Parallel Implementation Algorithm of Motion Estimation for GPU Applications

by

Tian Song[1,2*], Masashi Koshino[2], Yuya Matsunohana[2] and Takashi Shimamoto[1,2]

Abstract

The video coding standard H.264/AVC can achieve higher coding efficiency than previous standards. However, it comes at the expense of an increased encoding complexity, especially for motion estimation process which induces very time consuming task even for current central processing units (CPU). On the other hand, due to the rapid growth of the processing capability of graphics processing unit (GPU), using GPU as a coprocessor to assist the CPU in computing massive data becomes essential. In this work, we propose a fast parallel algorithm for motion estimation (ME) process in H.264/AVC on a computer unified device architecture (CUDA) platform. The proposed algorithm performs the parallel calculation of the residuals and SAD. Simulation results show that with the assistance of GPU the processing time is about 2 times faster than that of using CPU only.

Key words: INTER, INTRA, H.264/AVC, GPU

1. Introduction

H.264/AVC is an outstanding video coding standard that achieves the higher coding efficiency than conventional video coding technology. Many excellent coding algorithms are introduced in H.264/AVC which makes the process of the encoding become very complicated to achieve high coding efficiency. That is the reason why it increases the implementation cost of H.264/AVC not only for software but also for hardware implementation. It is considered very difficult to realize the real time encoding of H.264/AVC by only using software implementation especially for high resolution applications. However, using dedicated hardware implementation can significantly improve the coding speed but the implementation cost will be increased.

On the other hands, with the rapid development of the multi-core devices especially the development of graphics processing unit (GPU) in recent years, it is considered an reasonable tool to realize not only 3D rendering but also parallel processing of many applications, such as linear algebra or the other scientific calculation. Some excellent typical GPU architectures are provided by NVIDIA and an appropriate computer unified device architecture (CUDA) platform is also provided [1]. CUDA is a programmer's workbench complete set for GPU of NVIDIA, including a programming model, a programming language, the compiler and a library [2]. CUDA brings big improvement in flexibility and the programmability of GPU.

GPU is used in increasing fields due to its highly parallel processing ability [3][4]. Some previous works concerning the fast implementation of the encoder of H.264/AVC are proposed [5]-[8]. A previous work is dedicated to the improvement of the intra coding process on GPU [5]. Another

1 Computer Systems Engineering, Institute of Technology and Science, Graduate School of Engineering, Tokushima University, The University of Tokushima
2 Graduate School of Advanced Technology and Science, The University of Tokushima
* The University of Tokushima, 2-1 Minamijyosanjima, Tokushima city, 770-0814, Japan

implementation using GPU architecture is implemented only from the viewpoint of frame level parallel encoding [6]. However, it is obvious that the motion estimation process which accounts for the highest complexity has the highest priority to be parallel executed. Related work dedicated to motion estimation is also proposed for the motion estimation of H.264/AVC by rearranging the encoding order of 4x4 blocks [7]. However, this approach does not realize pixel level parallel computation. In another previous work, a fast GPU based motion estimation algorithm is proposed in which a small diamond search is adapted to the programming model of a GPU to exploit their available parallel computing power and memory bandwidth [8]. However, this work also does not concern the pixel level parallel processing. One more important issue is the fact that the fast improvement of GPU architecture and technology always encourages the development of updated fast algorithm.

In this work, we propose a fast implementation of motion estimation of H.264/AVC in the pixel level on GPU.

2. GPU Architecture

Many excellent GPU architectures are available to support the video coding algorithm of H.264/AVC. In this work, a typical NVIDIA GPU and CUDA architecture are used as a platform to realize the proposed algorithm.

1. Hardware Model

The architecture of GPUs are different depending on the generation of the cores or the model number but the basic architecture is compatible. There is a number of streaming multiprocessor (SM) in the GPU tip. Furthermore, each SM contains 8 operation processing units (streaming processor, SP). In GPU, Single Instruction Multiple Data (SIMD) type command set carrying out the same order for the 8 SPs in the SM.

2. Programming Model

A programming kernel model of GPU is shown in Figure 1. Kernel configuration is a multidimensional structure. Thread is a fundamental element of the parallel computation. In a parallel programming, main task is divided into the subtasks and we refer to them as the threads. Threads are grouped into the block to share intercommunication and memory resources. Block can be one, two or three dimensional structure. Block is possible to be grouped into a grid. The grid is a one or two dimensional structure. Because GPU is a single instruction multiple data (SIMD) hardware architecture, each thread in the grid will compute the same kernel function on different part of dataset. The thread count which will operate simultaneously is limited by the hardware architecture. The number of a thread handled at a time is limited and 32 threads are managed by a unit is called warp. In fact, the handling of 32 threads is carried out by carrying out 8 threads in 4 cycles because there are only 8 SPs in a SM.

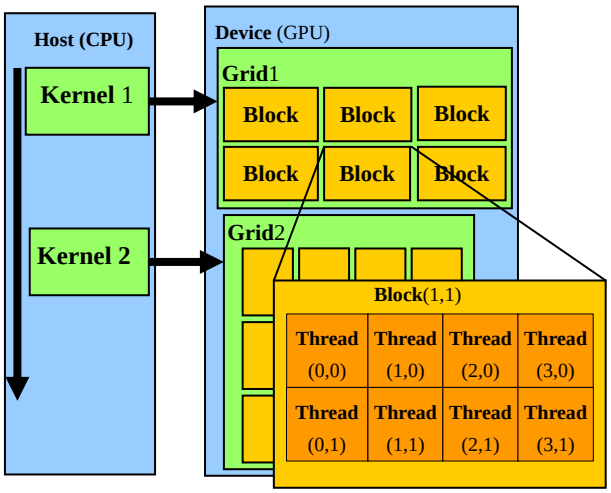

Figure 1: Programming Model

3. Memory Model

With the CUDA, six kinds of memories are usable (register memory, local memory, shared memory, global memory, texture memory, and constant memory). A memory model of GPU is shown in Figure 2.

a. Register Memory

Because the register memory is implemented on a tip of GPU, it can realize fast access. The value of a

variable used by a kernel function is typically stored on a register. When there are too many used registers, the kernel function can not be carried out.

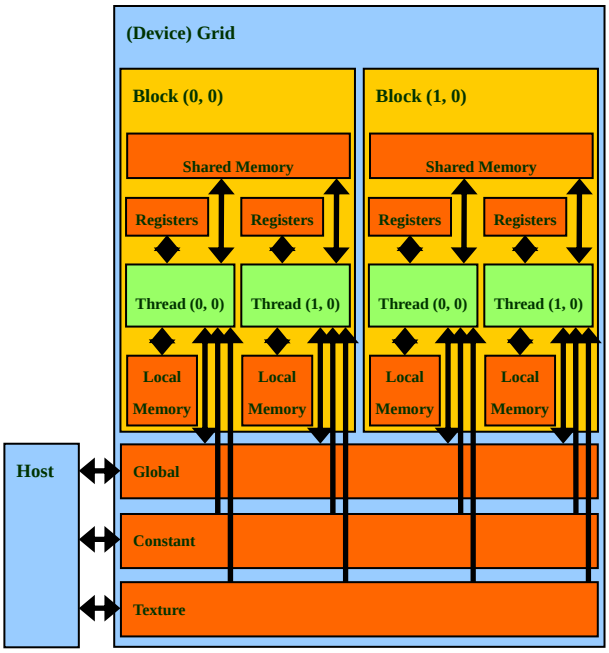

Figure 2: Memory model

b. Local Memory

Because the local memory is implemented outside a tip of GPU, access speed is 100 times slower when it compared with that using a register. When registers are not enough, local memory is used as an evacuation place of register data.

c. Shared Memory

Shared memory is memory characterizing the architecture of GPU of NVIDIA and is provided with 16,384byte in each SM. High-speed access is possible on equal terms with a register so that there is on a tip. The thread in the block is carried out in the same SM and shared memory can read and write from all threads in the block. In addition, it is used to store the value of the parameter of the kernel function.

d. Global Memory

Because the global memory is implemented outside a tip of GPU, access speed is 100 times slower when it compared with that using a register or shared memory. However, the capacity is very big. In addition, the global memory can read and write from all thread in all Block.

e. Constant Memory and Texture Memory

In the constant memory and texture memory, cache is equipped in each SM. Data are temporarily saved in the memory when memory access occurred. Therefore, data can be fast hit when it is on the memory.

3. Proposed Architecture

In ME, it is possible to find a high precision motion vector if a search range is wide enough. However, the bigger the search range is the much processing time is necessary. In this paper, the SAD for each macroblock is calculated by GPU in parallel to reduce significant calculation time in the search range.

At first, the pixels of current block and the reference frame have to be transferred from the CPU side to the memory of GPU. Because it will induce significant overhead of data transmission, an efficient data transmission approach is necessary. Next, the SAD for each search point in the search range is parallel calculated by SMs in GPU.

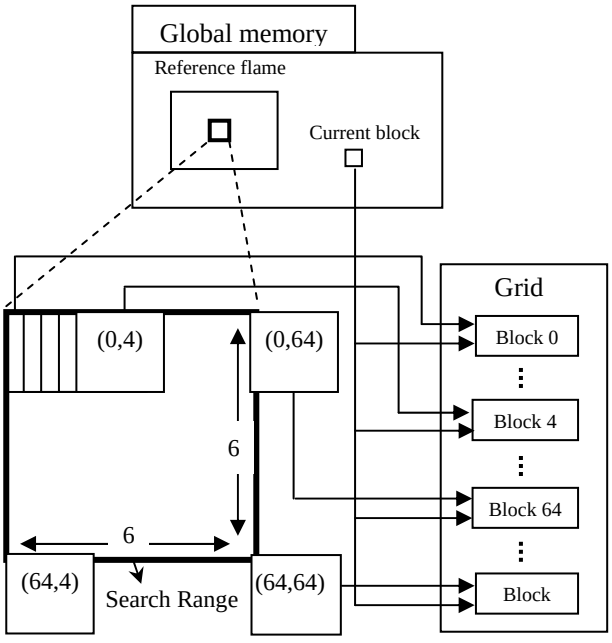

Figure 3: The proposed architecture

The motion estimation process for one macroblock (MB) can be parallel calculated at each search point. In addition, the calculated SADs are compared in parallel and the smallest SAD value is calculated by threads. This calculation is performed for each MB. This number of search point is equivalent to the consumption of SMs in GPU. The processing order and the distribution of the SMs are shown in Figure 3.

As shown in Figure 3, the calculation of the SAD is performed in parallel by SMs. These blocks are equivalent to SMs in GPU. The number of the blocks have to be set to GPU is the number of the search point in the search range. In each search point the SAD of each search point is calculated. The detail calculation of the 8x8 mode is described in Figure 4.

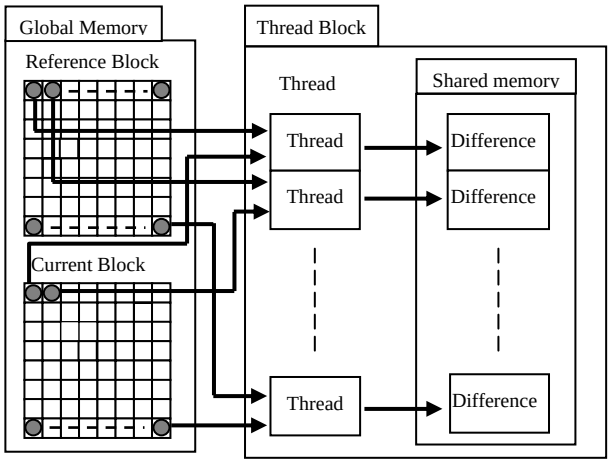

Figure 4: Difference calculation

As shown in Figure 4, one thread reads two pixel values at the same position of the current block and the reference block. Then, the difference of two pixels value is calculated and the absolute value is written back to the shared memory. Because these differences will be read out again to calculate the sum total of the SAD, the shared memory can realize fast memory access than other internal memory. These threads are equivalent to SPs in GPU.

The number of the threads have to be set is the number of the pixels of the current MB mode. This calculation of difference can be realized in parallel. Finally, these calculated differences are accumulated to calculate the SAD. This process is described in Figure 5.

As Figure 5 shows, firstly one thread reads and accumulates two differences in the shared memory. After that the accumulated sum is wrote back to the shared memory. If all calculations of the accumulation finished, the number of sums becomes half and the accumulation is repeated until the sum total is calculated. The value which is finally left is the SAD. The calculated SAD is written back to global memory at last. When the calculation for all search points finished, all the SADs are saved on global memory. Finally it is necessary to choose a minimum from these SADs. This process is described in Figure 6.

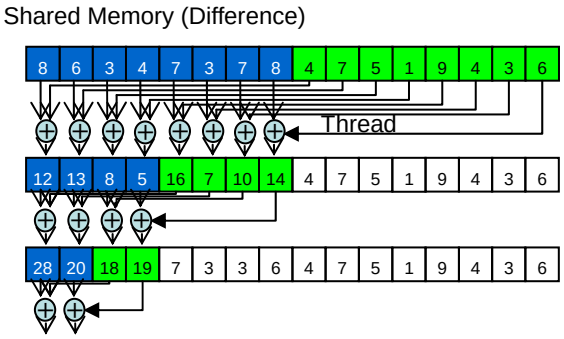

Figure 5: The calculation of the sum total

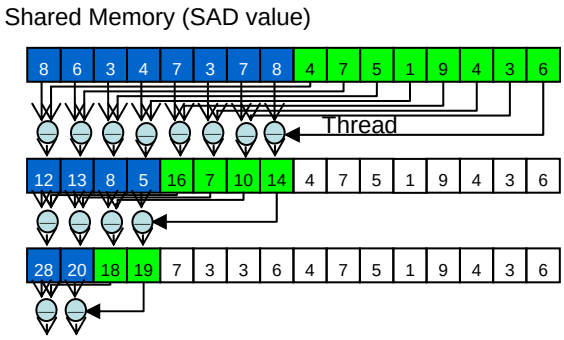

Figure 6: SAD comparison

As shown in Figure 6, this comparison is performed with the same algorithm as calculating the sum total. However, the comparison uses subtraction process instead of accumulation process. These comparisons are also performed in GPU in parallel. First, one thread compares two SADs and saves the smaller one

in the shared memory. If all comparisons finished the number of SAD becomes half and the comparison is repeated until the remainder becomes 1. Finally, the selected smallest SAD is transferred to the memory of CPU side.

4. Simulation Results

Simulation is performed at the following conditions: (1) AMD Athlon 64 X2 Dual Core 2.6GHz (2) 2048MB memory, (3) NVIDIA GeForce GTX 285 with 1024MB memory, (4) Microsoft Windows XP 32bit sp3. The proposed algorithm is implemented using reference software JM14.2. Sequence sizes are QCIF and CIF, respectively. Frame number is 50 and search range is set to 32. Table 1 shows the simulation results.

Table 1: Comparison of encoding time

Sequences	Enc. Time CPU (sec)	Enc. Time GPU (sec)
Carphone (QCIF)	140.958	81.122
Container (QCIF)	135.264	81.266
Foreman (QCIF)	171.174	82.666
Coastguard (CIF)	901.732	335.805
Container (CIF)	558.657	325.485
Flower (CIF)	697.650	337.231
Football (CIF)	735.355	321.310
Foreman (CIF)	587.286	325.422
Mobile (CIF)	845.419	342.799

As Table 1 shows, the proposed algorithm succeeds in reducing run time of 40～60% compared to the single CPU implementation. These results show the possibility to achieve fast implementation in the pixel level. However, as also mentioned in previous works the overhead for loading the reference data restricts the performance of the GPU.

5. Conclusion

In this paper we proposed a fast parallel algorithm for ME of H.264/AVC for GPU implementation. Simulation results show that our proposal succeeds in reducing run time of 40～60%. Combining the proposed algorithm to some previous work which realizes the parallel processing in macroblock level and frame level, faster implementation is expectable. We can not ignore the overhead of reference data loading from memory. However, this is unavoidable as far as the implementation is on the scope of H.264/AVC because of the correlation between the adjacent MBs.

References

1. NVIDIA, NVIDIA CUDA Compute Unified Device Architecture Programming Guide Version 2.3, 2009.

2. W. N. Chen, and H.M. Hang, "H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA)," IEEE International Conference on Multimedia, pp.697-700, june 2008

3. N. Seiller, N. Singhal, and I. K. Park, "Object oriented framework for real-time image processing on GPU," IEEE 17th International Conference on Image Processing, pp.4477-4480, Sep. 2010

4. Y. Sun, X. Sun and H. Zhang, "Research on parallel cone-beam CT image reconstruction on CUDA-Enabled GPU," IEEE 17th International Conference on Image Processing, pp.4501-4504, Sep. 2010

5. N. M. Cheung, O. C. Au, M. C. Kung, P.H.W. Wong, and C. H. Liu, "Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors," IEEE Transactions on Circuits and Systems for Video Technology, pp.1692-1703, nov. 2009

6. C. Y. Lee, Y. C. Lin, C. L. Wu, C. H. Chang, Y. M. Tsao, and S. Y. Chien, "Multi-Pass and Frame Parallel Algorithm of Motion Estimation in H.264/AVC for Generic GPU," IEEE Int'l Conf. on Multimedia and Expo, pp. 1603-1606, July 2007

7. C. W. Ho, et al.,Motion Estimation for

H.264/AVC Using Programmable Graphics Hardware, IEEE Int'l Conf. on Multimedia and Expo, pp. 2049-2052, July 2006

8. M. C. Kung, O. Au, P. Wong and C.H. Liu, "Block based parallel motion estimation using programmable graphics hardware," International Conference on Audio, Language and Image Processing, 2008

9. M. Schwalb, R. Ewerth, and B. Freisleben, "Fast Motion Estimation on Graphics Hardware for H.264 Video Encoding," IEEE Transactions on Multimedia, pp.1-10, Jan. 2009