Improved approach for full search motion estimation on GPU

FATMA ELZAHRA SAYADI [1-2], MARWA CHOUCHENE [1], HAITHEM BAHRI [1], OLFA HAGGUI[3], BOURAOUI OUNIR[3]
[1]Electronics and Microelectronics Laboratory, FSM,
University of Monastir
[2] Issat sousse, University of Sousse
3 Networked Objects Control & Communication Systems Lab (NOCCS), ENISO
University of Sousse
Tunisia
Sayadi_fatma@yahoo.fr

Abstract: - In order to speed up video coding efficiency such as H.264/AVC and H265/HEVC, we propose in this paper a parallel approach of full search (FS) algorithm for motion estimation on Graphic Processor Unit (GPU). We implemented the traditional sequential FS algorithm for motion estimation to computing unified device architecture (CUDA) optimizing memory usage, taking full ad-vantage of the powerful parallel computing capability to speed up FS motion estimation.
Experimental results show that our implementation on CUDA demonstrates substantial improvement up to 48 times than CPU counterpart available and can effectively speed up the FS for motion estimation.

Key-Words: - Full search, GPU, CUDA, Motion Estimation, shared memory, Optimization

1 Introduction

The joint Video Team (JVT) developed the H.264 or MPEG.4advanced video cod-ing(AVC) standardwich is widely deployed in many online vdeo streaming sys-tems.The emergence of UHD video services and the ubiquity of online video stream-ing have created a demand for coding efficiencies superior to H264/AVC. To this end MPEG and VCEG worked together to develop a new H.265 or High efficiency Video coding (HEVC) standard. This new standard, doubles the video compression ratio compared to its predecessor at the same video quality without any degradation. However, this important performance is achieved by increasing the encoder compu-tational complexity and thus from 2 to 10 times compared to the previous H264/AVC [1, 2]. That is why the decrease of HEVC video coding time, becomes a hot research topic in recent years.
In this context, we provide in this work a methodology for migrating full serach motion estimation algorithm from a processor onto the GPU in order to reduce the computational complexity of the video coding standard.
Because of their capacity, parallel GPUs are used to perform a partitioning of a complex task into a number of basic tasks, which can be made by its different pro-cessors working simultaneously, thus making the overall execution much faster than the sequential one.
In this paper, we will present the basic principles of the motion estimation algo-rithm as well as the parallelization techniques of such process on graphics processors.
Our work is organised as follows: Section 2, is devoted to the detailed description of the Full search algorithm. The third section is dedicated to the study and imple-mentation of this algorithm on CPU and GPU. In this section, a performance evalua-tion was also performed. Finally we conclude the paper.

2 Parallel Full Search on GPU

Motion Estimation is defined as searching the best motion vector, which is the co-ordinate of the best similar block in previous frame for the block in current frame. Of various approaches for motion estimation, the block-matching algorithm (BMA) is very popular in the framework of generic coding [3], [4]. Block-based matching algo-rithms find the optimal motion vectors which minimize the matching difference be-tween reference block and candidate blocks. Therefore, the same motion vector is used for all pixels within a block unified device architecture. The most straightfor-ward BMA is the full search (FS), which exhaustively searches for the best matching block within the search window.

However, FS yields very high computational com-plexity and makes ME the main bottleneck in real-time video coding applications. In order to improve video coding efficiency, we try to accelerate highly computational ME on Graphic Processor Unit (GPU) which is featured with parallel computing to greatlyimprove the processor performance.

2.1 Experimental Setup

The experiments were performed on a personal computer with an Intel Core I7-3770 3.40GHz CPU with 8GB memory and a graphic card Nvidia GeForce GTX480. This card is based on the GF100 Fermi chip which is produced in 40 nm at TSMC. The GTX480 uses only 480 of the 512 shaders of the GF100 silicon [5].

The software was coded in the programming language with CUDA Toolkits version 5.0 and the developed software was created in Visual C++ 2010.

2.2 Methodology

This section describes the approaches toward parallelism as well as different opti-mization strategies. Our goal was to encode video using the motion estimation algo-rithm in real-time. We have used the full search block-matching algorithm based on the sum of absolute differences (SAD).

In fact SAD is an extremely fast metric due to its simplicity. It is also easily paral-lelizable since it analyzes each pixel separately, making it readily implementable with popular parallel programming models such as GPUs [6].

After the parallelization has been done using CUDA, computational tests have been carried out on several test sequences, CIF, QCIF, HD720p, and HD1080p. The obtained results are hence presented and analyzed.

We begin with a simple version of Full search Motion implementation in order to find the optimal block displacement. The implemented algorithm applies the SAD error criterion which is an extremely simple video quality metric used for block-matching in motion estimation for video compression. It takes the absolute value of the difference between each pixel in the original block and the corresponding pixel in the block being used for comparison, and sums them up. The Full search Motion implementation Kernel creates a thread for each result element for the SAD compu-tation. Many threads are created in an attempt to hide the latency of global memory by overlapping execution. Each thread works on its column in 16x16 block. Once all blocks from the search range are iterated and the best match found, the output vec-tor is transferred to the global memory.

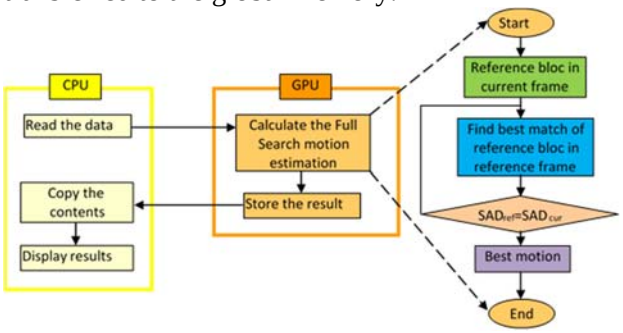

Fig. 1. Flow chart of the proposed Parallel full search algorithm

There are different methods to estimate a CUDA kernel runtime [7, 8], of. The most portable option is to use CUDA's built-in timer functions, which will function across different operating systems. To measure runtime, we need to create and start a timer before calling the kernel. After calling the kernel, we need to make sure that the kernel has finished, then stop, and read the timer.

Here cutStartTimer() and cutStopTimer() are used to place the start and stop events into the default stream, stream 0. The GPU will register a timestamp for the event when it attains that event in the stream. The cutGetTimerValue() function re-turns the time past between the recording of the start and stop events[9]. This value is formulated in milliseconds and has a resolution about of half a microsecond.

We should note that the timings are evaluated on the GPU clock, so the timing res-olution is operating-system-independent. We should mention also that the execution time computed for motion estimation is done on two video sequence successive im-ages.

Execution times on the Intel core i7 and those on the GTX480 are shown in table 1. The execution times indicate that the implementation scaled well for larger image sizes on GPUs. In fact, for HD1080p format we were able to achieve an execution time of 45,2 ms on the GTX480 against 962,7ms on serial execution, indicating a significant speedup.

	CPU execution time (ms)	Gpu execution time (ms)
Cif	12,43	3,68
QCIF	60,66	12,02
HD720p	520,8	23,1
HD1080p	962,7	45,2

Table 1. Execution time for Intel corei7 vs GTX480

The proposed implementation promises improved utility in real time motion esti-mation, since the

performance increase has multiplied and the execution time is lower than highly optimized libraries executing on a CPU.

2.3 Optimized implementation

In an effort to further improve performances, we attempt to apply a CUDA memory optimization strategies. Based on shared memory.

In the initial Kernel version, we have used a two-dimensional variable in local memory to keep the values of SADs. In the second version we will plazce them in the shared memory.

We will properly attribute elements of this new __shared__ array to the threads of the thread block thus all elements of the new virtual private array for each thread are placed in its own shared memory bank as shown in Fig. 2.

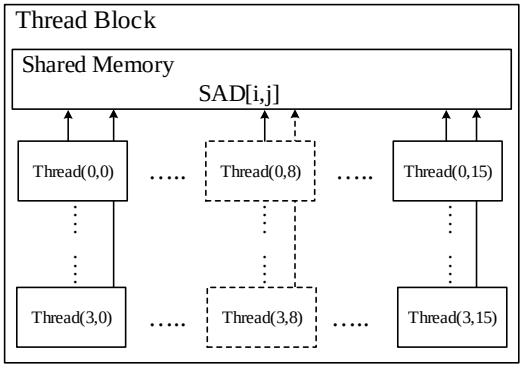

Fig. 2. Shared memory usage

Table 2 shows the time required to perform motion estimation on different se-quences on Gpu using shared memory.

Table2. Execution time using local memory vs. shared memory

	CPU execution time (ms)	GPU execution time (ms)	
		Local memory	Shared memory
Cif	12,43	3,68	0,4
QCIF	60,66	12,02	2,8
HD720p	520,8	23,1	10,84
HD1080p	962,7	45,2	19,96

In Table 2, we show the execution times using local memory versus shared memory. At first sight, we note that the times obtained with the shared memory are lower than those obtained with the local memory

3 Conclusion

We have presented in this paper an original Full search Motion Estimation GPU implementation that can be applied on both H264 and HEVC encoders. This imple-mentation is based on a single CUDA Kernel which is in charge of the SADs, the best SAD and the Motion Vector computations. We have started with the local memory which is used only to hold automatic variables. Experimental results show that our method provides up to 21 speed-up compared with single CPU core implementation.

To improve even more this speed up, we provided an optimized implementation based on shared memory in order to track down memory bottlenecks. The proposed algorithm based on the shared memory allowed an increase in speed up reaching 48.

References:

[1] Jens-Rainer Ohm, Gary J. Sullivan, Heiko Schwarz, Thiow Keng Tan, and Thomas Wiegand, "Comparison of the coding efficiency of video coding standards–including high efficiency video coding (HEVC)," IEEE Transactions on CSVT, Vol 22, No.12, 1649-1668, Dec 2012.

[2] Dong Zhang, Bin Li, Jizheng Xu, and Houqiang Li, “Fast Transcoding from H. 264 AVC to HEVC,” 2012 IEEE International Conference on Multimedia and Expo (ICME), Melbourne, Australia, July 2012.

[3] H. G. Musmann, P. Pirsch, and H. J. Grallert, “Advances in picture coding,”Proc. IEEE, vol. 73, pp. 523–548, Apr. 1985.

[4] F. Dufaus and F. Moscheni, “Motion estimation techniques for digital TV: A review and a new contribution,”Proc. IEEE , vol. 83, pp. 858–876,Jun. 1995

[5] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications

[6] Mark Harris (2007) Optimizing Parallel Reduction in CUDA NVIDIA Developer Technology

[7] CUDA API REFERENCE MANUAL (2012) Version 4.2

[8] Farber, R.:‘Cuda application Design and Development’ (Morgan Kaufmann, Elsevier, 1st edn., 2011

[9] couturier, R.: ‘Designing Scientific Application on GPUs’ (CRC Press, Taylor Francis Group, 2014