GPU Acceleration of H.264 / MPEG-4 AVC Software Video Encoder

Tatsuji Moriyoshi, Fumiyo Takano and Yuichi Nakamura
System IP Core Research Laboratories, NEC Corporation, Japan
E-mail: moriyosi@ce.jp.nec.com, f-takano@ce.jp.nec.com, yuichi@az.jp.nec.com

***Abstract*— H.264 / MPEG-4 AVC encoding of high-resolution video is still a heavy task for today's high-end PCs. To accelerate the H.264 software encoder, we utilize many-core GPU (Graphics Processing Unit). Because spatial and temporal data dependencies between macroblocks (MBs) limit the MB-level parallel processing, we propose a relaxation method of inter-MB dependencies and a frame-pipelining to obtain higher parallelism. Experimental results show that our GPU accelerated encoder runs more than 10 times faster than the CPU implementation while keeping the bit rate increase acceptably small. This encoder achieves 60 frames per second encoding of full-HD resolution videos.**

I. INTRODUCTION

H.264 / MPEG-4 AVC [1], which is the latest international video coding standard that can provide higher coding efficiency than earlier standards, is popularly adopted to various applications such as broadcasting, video streaming, video cameras and recorders, and portable media players. Since PCs are widely used for viewing, editing or creating video contents, software H.264 codec for PCs is needed. Thus, we have been developing an H.264 software encoder.

Since the complexity of H264 is significantly higher than the previous standards, it is necessary to speed-up encoding process to develop a practical H.264 encoder. To improve encoding speed, we have developed low complexity algorithms for H.264 encoding, such as fast motion estimation and fast intra/inter decision. We have also parallelized our H.264 encoder to effectively utilize multi-core processors [2]. However, real-time encoding of high-resolution video, such as full-HD (High Definition), is still a heavy task even for today's high-end multi-core processors.

In recent years, the performance improvement of Graphics Processing Unit (GPU) is remarkable and GPUs are becoming attractive as accelerators for heavy tasks. Modern GPUs achieves much higher computational performance than that of CPUs through their many processing cores and wide memory bandwidth, and are used not only for 3D graphics rendering but also for general data-parallel processing. This technique is called General-purpose computing on GPU (GPGPU).

In this paper, we present efficient implementation of the H.264 software encoder with GPU acceleration. To fully utilize many processor cores on GPU, we propose a new method to relax the constraints on concurrent processing caused by the data dependencies in H.264 encoding.

II. GPGPU AND CUDA

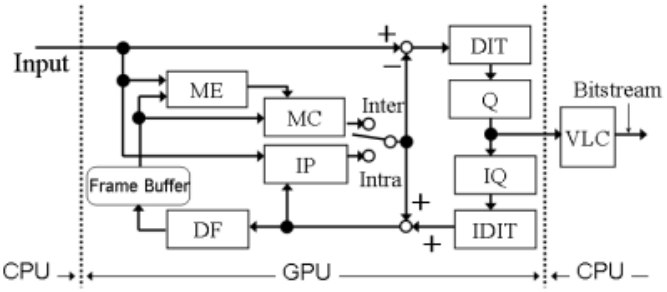

Fig. 1 Block diagram of GPU H.264 encoder

CUDA (Compute Unified Device Architecture) [3] is a GPGPU architecture developed by NVIDIA. CUDA provides single program multiple data (SPMD) model where many threads simultaneously execute single program with different input and output data.

CUDA GPU architecture is a hierarchical structure. A processing element is called "CUDA core" and multiple CUDA cores are grouped into a "Streaming Multiprocessor (SM)", and single GPU chip consists of multiple SMs. For example, NVIDIA GeForce GTX 580 GPU consists of 16 SMs and each SM has 32 CUDA cores, so the total number of CUDA cores is 512. The programming model of CUDA is also hierarchical. One "thread" runs on a CUDA core and multiple threads are grouped into "thread block". All threads in a thread block run on single SM. On a GPU chip, one "grid" that consists of multiple thread blocks is executed.

CUDA architecture defines multiple types of memory. A "shared memory" is a high speed on-chip memory associated with each SM. The shared memory can be shared by all threads in a thread block but can't be accessed from other thread blocks. A "global memory" is an off-chip memory that is shared by all threads. Generally, the shared memory is small capacity memory with high speed access, and the global memory is large capacity memory with relatively slow speed access.

III. GPU ACCELERATION OF H.264 ENCODER

A. Implementation of GPU H.264 encoder

Fig. 1 shows the block diagram of our H.264 software encoder with GPU acceleration. Most of function blocks, Motion Estimation (ME), Motion Compensation (MC), Intra Prediction (IP), Discrete Integer Transform (DIT), Quantization (Q), Inverse Quantization (IQ), Inverse Discrete

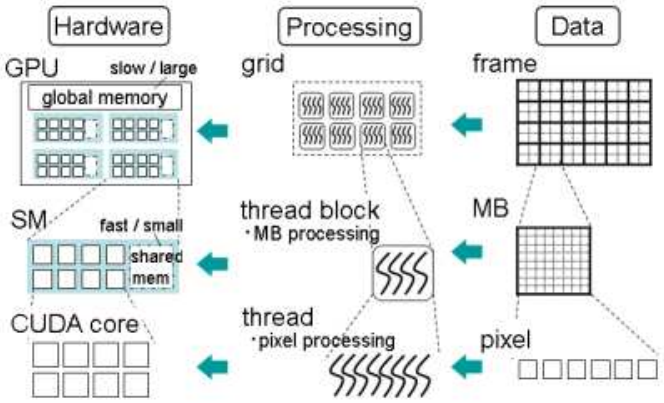

Fig. 2 Hierarchical parallel processing

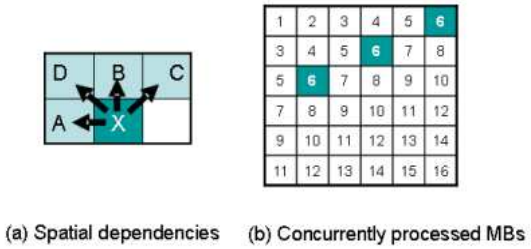

Fig. 3 Spatial MB dependencies

Integer Transform (IDIT) and Deblocking Filter (DF) are offloaded from CPU to GPU because these image processing functions have pixel level data parallelism and are suited to be accelerated by parallel processing on many core GPU. On the other hand, Variable Length Coding (VLC) is processed by CPU as usual, because VLC essentially requires sequential processing. The basic coding unit of H.264 is 16x16 pixel macroblock (MB). Each input video frame is divided into MBs and the encoding process is performed on each MB. Independent MBs can be encoded in parallel. In addition, there is a pixel-level parallelism within each MB. To effectively accelerate the encoding process on CUDA architecture, we adopt two-level hierarchical parallel processing as shown in Fig. 2. The video frame data is stored into the large global memory. The encoding task of each MB is assigned to CUDA thread block. Within the thread block, many threads perform independent pixel processing in parallel. The MB encoding process is optimized to exploit the high speed shared memory for fast reading/writing and reusing of intermediate MB data.

B. Data dependencies in H.264 encoding

If there is no data dependency between MBs or between pixels, H.264 encoding process is easily accelerated with GPU parallel processing as shown in Fig. 2. However, because H.264 utilizes the spatial and temporal correlations inherent in video signal to achieve high coding efficiency, there are some kinds of data dependencies. Especially, data dependencies between MBs severely limit the scalable parallel processing.

As illustrated in Fig. 3 (a), the current coding MB (X) has dependencies on its adjacent left (A), upper-left (D) , upper (B), and upper-right (C) MBs [4][5]. These spatial dependencies limit the number of concurrent processing MBs as Fig. 3 (b). In Fig. 3 (b), each box represents MB and the number in each box is processing order. MBs with same processing order are processed concurrently. Because of the dependencies as Fig. 3 (a), only one MB per every two MB column can be processed. As a result, the maximum number of concurrent processing MBs is limited to the half of the horizontal number of MBs in a frame (e.g. 40 for 720p and 60 for 1080p/i). This is not sufficiently large for fully utilizing many processing cores of today's GPUs. Therefore, relaxation methods of dependencies between MBs to obtain higher parallelism are needed.

The data dependencies in Fig. 3 (a) can be classified into three types [4][5][6].

- Motion estimation

  In H.264, a motion vector is coded as the difference between the optimal motion vector and predicted motion vector (PMV), because neighboring motion vectors have high correlations. Since coded bitstream contains both pixel difference information and motion vector difference [1], practical encoders determine the optimal motion vector so that the joint cost of pixel difference and motion vector difference is minimized [7][8]. This motion estimation process needs PMV, which is derived from motion vectors of MB A, B and C.
- Intra prediction

  Pixels in current MB (X) may be predicted from pixels in MB A, D, B or C.
- Deblocking filter

  Filtering of current MB needs pixels in MB A and B. Since pixels in MB B may be modified by the filtering process of MB C, current MB depends also on MB C [6].

Recently, various motion estimation methods utilizing GPUs has been proposed [9][10][11][12]. In references [9][10], the motion estimation for all MBs in a frame are processed concurrently. This method is effective to achieve high processing speed, but the coding efficiency tends to degrade because PMV information can not be used in motion estimation. We have evaluated the importance of the PMV in motion estimation process. We modified the H.264 reference encoder JM16.0[8] not to use PMV and evaluated BDBR (Bjontegaard Delta BitRate)[13] over original JM16.0. Eight full-HD sequences from ITE standard test sequence [14] are used. Fig. 4 shows the BDBR evaluation results. The bit rate increase caused by disabling PMV in ME is not small, 9% on average and 44% in worst case.

The motion estimation in [11][12] utilizes PMV. However, with [11], the number of concurrent processing MBs is limited as in Fig. 3 (b). On the other hand, [12] proposes to relax the constraints on concurrent processing by removing dependency to left MB (MB A) as shown in Fig. 5 (a). Since the information of MB A is not available, PMV is derived from MB B, C and D. With this relaxation method, all MBs in a MB row are processed concurrently as Fig. 5 (b) and the number of concurrent processing MBs is doubled. However, removing dependency to adjacent left MB may cause the

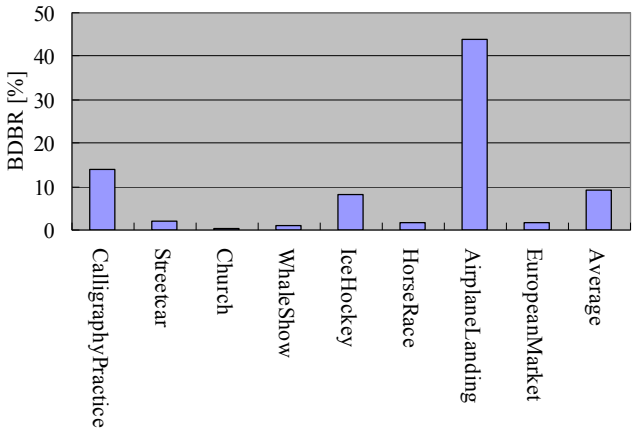

Fig. 4 BDBR evaluation of ME without PMV

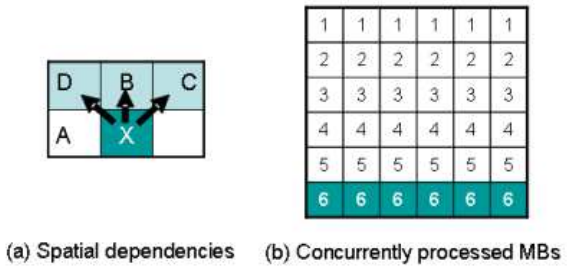

Fig. 5 Modified spatial MB dependencies ([12])

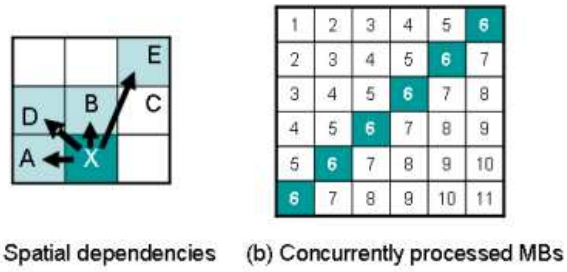

Fig. 6 Modified spatial MB dependencies (proposed)

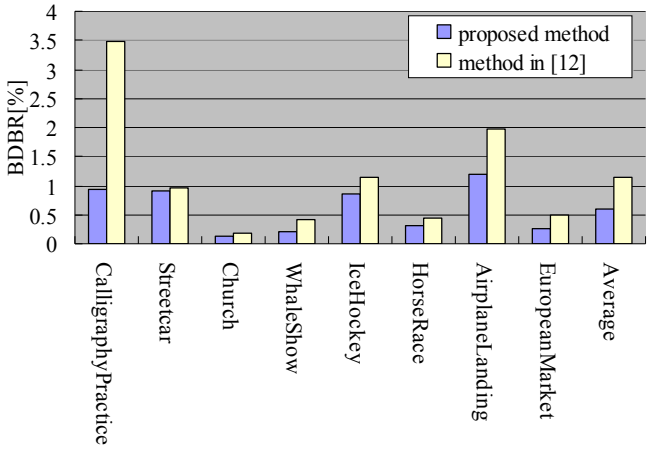

Fig. 7 BDBR evaluation of dependency relaxation methods

degradation in coding efficiency because the accuracy of PMV becomes lower without left MB information. In addition, this relaxation method is not suitable for other types of dependencies, that is, the dependencies related to intra prediction and deblocking filter. Particularly, dependencies to left MB and upper MB in deblocking filtering process can not be removed since the filtering process is strictly defined in H.264 standard.

C. Proposed MB dependency relaxation method

To relax the constraints on concurrent processing while keeping the coding efficiency, we propose a new relaxation method. As mentioned above, the dependencies to left MB and upper MB can not be removed because of the deblocking filter specification. Therefore, we propose a relaxation method shown in Fig. 6 (a) that preserve the dependencies to left and upper MBs. Instead, the dependency to upper-right MB (C) is removed and replaced to the dependency to upper MB of MB C (MB E). PMV is derived from MB A, B and E. Since the proposed method removes the diagonal dependency, all MBs in diagonal MB line are processed concurrently as shown in Fig. 6 (b). The maximum number of concurrent processing MBs is the horizontal number of MBs in a frame.

Because the dependencies to spatially nearest MBs, left and upper MBs, are preserved, the accuracy of PMV and the coding efficiency are expected to be higher than [12]. We compared BDBR of two relaxation methods, [12] and proposed, over original JM16.0 (Fig. 7). As shown in Fig. 7, the bit rate increase with proposed relaxation methods is smaller than that with [12] for all test sequences. So, the proposed method has higher coding efficiency than [12].

Furthermore, the proposed relaxation can be applied to the dependencies in intra prediction and deblocking filter. The dependency to MB C in intra prediction process can be easily removed by simply prohibiting the use of intra prediction modes that refer to pixels in MB C, specifically, mode 3 and 7 of 4x4 luma prediction [1]. Because these two prediction modes are not frequently selected, the negative impact on coding efficiency is small. As for deblocking filter, with simple MB by MB processing, current MB depends on the filtering result of MB C. However, with parallelization method proposed in [6], the dependency to MB C can be avoided. Thus, all three types of MB dependencies can be fit to proposed relaxation method. As a result, all encoding functions for MBs in diagonal MB line are concurrently processed.

In addition to concurrent MB processing within a frame, spatial-temporal parallel processing, called 3D wave front processing [4][12] is adopted to the GPU H.264 encoder.

IV. PERFORMANCE EVALUATION

The encoding speed and the coding efficiency of proposed GPU H.264 encoder is evaluated with test conditions shown in Table. 1. Since original JM16.0 encoder is too slow, we developed a simplified version of JM with encoding

Table. 1 Test conditions

CPU	Intel Core i7 (4 cores, 3.07GHz)
GPU	NVIDIA GeForce GTX 580 (512 CUDA cores, 1.54GHz)
Encoding Parameters	H.264 Baseline Profile, fixed QP (16, 20, 24, 28), N=30, 1 reference frame, RD Optimization OFF, EPZS ME
Sequences	ITE Full HD sequence [14] (CalligraphyPractice, Streetcar, Church, WhaleShow, IceHockey, HorseRace, AirplaneLanding, EuropeanMarket)

Table. 2 Simplified encoding algorithms

Integer-pel ME	16x16, 16x8, 8x16
Fractional-pel ME	One of 16x16, 16x8, 8x16
Initial candidate vectors	8 (original JM: 68)
Intra prediction	16x16: 4 modes, 4x4: 7 modes

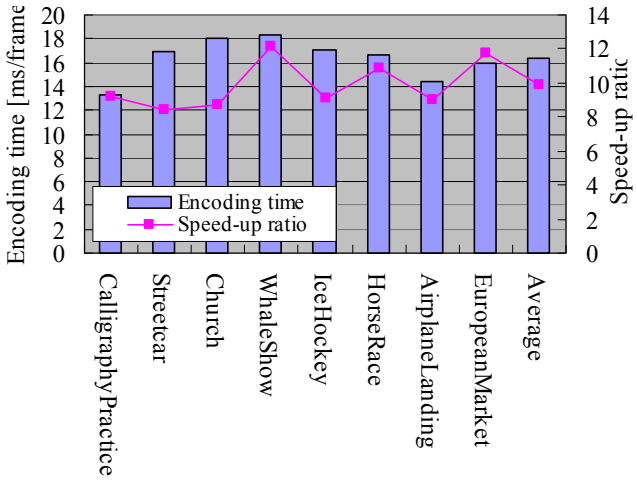

Fig. 8 Encoding time and speed-up ratio

algorithms shown in Table. 2. Almost same algorithm is implemented in the GPU H.264 encoder, but the GPU encoder uses modified PMV in motion estimation as presented before. The processing time of the GPU encoder to encode full-HD sequences and speed-up ratio, compared to simplified JM on CPU, is shown in Fig. 8. The average full-HD encoding time is 16.3 msec / frame, that is about 10 times faster than CPU encoder. The GPU encoder achieves 60 frames per second encoding of full-HD.

Fig. 9 shows BDBR evaluation result of the GPU encoder. In this evaluation, the anchor is the original JM16.0 encoder, not simplified version. The average bit rate increase is 3.5%, which is sufficiently small for practical applications.

V. CONCLUSIONS

In this paper, we presented efficient implementation of the H.264 software encoder with GPU acceleration. To fully utilize many processor cores on GPU, we proposed a new relaxation method of MB dependencies. With this method, the number of concurrent processing MBs is effectively increased while keeping the bit rate increase acceptably small. Our GPU accelerated H.264 encoder achieves 60 frames per second encoding of full-HD resolution videos.

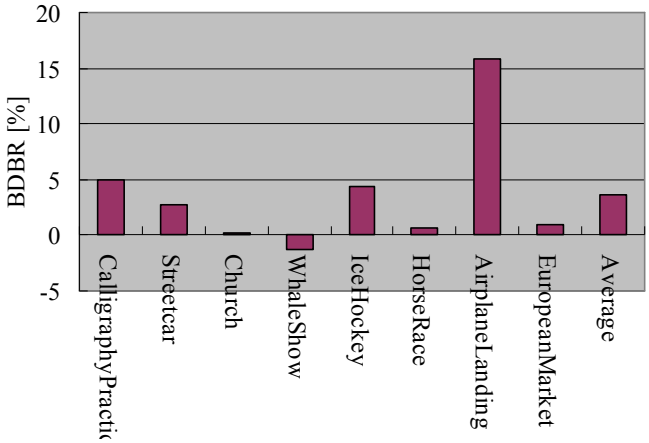

Fig. 9 BDBR evaluation of the GPU encoder

REFERENCES

[1] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services," 2003.

[2] T. Moriyoshi and S. Miura, "Real-time H.264 encoder with deblocking filter parallelization," IEEE Int. Conf. on Consumer Electronics, pp.63-64, 2008.

[3] NVIDIA, "CUDA Programming Guide," 2010.

[4] Z. Zhao and P. Liang, "data partition for wavefront parallelization of H.264 Video Encoder," IEEE International Symposium on Circuits and Systems, 2006.

[5] Y. Chen, E. Li, X. Zhou1 and S. Ge, "Implementation of H.264 encoder and decoder on personal computers," Journal of Visual Communication and Image Representation, Volume 17, Issue 2, April 2006, pp.509-532, 2006.

[6] H. Lieske and S. Kyo, "A parallel H.264 de-blocking filter implementation method for a multicore processor," PCS2010 Workshop 2010, 2010.

[7] G.J. Sullivan, T. Wiegand, "Rate-distortion optimization for video compression," Signal Processing Magazine, IEEE, vol.15, no.6, pp.74-90, 1998.

[8] A. M Tourapis, K. Sühring, G. Sullivan, "H.264/MPEG-4 AVC Reference Software Manual," JVT-AE010 2009.

[9] W.N. Chen1, H.M. Hang1, "H.264/AVC Motion Estimation Implementation on Compute Unified Device Archtecturer (CUDA)", Multimedia and Expo, 2008 IEEE International Conference on, pp.697-700, 2008.

[10] D. Ailawadi, M. K. Mohapatra, A. Mittal, "Frame-Based Parallelization of MPEG-4 on Compute Unified Device Architecture (CUDA), IEEE Advance Computing Conference (IACC), pp.267-272, 2010.

[11] M. Kung, O. Au, P. Wong, and C. Liu, "Block based parallel motion estimation using programmable graphics hardware," in Proc. Int. Conf. Audio, Language and Image Processing, pp. 599–603, 2008.

[12] A. Obukhov, "GPU-Accelerated Video Encoding", NVIDIA GPU Technology Conference, 2010.

[13] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," ITU-T VCEG M-33, 2001.

[14] "Standard test sequence for subjective assessment," ITE, the Institute of Image Information and Television Engineers, Japan.