Our main goal is to evaluate the feasibility of achieving real-time high-definition video encoding performance running on GPUs. For comparison reasons, multi-core parallel and distributed versions of these algorithms were developed using OpenMP and MPI (Message Passing Interface) libraries, respectively. The CUDA-based solutions achieve the highest speed-up in comparison with OpenMP and MPI versions for both algorithms and, when compared to the state-of-the-art, our FS and DS solutions reach up to 18x and 11x speed-up, respectively.

1. Introduction

Among all innovative tools featured by the latest video coding standards, e.g. H.264/AVC [1], the ME still provides most of compression gains by reducing the temporal redundancy between frames. ME employs search algorithms to find in the reference frames (the previously reconstructed frames) the most similar regions to the current frame. Full Search (FS) ME algorithm is known as optimal since it finds the best match by exhaustively searching in the reference frames. Typically the best match is defined considering the Sum of Absolute Differences (SAD) between blocks [2]. To reduce the number of SAD calculations, the search may be constrained to a search area (a region in the reference frame). However, the FS still requires increased computational effort. To reduce this complexity, fast motion estimation algorithms have been proposed [2]. Diamond Search (DS) [3] is a fast ME algorithm that significantly reduces the ME complexity by reducing the amount of SAD calculations while keeping the video quality near to FS performance. The DS employs two search patterns: the Large Diamond Search Pattern (LDSP), with 9 SAD calculations, and the Small Diamond Search Pattern (SDSP), with 4 SAD calculations for final refinement.

These Motion Estimation algorithms present a high potential of parallelization and are proper for implementation on parallel architectures. This potential may be exploited by using the massively parallel GPUs (Graphic Processing Units) available in most of the current computers. Using CUDA (Compute Unified Device Architecture) [4], proposed by NVIDIA in 2007, that provides a programming API for GPUs, it is possible to use the GPUs for general purpose processing.

Different video encoding software solutions have been developed, e.g. JM H.264 reference software [7], *x264* free software library [8]. However, they have no GPU acceleration support or use proprietary libraries for this purpose (the case of x264). Research works that aim to accelerate video encoding using GPU can be found in the literature. The authors focus specifically on the implementation of distinct ME algorithms onto GPU, which is directly related to the scope of this work. These solutions consider the implementation of different versions of ME algorithms considering variable block size and multiple reference frames.

Chen *et al*. [9] presented a ME FS algorithm for GPU implementation using CUDA architecture. This work suggests the ME in different steps to achieve high parallelism through and low data transference between CPU and GPU memories. The work considers variable block sizes (*16x8*, *8x16*, *8x8 – 8x4*, *4x8*, *4x4*) and spends large processing time to decide the size of the block that will be used. Lin *et al*. [10] proposed an algorithm based on a multi-pass encoding technique for the FS algorithm. This solution considers four Motion Estimation loops. The main drawback of this approach is the performance limitation imposed by multiple iteration steps for SAD calculation and SAD values comparisons. Lee *et al*. [11] presents three alternatives of ME in GPU based on FS algorithm: Integer Accuracy, Fractional Accuracy and Integer Accuracy considering three reference frames in parallel. In this work four loops were considered. Cheng *et al.* [12] suggest block-based ME techniques, such as FS, TSS (Three Step Search), FSS (Four Step Search) and DS. The strategy applied in this work is based in use as many threads as possible to accelerate the computation.

By exploring the maximum inherent parallelism potential of FS and DS and the available parallel processing capability of recent GPUs, this work presents an efficient method to map FS and DS algorithms onto GPGPU architecture using CUDA programming model aiming to achieve real time processing (up to 30 frames per second – 30fps) in the entire video encoding. Further, for comparison purpose, we compared the CUDA solutions with a parallel implementation for multi-core GPP using OpenMP library [5] and with a distributed implementation to run onto cluster/grid machines using Message Passing Interface (MPI) library [6]. The

performances of our GPU solutions are extensively compared, using real video sequences. Our GPU solutions are also compared to state-of-the-art ME implementations onto GPUs. This paper is organized as follows. In Section 2, are described the ME FS and DS implementations on parallel and distributed architectures. Section 3 shows the results, analysis, and comparisons with state-of-the-art and between ME algorithms. Section 4 concludes the work.

2. Motion Estimation Parallelization

By analyzing the computational complexity introduced by ME, this work proposes a highly parallelizable solution for exhaustive and fast algorithms on a massively parallel platform, the video graphic cards (GPU).

The parallelization strategy for the algorithms (FS and DS) proposed in this work is presented in fig. 1.

Initially, the video sequence is loaded from a YUV video file and, according to the processing order, the current frame and the reference frames are sent to the GPU device.

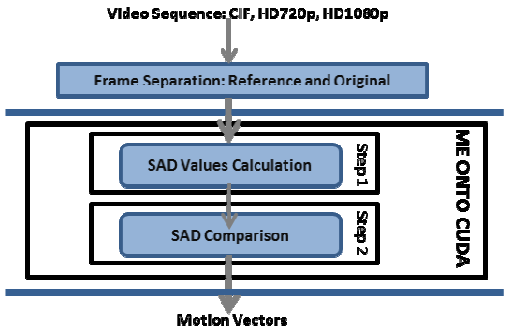


Fig. 1 – Proposed ME Algorithms Flow.

The parallel ME is composed by two steps: ***(i)*** SAD values calculation, and ***(ii)*** comparison of SAD values (the lowest SAD is chosen). Finally, the motion vectors are generated for each current frame and transferred back to CPU.

2.1. Motion Estimation Parallelization onto CUDA Architecture

The hardware platform used in this work is composed by a CPU and a NVIDIA GPU that supports CUDA. There are two communications between CPU and GPU in both algorithms presented (FS and DS): *i)* reference and current frames are sent to GPU from CPU; *ii)* resulting data (motion vectors and SAD values) from GPU to CPU. The ME programming model considers the CUDA hierarchy as shown in the fig. 2.

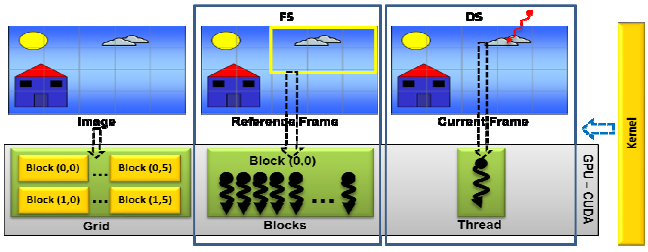


Fig. 2 – CUDA Programming Models – Algorithm Allocation.

The ME parallelization in GPU is based in only one *kernel* (parallel procedure executed in GPU). It is responsible for execution of the ME block matching algorithms onto CUDA (FS and DS): the SAD values calculation and the SAD values final comparison. The parallelization of FS algorithm onto CUDA was performed by considering the following entities: ***(i) thread:*** each thread is responsible for the computation of one *4x4* video block: the *4x4* video block is the basis for the block matching operation, since it allows finer motion granularity compared to the *macroblock* (*16x16* pixels); ***(ii) block:*** the size of the blocks in this application is variable according to the size of the search area in FS algorithm (see fig. 2 - FS) and, ***(iii) grid:*** the grid size is related to the number of current blocks that compose the video. The DS parallelization onto CUDA requires some changes in the programming model: ***(i) thread***: each thread that composes the GPU block is responsible to execute the ME for one current block (see fig. 2 - DS); ***(ii) block***: the block size refers to maximum number of threads that the graphic card allows; ***(iii) grid***: the grid size is also related to the number of current blocks that compose a image.

2.2. ME onto Multi-core Processors and Distributed Platforms

To establish a comparative basis for our CUDA-based algorithms, we also implemented a sequential version, a parallel OpenMP-based version [4] and a distributed and parallel MPI-based version [5] of the ME FS and DS algorithms. Both OpenMP and MPI versions (FS and DS) implement a 4x4 block comparison in each thread (as well as the algorithm flow in fig. 2). Then, these implementations are based on two

communications: ***(i)*** one broadcast of the current block; ***(ii)*** one broadcast of the reference frame. Finally, the SAD values are compared to define the *best match*.

3. Experimental Results

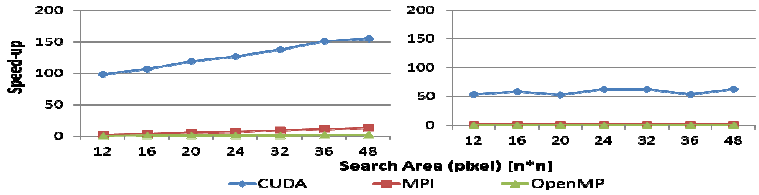
In this section we present the experimental for FS and DS ME algorithms comparing CUDA, OpenMP and MPI solutions. These tests were performed using three video resolutions (CIF, HD720p and HD1080p) for search areas ranging from *12x12* pixels to *128x128* pixels, considering a current block of *4x4* pixels size.

The experimental setup features a NVIDIA GTX480 @ 1.4GHz (480 CUDA cores) connected via PCI-Express interface through an Intel Core2Quad Q9550 @ 2.82GHz CPU. For MPI processing the *Xiru* cluster was used (Dell PowerEdge 1950). It is composed by 14 dual-CPU nodes and four cores per CPU (Intel Xeon CPU E5310 @1.60GHz). This infrastructure is part of the Grid5000 since 2010 [12].

Fig. 3 shows the speed-up achieved in relation to the sequential implementation for both algorithms (FS and DS). For these results *Blue Sky* video sequence (HD1080p resolution) was used. Four nodes and eight CPUs (32 processes) are used in MPI version, while four threads are used in the OpenMP version.

For FS, the speed-up obtained by CUDA version presented a linear growth in relation to the size of the search area, while for OpenMP version the speed-up is kept constant. The obtained speed-ups of CUDA, MPI and OpenMP versions are *154x*, *13x* and *2x*, respectively. CUDA version achieved a *14x* and *77x* speed-up compared to MPI and OpenMP, respectively.

DS algorithm presented *62x*, *4x* and *2x* speed-up for CUDA, MPI and OpenMP versions, respectively. Speed-ups are calculated in relation with the results obtained with the sequential version execution. CUDA version achieved a 77*x* speed-up in relation to MPI and *191x* compared to OpenMP.



(a) FS (b) DS

Fig. 3 – HD1800p ME (a) FS and (b) DS - Speed-up Results (CUDA vs. OpenMP vs. MPI vs. Sequential).

Fig. 4 shows the timeline of the CUDA implementations execution, including the four steps that imply the total time execution this algorithm in GPU: data allocation, data transfer from CPU to GPU, kernel execution and data transfer from GPU to CPU.

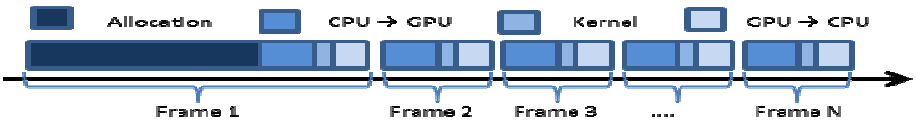


Fig. 4 – Timeline - DS Execution.

Analyzing fig. 4 we observe that the allocation time for DS represents 83.78% of the total ME time of the first frame while the CPU-GPU data transfer, kernel execution and GPU-CPU data transfer represent 12.87%, 0.89% and 3.22%, respectively. For FS the allocation spends 46.62% of total encoding time of the first frame (26.99%, 0.89% and 25.49% for the other steps, respectively).

Since the allocation is the phase where the internal GPU memory is reserved for the required ME reference data, this step is only necessary in the beginning of the video encoding. Since this initial latency is amortized along the frames processing, the execution time per frame will be represented in this work only by ***(i)*** the CPU to GPU transfer time, ***(ii)*** the kernel execution time and ***(iii)*** the GPU to CPU results transfer time.

Tab. 1 presents the performance results of the proposed implementations for the three target resolutions. These results consider only the ME execution time.

Tab 1 - ME Performance Results

	CIF	**HD720p**	**HD1080p**
DS (fps)	1754 fps	310 fps	158 fps
FS (fps)	34 fps	33 fps	43 fps
	48x48	*24x24*	*12x12*

The ME FS achieves real time processing due the high level of parallelization employed in our approach. Since the FS analyzes all possible candidate blocks, the search area is determinant for the performance. For larger resolutions, the real time processing is achieved using smaller search area dimensions, as it can be seen in tab. 1. The ME DS surpasses the real time processing for all tested video sequences. These results consider the maximum number of iterations necessary to the algorithm converge. Fig. 5 presents the total time of the DS computation for CUDA implementation when compared with the minimum real time requirement, 30 fps.

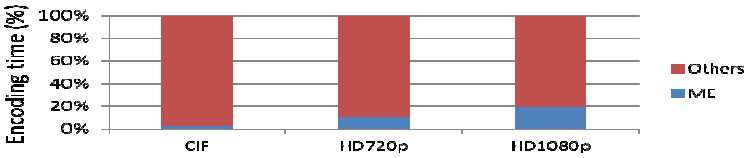


Fig. 5 – Timeline - DS Execution.

If the proposed DS CUDA implementation is used in combination with a CPU approach for the entire H.264/AVC encoder, it will represent only 20% of the total required execution time for real time processing, in the worst case scenario when HD1080p videos are the target. In addition, this work is also compared to related works [8-10] where the features used are presents in tab. 2. The works in tab. 2 consider integer pixel accuracy ME for NVIDIA boards using CIF [8-10] and HD1080p resolutions [11]. The speed-ups results presented include *9x9*, *16x16* and *32x32* pixels search area showing that our algorithm achieves the highest speed-up considering different search areas for FS and DS. The FS algorithm in [9] achieved the highest speed-up among others works that propose FS parallelization onto GPUs. Our CUDA FS version achieved a *3x* and *1x* speed-up increase compared to [9] for *16x16* and *32x32* search areas, respectively. Additionally, a gain of *18x* over the work in [11] is observed, for *9x9* search area. Moreover, the DS algorithm version also presents the best result in comparison with state-of-the-art. A speed-up increase of *11x* over [11] is observed, considering *9x9* pixels search area.

The results were achieved due to our efficient mapping of FS and DS algorithms to the threads in CUDA architecture. Also, the device we have used in our experiment is faster and has more cores than those used in related work (*2x* more than in [11]). However, our speedups are at least one-order-of-magnitude higher than related work [8-11]. It is shown that algorithm mapping is scalable and benefits from the increase of computation cores of GPUs.

Table 2 - Related Works Comparison

	[10]	**[9]**	**[8]**	**[11]**	**This Work**
#Cores	-	-	128	192	**480**
Memory	Shared	Texture	Texture	Shared / Texture	**Global**
BMA	FS	FS	FS	FS/DS	**FS/DS**
Level	Block	Pixel	Pixel	Block	**Block**
Results - Speedup					
FS 9x9	n.a.	n.a.	n.a.	8.76	**148.18**
FS 16x16	1.79	12.08	n.a.	n.a.	**41.87**
FS32x32	2.18	26.76	10.38	n.a.	**34.03**
DS 9x9	n.a.	n.a.	n.a.	6.86	**70.11**

4. Conclusions

This paper presented Motion Estimation module implementation in GPU architecture, for the *Full Search and Diamond Seach* algorithms. The parallelized algorithms were developed and mapped in NVIDIA CUDA architecture. MPI, OpenMP and sequential versions were also implemented for comparison purposes with GPU. The performance results presented by CUDA implementations meet the constraints for the real-time encoding of HD720p (FS and DS) and HD1080p (DS). If compared to [9] (which provides the best results in the literature), our *Full Search* solution achieves *3x and 1x* speed-up increase for *16x16* and *32x32* search areas, respectively. Our *Diamond* Search for CUDA provides *11x* speed-up for *9x9* search area when compared to [11]. These gains were achieved through an optimized mapping of FS and DS algorithms to the CUDA.

5. References

[1] ITU-T Recommendation H.264/AVC (03/10): advanced video coding for generic audiovisual services, 2010.

[2] Kuhn, P., *Algorithms, Complexity Analysis and VLSI Architectures for MPEG4 Motion Estimation*, Boston: Kluwer Academic Publishers, 1999. 239 p. ISBN 0-7923-8516-0.

[3] NVIDIA CUDA Programming Guide 1.1. www.nvidia.com

[4] The OpenMP API specification. http://openmp.org/wp

[5] The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/projects/mpi

[6]JM H.264 v. 14.2. http://iphome.hhi.de/suehring/tml/download

[7] x264 codec. http://www.videolan.org/developers/x264.html

[8] W-N. Chen, H-M. Hang. “H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA)”, ICME 2008, pp. 697-700

[9] Y-C. Lin, P-L Li, C-H. Chang, C-L. Wu, Y-M. Tsao, S-Y. Chien, “Multi-pass algorithm of motion estimation in video encoding for generic GPU”, ISCAS 2006, pp. 4451-4454

[10] C-Y. Lee, Y-C. Lin, C-L. Wu, C-H. Chang,Y-M. Tsao, S-Y. Chien. “Multi-Pass and Frame Parallel Algorithms of Motion Estimation in H.264/AVC for Generic GPU”, ICME 2007, pp. 1603-1606

[11] R. Cheng, Y. Eryan, T. Liu. “Speeding Up Motion Estimation Algorithms on CUDA Technology”, *PrimeAsia*, 2010.

[12] Grid’5000. www.grid5000.fr