MULTIVIEW ENCODER PARALLELIZED FAST SEARCH REALIZATION ON NVIDIA CUDA

Chih-Te Lu[1] and Hsueh-Ming Hang[1,2]

[1]Dept. of Computer Sci. and Inform. Technology, National Taipei University of Technology, Taipei, Taiwan

[2]Department of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan

ABSTRACT

NVIDIA announced a powerful GPU architecture called Compute Unified Device Architecture (CUDA) in 2007, which is able to provide massive data parallelism under the SIMD architecture constraint. We use NVIDIA GTX-280 GPU system, which has 240 computing cores, as the platform to implement a very complicated video coding scheme, the Multiview Video Coding (MVC) scheme. MVC is an extension of H.264/MPEG-4 Part 10 AVC. It is an efficient video compression scheme; however, its computational compexity is very high. Two of its most time-consuming components are motion estimation (ME) and disparity estimation (DE). In this thesis, we propose a fast search algorithm, called multithreaded one-dimensional search (MODS). It can be used to do both the ME and the DE operations. We implement the integer-pel ME and DE processes with MODS on the GTX-280 platform. The speedup ratio can be 89 times faster than the CPU only configuration. Even when the fast search algorithm of the original JMVC is turned on, the MODS version on CUDA can still be 20 times faster.

Index Terms— Multiview video coding (MVC), H.264/AVC, motion estimation, disparity estimation, parallel, CUDA, GPU, Multi-core, fast search algorithm

1. INTRODUCTION

The deployment of 3D techniques in recent years has increased remarkably. As the technology of 3DTV [1] and free view point TV (FTV) [2] is getting more and more mature, the Multi-View Video Coding (MVC) technology becomes one of crucial IT industry components. In a 3D TV system, the scene is recorded by two or more cameras and then it is compressed, transmitted and displayed on a 3D display. Another scenario is the free viewpoint television, in which a camera array is employed and a (3-D) scene is synthesized at any 3-dimensional virtual viewpoint. Thus, the user has the freedom to select his/her preferred viewing position. However, the multi-camera techniques not only increase the demand for bandwidth but also increase greatly the complexity of encoding.

To achieve a good coding efficiency for a wide range of applications, including free view point television and 3D television, the video coding standard H.264/MPEG-4 Part 10 was defined by the JVT (Joint Video Team, jointly by the ITU-T and ISO/IEC). The Multi-view Video Coding (MVC) technique inherits the coding tools from AVC and furthermore combines temporal/inter-view predictions, where images are predicted from temporally neighboring images and spatially related images in adjacent views.

In recent years, there has been a significant progress in the development of graphics processing units (GPUs). The NVIDIA GPUs are capable of performing non-graphics computations using the NVIDIA's C-like CUDA programming interface. Several articles have published on using CUDA for H.264 encoding [3][4][5]. All of these approaches targeted at parallelizing the full search block matching algorithm (FSBMA). For example, in [3], using CUDA to implement the FSBMA for motion estimation, Chen and Hang were able to achieve about 12 times faster than using the PC CPU only. However, few research projects try to parallelize a fast search algorithm on CUDA due to the limitation of the GPU architecture. Often, a fast algorithm eliminates ineffective calculations by checking certain termination criterions, which produces conational branching instructions. Unfortunately, the early termination criteria do not match the GPU architecture because the SIMD processors have to synchronize all concurrently executing instructions. Also, branching instructions and memory access latency, for example, often introduce hardware idling. To achieve the full utilization of all processors in a GPU, we need to modify the conventional algorithms to reduce the hardware idling time. Our contribution in this study is to design a highly parallel fast search algorithm, so-called multithreaded one-dimensional search (MODS) that can take the advantages of the CUDA SIMD structure.

2. COMPUTER UNIFIED DEVICE ARCHITECTURE

2.1. Hardware Architecture of GT200

As a computation device, a single GT200 is composed of 30 Streaming Multiprocessors (SM), each of which consists of

8 32-bit Scalar Processor (SP) cores for single-precision floating-point mathematical functions, 2 special function units (SFU) for a bevy of unusual functions (such as sin, cosine, etc) and a 64-bit SP core for double-precision floating-point mathematical functions.

2.2. Programming Model

In programming CUDA, the GPU code is compiled into the instructions, called "kernel". Invoked by the CPU, the kernel is loaded to the GPU that acts as a coprocessor to the CPU. It is executed by the mechanism of "thread blocks", which are composed of "threads". Generally, the kernel function is executed by thread blocks in parallel. In runtime, a thread block is assigned to an SM by hardware scheduler. All threads within this thread block are scheduled to SPs or SFUs within the assigned SM. When a thread block is finished in an SM, the scheduler quickly allocates the next thread block that needs to run on that SM. In this way, the GPU can be more productive as long as there is enough parallelism to keep them busy.

3. MVC ENCODER ACCELERATION BY CUDA

3.1. Multithreaded One-Dimensional Search

The main contribution of this paper is to implement a fast motion search algorithm on CUDA. The known previous CUDA implementations all use the exhaustive (full) search to maximize parallelism. We modify an existing fast algorithm and modify it so that it can fit into the CUDA SIMD structure. We call this scheme "multithreaded one-dimensional search (MODS)". It is based on the parallel hierarchical one-dimensional search (PHODS) proposed by Liang-Gee Chen et al [6]. PHODS is designed to reduce the number of sequence steps and search points. Using the principle of PHODS, we design the MODS that has a regular control flow and fixed instructions for each iterative process. Hence, each iterative process can be independently executed by a different thread in parallel so that it matches the SIMD model. The search procedure of MODS is described by using the psudocode as follows:

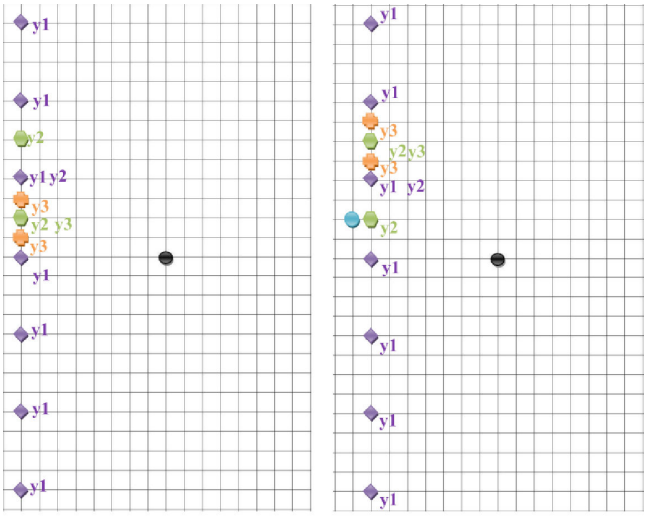
```
x = -search range;   // macroblock location  on the x-axis

For-loop ( ; x < search range; x++) {          // in parallel
      y = 0;   // macroblock location on the y-axis
      y = MinSad ( y, y±4, y±8, y±12);         // the first step
      y = MinSad ( y, y±2);                    // the second step
      candidate[x] = MinSad ( y, y±1);         // the third step
}

MV = MinSad(    candidate[±search range] );   // in parallel
```

Fig. 1 is used to illustrate the operation of MODS. We assume that the search range is 8. The first step is to calculate all y1 locations. Next, the one with the least sum of absolute difference (SAD) is as the searching center for the second step. The second step is to compute all y2 locations. Same as the first step, the one with the least SAD becomes the searching center for the third step. In the third step, we can choose the least SAD position among all y3 locations. After the search along y axis is done for a fixed x location, the best y is identified along the vertical line for each x location. The last search is performed on all the (x, y) candidates obtained from the previous step. At the end, the best overall candidate with the least SAD is the chosen motion vector (MV).

To implement the MODS scheme on CUDA, a thread block composed of many threads processes a motion search of a macroblock. The MODS is executed N times in parallel by N different threads within a thread block. In general, the value N is decided by the search range. The mapping of the search steps to threads is shown by Fig. 2.



(a) (b)

Fig. 1. (a) Search for the best y along the vertical line with the x = -8. After three steps, the point labeled by y3 with the least SAD is the best point. (b) Search for the best y along the vertical line with the x = -7. After three steps, the point labeled by y3 with the least SAD is the best point.

3.2. Maximizing Parallel Execution

In JMVC (Joint Multi-view Video Coding) software [7], macroblocks of each frame are encoded in a raster scan order. Thus, it processes a motion search of a macroblock sequentially. If we implement MODS using CUDA with the original JMVC processing flow, it is unable to achieve the full utilization of SMs. Since a thread block processes a motion search of a macroblock, it only actives an SM to execute a thread block at the same time.

In order to maximize the number of concurrently active SMs, we modify the original processing flow, as depicted in Fig. 3. We design a MV search module using CUDA to

obtain the MVs of all macroblocks within a frame. Then, the rest of operations (including RD cost calculation, mode selection, etc.) are still done in the original flow. Due to this modification, all macroblocks within a frame are available simultaneously for scheduling on running on to free SMs. The parallel macroblock processing structure is illustrated by Fig. 4.

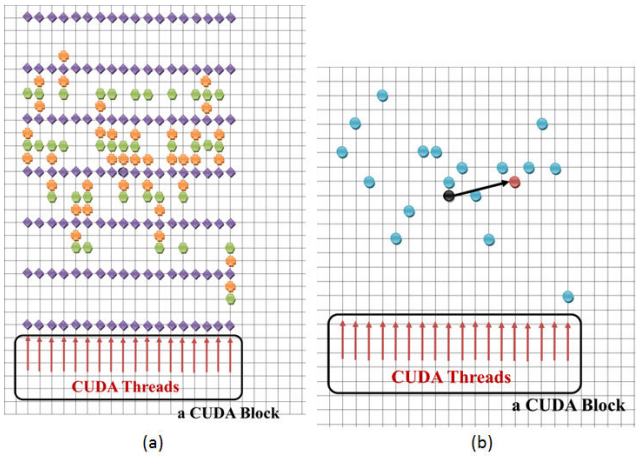


Fig. 2. (a) All vertical lines are searched in parallel in a thread block. (b) The results of (a) are compared in a thread block for finding the final motion vector.

3.3. Encoder Implementation

In JMVC, a range of block sizes (from 16x16 down to 4x4) is supported. Depending on block sizes, we implement seven corresponding kernel functions, which calculate the MVs in the MV search module. In each kernel function, every thread block processes a motion search of a macroblock in parallel. The total number of thread blocks in each kernel function is:

$$threadblock\_num = \frac{frame\_width}{macroblock\_width} \times \frac{frame\_height}{macroblock\_height} = macroblock\_num$$

For example, if the frame_width is 640 and frame_height is 480, the number of thread blocks in the 16x16 kernel function is:

$$threadblock\_num = 1200 = \frac{640}{16} \times \frac{480}{16} = 16 \times 16\_macroblcok\_num$$

The number of 16x16 macroblocks in a frame is 1200 and they are processed by 1200 thread blocks in parallel. Every thread block uses the MODS algorithm to concurrently search for a macroblock MV. After the search process finishes, a 16x16 kernel function produces 1200 MVs (of 1200 macroblocks) within a frame.

The dimension of a thread block depends on the search range. Adopting a fairly large search range to include more MV candidates, we set 128 threads (T0 ~ T127) in a thread block to handle ±64 one-dimensional search. After all searches are done, 128 threads produce 128 candidates and put them into the on-chip shared memory for the next *candidate reduction process*. In this reduction process adapted from [8], each thread compares a pair of candidates. The smaller one is stored back to the shared memory so that it keeps only half of candidates for the next iteration. This process is repeated until the final winner (best MV) is obtained.

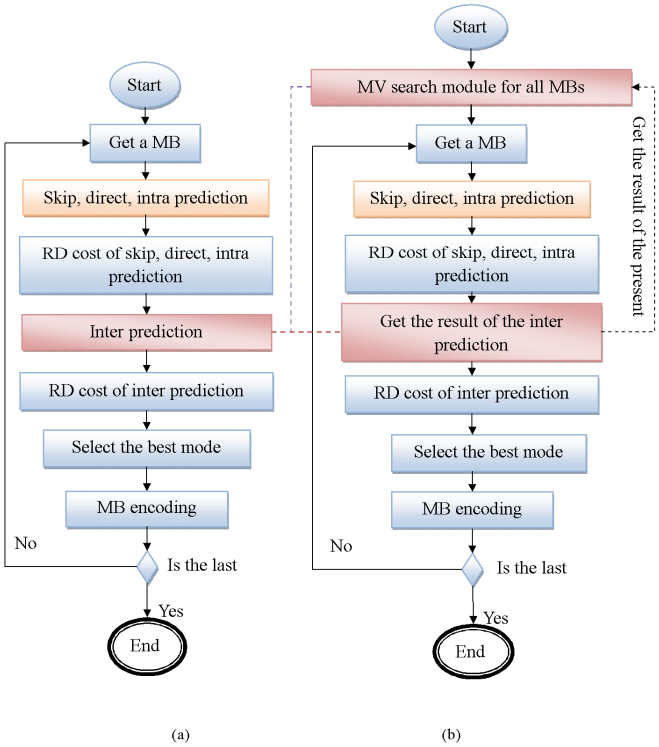


Fig. 3. (a) shows the original flow (b) shows the modified flow. The original inter prediction in (a) is divided into two procedures in (b).

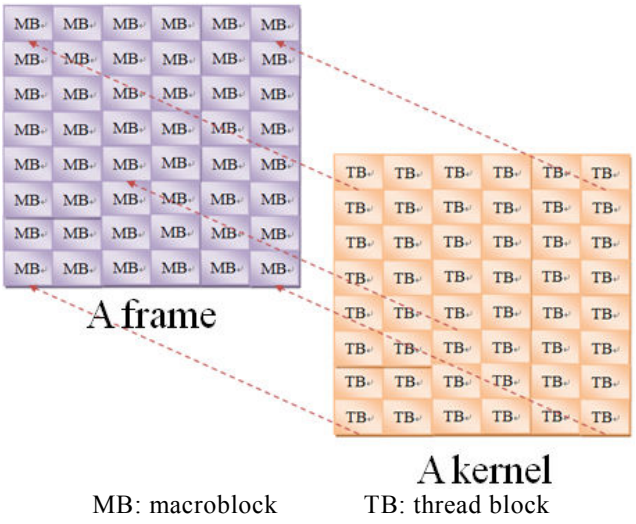


Fig. 4. All motion searches of all macroblocks within a frame are concurrently executed by thread blocks in a kernel. Since the number of thread blocks in a kernel equals to the number of the macroblocks in a frame, this arrangement is schedulable for a free SM.

4. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed algorithm on CUDA, the following development environment and parameters are used: (1) Intel Core 2 Quad 2.4 GHz Q6600 , (2) NVIDIA GeForce GTX-280, (3) Ubuntu Linux 9.04, (4) g++, (5) CUDA Toolkit and Driver 2.2, (6) JMVC search mode: Fast Search (EPZS) with search range 32, (7) CUDA search mode: MODS with search range 64, (8) Motion search with RDO: off, (9) GOP size: 8, (10) Frame to be encoded: 65, (11) Basic QP: 24,28,32,36.

The *ballroom* (640x480), and the *exit* (640x480) both are 8-view MPEG test sequences. In Table 1, the point to observe is that the PSNR drops are below 0.007dB in average cases. However, we save 1.013% on the average bit rate. Table 2 describes the speedup ratio of MODS on CUDA and it also shows the performance comparison between EPZS [9] and MODS on CUDA. Our scheme is about 88 times faster than the MODS on PC only in all cases. Next, the performance comparison between the original fast search of JMVC and MODS on CUDA is highly dependent on whether the DE is applied to them. Generally, the speedup ratios are about 18~20x when both DE and ME are in use.

The overall encoding with MODS on CUDA can get approximate 4 times faster than the PC-only version. Compared with the original JMVC encoder, the proposed encoder has the 31%~49% time saving. Since only the integer-pel ME and DE parts are executed on CUDA. The rest of encoding task is done by PC, and the speedup of the overall encoding time is not included.

In [5], the proposed ME speedup is about 1.9 times as compared to EPZS. Our proposed ME provides a 9~17 times speedup as compared to EPZS. Although different platforms contribute part of our additional gain, the highly parallel studture of our scheme significantly helps in improving the overall perforamance.

5. CONCLUSION

This paper presents a parallel fast search algorithm that can be effectively implemented on NVIDIA CUDA. This scheme, so-called Multithreaded One-Dimensional Search (MODS), can be used for both motion estimation (ME) and disparity estimation (DE) in the MPEG MVC encoder. It is proposed to overcome the software design challenge of a multi-core processor. We demonstrate that the GPU acting as a coprocessor can effectively accelerate the massive data computation. Experimental results show that with GPU, the ME and DE processes with MODS can be 89 times faster than its counterpart on the PC entirely. When the fast search algorithm, EPZS, of the JMVC coder on PC is turned on, the MODS version on CUDA can still be 20 times faster. Compared with the original JMVC encoder, the proposed encoder has a minor coding quality loss of only 0.001–0.026 dB in PSNR.

Table. 1. Rate-Distortion comparison between proposed encoder and original encoder

Sequence	DPSNR(db)	DBR(%)
Exit	-0.006 ~ -0.026	-0.809 ~ -1.353
Ballroom	-0.001 ~ 0.002	-0.619 ~ -1.319

Table. 2. Performance comparison of the motion search

Sequence	Speedup of MODS using CUDA	Proposed (MODS) vs Original (EPZS)
Exit	86~89x	9~18x
Ballroom	87~89x	12~20x

6. ACKNOWLEDGMENT

This work was supported in part by the NSC, Taiwan under Grants 98-2221-E-009 -076 and 98-2219-E-009 -015.

7. REFERENCES

[1] A. Smolic and P. Kauff, “Interactive 3-D video representation and coding technologies,” in *Proc. IEEE*, vol. 93, no. 1, pp. 98–110, Jan. 2005.

[2] T. Fuji and M. Tanimoto, “Free-Viewpoint TV Systems Based on Ray-Space Representation,” in *Proc. of SPIE*, vol. 4864, pp. 175-189, Nov. 2002.

[3] W. -N. Chen, H. -M. Hang, “H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA)”, *IEEE International Conference on Multimedia and Exposition*, Hannover, pp. 697-700, 2008.

[4] B. Pietersa, C. F. Hollemeersch, P. Lambert, and Rik Van de Walle, “Motion Estimation for H.264/AVC on Multiple GPUs Using NVIDIA CUDA”, *SPIE*, San Diego, vol. 7443, 2009.

[5] Y.-L. Huang, Y.-C. Shen and J.-L. Wu, “Scalable computation for spatially scalable video coding using NVIDIA CUDA and multi-core CPU”, *Proceedings of the 7th ACM international conference on Multimedia*, Beijing, pp. 61-370, 2009.

[6] L.-G. Chen, W.-T. Chen, Y.-S. Jehng, and C.-T. Church, “An efficient parallel motion estimation algorithm for digital image processing,” *IEEE Trans. Circuits and Systems for Video Tech*, vol. 1, pp. 378–385, Dec. 1991.

[7] Y. Chen, P. Pandit, and S. Yea, “WD 4 reference software for MVC,” ISO/IEC JTC/ISC29/WG11 and ITU-T Q6/SG16, Doc. JVT-AD207, Jan. 2009 (JMVC).

[8] Mark Harris, “Optimizing Parallel Reduction in CUDA,” NVIDIA Developer Technology, 2007.

[9] Tourapis, H.-Y. Cheong; Tourapis, A.Michael, “Fast motion estimation within the H.264 codec”, *IEEE International Conference on Multimedia and Exposition*, July 2003.