

Highly Parallel Rate-Distortion Optimized Intra Mode Decision On Multi-Core Graphics Processors

Ngai-Man Cheung, Oscar C. Au, Man-Cheung Kung, Peter H.W. Wong, Chun Hung Liu

Abstract—Rate-distortion (RD) based mode selections are important techniques in video coding. In these methods, an encoder may compute the RD costs for all the possible coding modes, and select the one which achieves the best trade-off between encoding rate and compression distortion. Previous work has demonstrated RD based mode selections can lead to significant improvements in coding efficiency. RD based mode selections, however, would incur considerable increases in encoding complexity, since these methods require computing the RD costs for numerous candidate coding modes. In this paper, we consider the scenario where software-based video encoding is performed on personal computers or game consoles, and investigate how multi-core Graphics Processing Units (GPUs) may be efficiently utilized to undertake the task of RD optimized intra prediction mode selections in AVS and H.264 video encoding. Achieving efficient GPU-based intra mode decisions, however, could be non-trivial for two reasons. First, intra mode decision tends to be sequential. Specifically, the mode decision of the current block would depend on the *reconstructed data* of the neighboring blocks. Therefore, the coding modes of neighboring blocks would need to be computed first before that of the current block can be determined. This dependency poses challenges to GPU-based computation, which relies heavily on parallel data processing to achieve superior speedups. Second, RD based intra mode decision may require conditional branchings to determine the encoding bit-rate, and these branching operations may incur substantial performance penalties when being executed on GPUs due to the pipeline architectural designs. To address these issues, we analyze the data dependency in intra mode decision, and propose novel greedy-based encoding orders to achieve highly parallel processing of data blocks. We also prove that the proposed greedy-based orders are optimal in our problem, i.e., they require the minimum number of iterations to process a video frame given the dependency constraints. In addition, we propose a method to estimate the coding rate suitable for GPU implementation. Experimental results suggest our proposed solution can be more than fifty times faster than the previously proposed parallel intra prediction, since our work can efficiently exploit the massive parallel opportunity in GPUs.

I. INTRODUCTION

A. Motivation

State-of-the-art video encoding algorithms employ rate-distortion (RD) methods to achieve competitive coding performance [3], [4]. Specifically, to determine the optimal encoding

This work was presented in part at [1], [2]. Ngai-Man Cheung, Oscar C. Au and Chun Hung Liu are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology. Man-Cheung Kung and Peter H.W. Wong are with the VP Dynamics Labs (Mobile) Ltd. E-mail: eechung@ust.hk, eeau@ust.hk, mckung@ust.hk, eepeter@ust.hk, hungsiu@ust.hk. Tel: +852 2358-7053. Fax: +852 2358-1485. Copyright (c) 2009 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org.

parameters, an encoder may compute the bit-rate and the compression distortion of each of the possible encoding options, and select the one that achieves the minimum Lagrangian RD cost. RD techniques can be applied to determine the optimal choices of motion vectors [5], quantization parameters [6], [7], and intra/inter prediction modes [8].

This paper focuses on RD optimized intra prediction mode selection in intra-frame coding. In RD optimized intra mode selection, the encoder may compute the RD costs of all the possible prediction modes, and select the one with the minimum RD cost to encode the current block. Each prediction mode would correspond to a different intra prediction direction. In general, RD cost calculation may involve computing the intra prediction residue, transformation and quantization on the prediction residue, inverse quantization and inverse transformation, and entropy coding of the quantized transform coefficients. These operations may need to be repeated for every candidate prediction mode when encoding each video block. Therefore, the computational complexity of RD optimized intra mode selection could be very significant [9]–[11].

Focusing on software-based video encoding applications running on personal computers or game consoles equipped with both Central Processing Units (CPUs) and programmable Graphics Processing Units (GPUs), this paper investigates how RD optimized intra mode decision can be efficiently implemented on GPUs to accelerate intra frame encoding. Originally designed as specialized hardware for 3D graphics processing and rendering, GPUs have emerged recently as co-processing units for CPUs to accelerate various numerical and signal processing applications [12]–[15], thanks to their high performance multi-core architectures [13]. Achieving efficient GPU-based intra mode decision, however, could be non-trivial, as will be discussed.

B. GPU based RD Optimized Intra Mode Decision

Modern GPUs may consist of hundreds of highly decoupled processing cores to achieve immense parallel computing performance. For example, the NVIDIA GeForce 8800 GTS processor, which is used in our simulations, consists of 96 individual *stream processors* each running at 1.2 GHz [16]. The stream processors can be grouped together to perform Single Instruction Multiple Data (SIMD) operations suitable for data-intensive applications. With the advances in GPU programing tools such as thread computing and C programing interface, GPUs can be efficiently utilized to perform a variety of numerical and signal processing tasks in addition to conventional vertex and pixel operations.

In this work, we investigate the use of GPUs to perform RD optimized intra mode selection in AVS [17]–[19] and H.264 [20] video encoding. In order to take advantage of the massive processing capability, the algorithms would need to be judiciously designed to properly map into the multi-core and pipelined GPU architectures. In particular, minimum dependency between individual processor cores should be maintained during program execution, so that each processing thread can proceed independently without delay or interruption. However, this could be non-trivial to achieve in the case of intra mode decision, for two reasons. Firstly, intra mode decision tends to be sequential. Specifically, to evaluate various candidate prediction modes for the current block, it is necessary to have the *reconstructed data* of the neighboring blocks available to be used as reference pixels [18], [20]. Therefore, the coding modes of neighboring blocks would need to be first determined, and these blocks would be encoded and reconstructed accordingly. Then, prediction mode of the current block can be determined based on the reconstructed neighboring blocks. Such dependency between adjacent blocks would complicate the mapping of intra mode decision into multi-core architectures. Secondly, to determine the encoding bit-rate for each of the candidate modes, some conditional branchings may be needed. For example, AVS video encoding [17], [19] uses a context-based entropy coding scheme to encode the transform coefficients, where different VLC tables are employed adaptively to code the symbols depending on the coefficient information. Such conditional coding may require some branching operations if a straight-forward approach is used to determine the encoding rate, resulting in substantial performance penalties in GPU programs due to the pipelined processor architecture.

C. Our Contributions and Related Work

In this work, we analyze the dependency constraints in intra mode decision, and propose a strategy to determine the mode decisions of video blocks in parallel. In particular, we derive graphical models that capture the dependency relationship in intra mode decision. Based on the dependency models we propose to encode the blocks in novel orders, so that highly parallel processing can be achieved. To facilitate implementation on GPU, we also extend a bit-rate approximation method to estimate the rate in RD cost computation.

Previous work has proposed several techniques to speed up RD optimized intra mode decision, e.g., [9], [11], [21], [22]. In general these techniques try to disable some candidate modes based on some heuristics, leading to execution time reduction but at the expense of some coding performance. On the contrary, our work would evaluate all the candidate modes so that the optimal coding performance can be achieved. As will be discussed, our work addresses the block dependency issue by proposing new encoding orders, rather than disabling certain candidate modes to remove some dependency. While the latter approach could be a possible extension of the current work, it may result in degradation in coding efficiency in general.

Some recent work has proposed to use GPU for non-graphics applications. [23] proposed a high dynamic range

(HDR) texture compression system suitable for graphics hardware. [15] proposed a real-time video watermarking system running on programmable graphics hardware. [12] proposed GPU implementations of several numerical algorithms, which include matrix and vector computations. [14] analyzed the performance of fast Fourier transform and convolution in the context of image filtering on GPU. Also, some research has investigated using GPU in video compression framework. [13] proposed a system where CPU and GPU work in parallel to accelerate video decoding in the PC environment. In their system, the motion compensation feedback loop of decoding is performed on GPU, while the rests are on CPU. [24] proposed a multi-pass motion estimation algorithm in video encoding for generic GPU implementation. [25] proposed a parallel motion estimation for H.264, where each macroblock is divided into sixteen 4×4 blocks, and the cost of each 4×4 block is computed by the parallel hardware. The current research focuses on the implementation of RD optimized intra mode decision using GPU. Similar to [24], [25], the current work focuses on deriving algorithms suitable for GPU implementation, rather than investigating the optimal partition between GPU and CPU. Some recent work has also proposed to accelerate intra prediction by exploiting parallelism within a macroblock [26]–[28]. However, [26], [27] cannot be implemented on GPU, since they proposed to overlap *different* computations (prediction and reconstruction), and on GPU we can only perform *identical* computation concurrently on different stream processors. Comparing to [28], the current work proposes to exploit parallelism within a *frame*, leading to high degree of parallel computation suitable for GPU implementation. As will be discussed, by exploiting frame level parallelism, our work can outperform [28] significantly. Other work has proposed new image analysis algorithms for parallel implementation, e.g., [29]. Note that while previous work has studied multimedia processing using parallel computation, as we have summarized above, only a few have discussed parallel intra prediction, and general purpose parallel computation on GPUs is also a recent topic, while previous parallel computation work mostly focuses on other platforms (e.g., hardware implementation [27]).

The rest of this paper is organized as follows. In Section II we briefly review intra prediction. Section III presents a dependency analysis of intra decision, and proposes novel encoding orders to facilitate parallel processing. We also discuss a method to estimate the encoding rate. Section IV presents our simulation results. Finally, Section V concludes the work.

II. INTRA PREDICTION

A. RD optimized Intra Prediction in H.264

Here we give a brief overview of H.264 RD optimized intra prediction [20]. H.264 supports luma prediction block sizes of 16×16 , 8×8 and 4×4 . In this paper we focus on 4×4 intra prediction, while the analysis can be readily applied to other prediction block sizes. In H.264 4×4 intra prediction, pixels in a 4×4 block (“a” to “p” in Figure 1(a)) would be predicted by the 13 neighboring reconstructed samples (“A” to “M”

in Figure 1(a)) following one of the nine prediction modes. Each mode corresponds to DC or one of the eight prediction directions depicted in Figure 1(b). Reconstructed samples prior to de-blocking are used to form the 4×4 predictor.

To determine the optimal mode an encoder may compute the Lagrangian costs for all the nine coding modes, and select the one which achieves the minimum. The Lagrangian cost can be computed as follows (these are applicable to AVS as well). In the “low complexity mode” the Lagrangian cost J is computed by

$$J = SATD + \lambda(R_{header}), \quad (1)$$

where SATD is the sum of absolute transformed differences between the original block and prediction block, R_{header} is the estimated bits for the header information, and λ is the Lagrangian multiplier. In the “high complexity mode”, J is computed by

$$J = SSD + \lambda(R_{header} + R_{res}), \quad (2)$$

where SSD is the sum of square differences between the original block and reconstruction block, R_{header} and R_{res} are the encoded bits for header and quantized residual block.

B. Intra Prediction in AVS

The Audio and Video Coding Standard (AVS) is developed by the Audio and Video Coding Standard Working Group of China. AVS includes several related parts for video, e.g., AVS-M for mobile video applications, and AVS 1.0 for broadcast and DVD. AVS video is a hybrid coding scheme based on spatial and temporal prediction, entropy coding, transformation, quantization, de-blocking filter, etc [17]–[19].

In AVS 1.0, every 8×8 block can be intra predicted by one of the five prediction modes, each corresponding to a different prediction direction as depicted in Figure 2 [17], [18]. The vertical and horizontal modes in AVS are similar to those of H.264. The DC and down-right modes are also similar to those of H.264 but with different filters to compute the predictor pixels. The down-left mode uses bidirectional prediction and is quite different from that of H.264. Also in the DC, down-right and down-left modes, a 3-tap low pass filter would be applied to the neighboring reconstructed pixels before computing the prediction block in order to alleviate blocking artifacts.

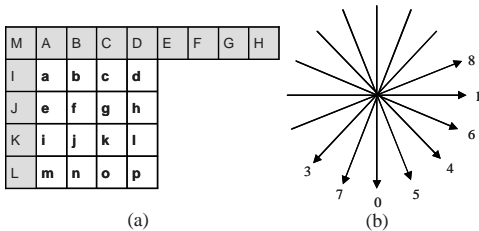


Fig. 1. Intra prediction in H.264 [20]: (a) 4×4 current blocks and their neighboring reconstructed pixels. (b) Prediction directions and their corresponding modes.

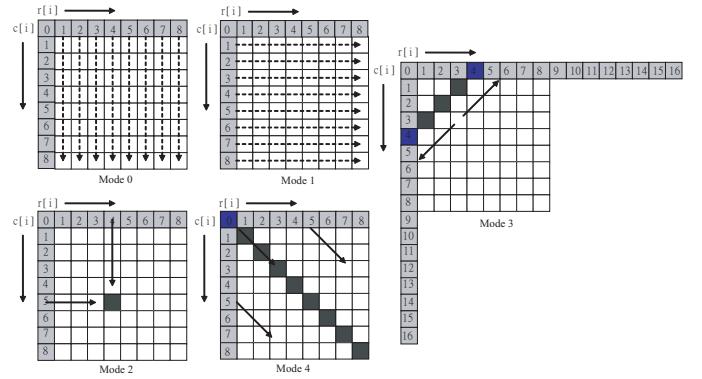


Fig. 2. Intra prediction in AVS 1.0 [18]: mode 0: vertical, mode 1: horizontal, mode 2: DC, mode 3: down-left, and mode 4: down-right. Here white squares denote the current block pixels and grey squares represent the neighboring reconstructed pixels, and $r[1-8]$ and $c[1-8]$ are the reconstructed pixels of the top and left adjacent blocks, respectively. Reconstructed pixel $r[0]$ belongs to the top-left neighboring blocks.

III. PARALLEL RD OPTIMIZED INTRA MODE DECISION

A. Dependency Analysis

As discussed, while GPU based computation relies on parallel data processing to achieve acceleration, intra prediction could result in dependency between neighboring blocks and present challenges to parallel implementations. In this section, we analyze the block dependency due to intra prediction and derive the dependency graph in encoding a video frame. Based on the dependency graph results, we propose novel block encoding orders in Section III-B to facilitate parallel RD cost computation on GPUs.

Discussion in Section II suggests the following characteristics in intra prediction may contribute to dependency constraints on block encoding order:

- 1) **Prediction direction.** Prediction directions of various modes impose the most obvious dependency among neighboring blocks. Since reconstructed pixels are used as reference in prediction, it would not be possible to determine the RD costs of the current block before all the candidate reference blocks have been encoded and reconstructed.
- 2) **Pixel filtering.** Pixel filtering such as that in AVS intra prediction may impose additional dependency among adjacent blocks. Specifically, filtering may be applied to the reconstructed pixels of the adjacent blocks before they are used in prediction, and this filtering may involve pixels from several blocks, leading to additional block dependency¹. For example, in AVS, a 3-tap filter is applied to the top and left samples in the DC prediction mode (Mode 2 in Figure 2) before these samples are used in predicting the current block. The filtering of the top sample $r[1]$ requires reconstructed sample $r[0]$ of the top-left block, and as a result, the top-left block would need to be encoded first before the RD cost of DC prediction can be computed for the current block.

¹Note that this pixel filtering in intra prediction is different from deblocking, which is applied to the current frame pixels *after* intra prediction.

By considering the prediction direction and pixel filtering characteristic of each applicable mode, we can derive the dependency between the current block and its neighbors. For example, Figures 3 and 4 show the dependency relationships when computing the intra prediction RD costs in AVS and H.264 respectively. A directed edge going from block A (parent node) to block B (child node) indicates that block B requires the reconstructed pixels from block A to determine the RD costs of various candidate modes. Therefore, block B cannot be encoded before block A .

The dependency constraints when encoding a macroblock or a video frame can then be derived by merging that of each block. Figures 3(b) and 5(b) show the dependency constraints of macroblock encoding and frame encoding in AVS respectively, and Figures 4(b) and 5(c) depict that of H.264. As suggested by the figures, the dependency relationships form directed acyclic graphs (DAG)² [30]. The figures also suggest it is not possible to parallelize the RD cost computation of the four constituent blocks of the same macroblock in AVS. On the other hand, it may be possible to compute in parallel RD costs of the blocks from *different* macroblocks. Likewise, while it is not possible to parallelize the RD cost computation of the four constituent 4×4 blocks of the same 8×8 block in H.264, it may be possible to process in parallel the 4×4 blocks from different 8×8 blocks.

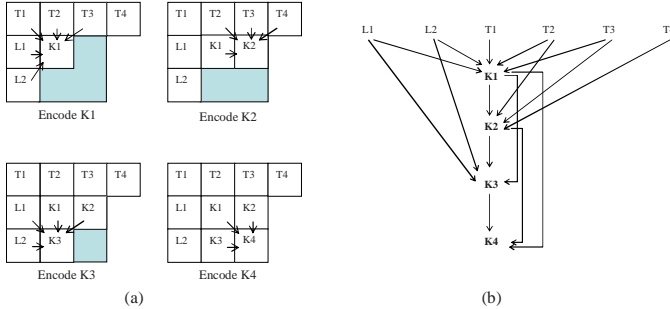


Fig. 3. (a) Dependency between the four 8×8 blocks ($K1, K2, K3, K4$) in the current macroblock and their spatially adjacent neighbor blocks ($T1, T2, T3, T4, L1, L2$), in AVS intra prediction. (b) Dependency constraints in encoding the four 8×8 blocks ($K1, K2, K3, K4$) of the current macroblock arising from AVS intra prediction.

B. Greedy-based Block Encoding Order

Based on the discussed DAG representation of dependency, we propose a greedy-based block encoding order in this section to facilitate parallel intra mode decision. In particular, when compressing a video frame (with intra coding), instead of encoding each block one by one following the conventional raster scan order, we propose to encode the blocks from different macroblocks in parallel subject to the dependency constraints in intra prediction, so that maximum parallelization can be achieved in computing the RD costs at the GPU. In the proposed encoding ordering, we encode, at each iteration, *all* the blocks of which the parent blocks have been encoded, i.e., we encode those blocks of which

²The graphs should be acyclic; otherwise, circular dependency between blocks would occur, and no proper encoding would be feasible.

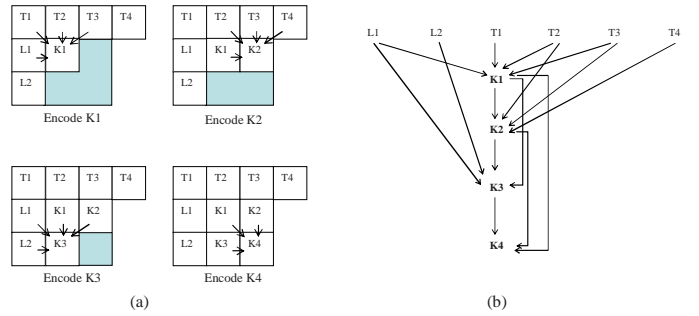


Fig. 4. (a) Dependency between the four 4×4 blocks ($K1, K2, K3, K4$) in the current 8×8 block and their spatially adjacent neighbor blocks ($T1, T2, T3, T4, L1, L2$), in H.264 intra prediction. (b) Dependency constraints in encoding the four 4×4 blocks ($K1, K2, K3, K4$) of the current 8×8 block arising from H.264 intra prediction. Note that these are similar to AVS except the dependency between $K1$ and $L2$ does not exist in H.264.

all the reference reconstructed pixels are available. This is illustrated in Figure 5(b) for AVS standard. In the first four iterations, we encode sequentially blocks $A1, A2, A3$ and $A4$ starting from the top-left most block $A1$ (i.e., the root block). Then we encode in the 5-th iteration two blocks $B1$ and $D1$ in parallel from different macroblocks. This is feasible as their parent blocks ($A2, A4$ for $B1$; $A3, A4$ for $D1$) have already been encoded. The encoding proceeds iteratively in this manner, and in each iteration those child blocks which *all* their parent blocks have been encoded will be selected for encoding. Figure 6(a) depicts this encoding order within an image frame.

To ease the implementation on a GPU, in our experiment, we use a slightly modified version of the discussed greedy-based order in AVS encoding. This is shown in Figure 6(b), where we postpone the encoding of several blocks along the left frame boundary, so that all the four constituent blocks of any macroblock could be encoded consecutively, while still meeting the dependency constraints. Comparing Figures 6(a) and (b) also suggests that the modified order does not incur any execution time penalty, i.e., Figures 6(a) and (b) require the same number of iterations when encoding a video frame³.

Figures 5(c) and 6(c) depict our proposed encoding order to facilitate parallel H.264 4×4 block intra mode decision.

We remark that while we propose to encode the video blocks in the greedy-based orders, the output bitstream will be organized conforming to the standards, i.e., bits are arranged according to the raster scan ordering of video blocks. Therefore, the output bitstream is standard compliant. The buffering and delay overheads of the proposed ordering are limited to that of encoding a single frame, and should be reasonable in personal computers or game consoles encoding environments.

Note that, under the subject of optimizing compilers, *list scheduling* [31], which is also based on the principle to process data in a greedy fashion, has been proposed to address instruction scheduling in processors with multiple functional units. However, there are some differences between the problems

³This can also be observed by examining the dependency graph in Figure 5(b). For example, postponing the processing of block $D1$ from the 5-th iteration to the 7-th iteration does not affect the scheduling of all of its descendants.

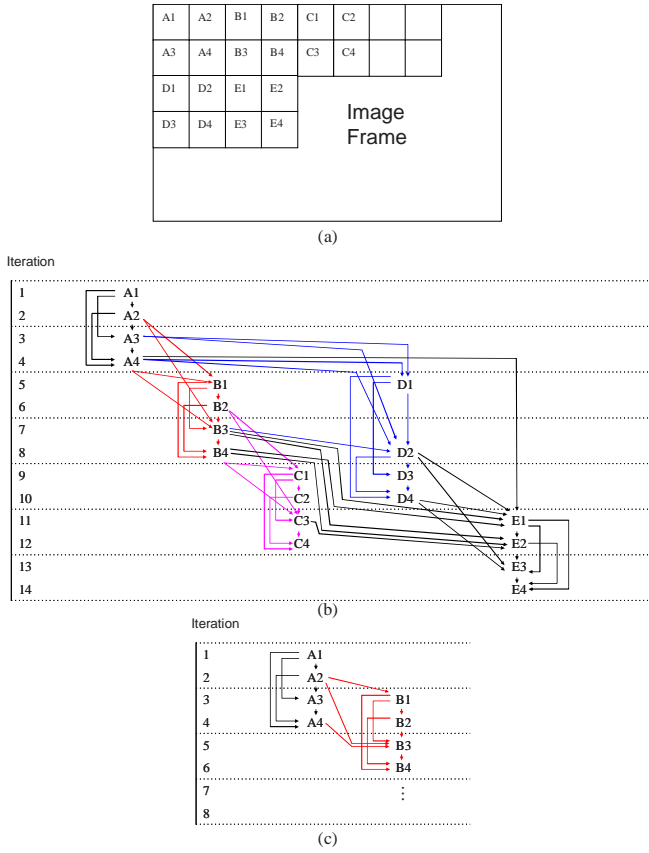


Fig. 5. (a) Notations for dependency graphs: each block corresponds to an 8×8 block in the case of AVS, or a 4×4 block in the case of H.264. (b) Dependency graph when encoding an image frame in AVS standard. Each node represents an 8×8 block. (See Figure 5(a) for notations). (c) Dependency graph when encoding an image frame in H.264 standard. Each node represents a 4×4 block. The graph is processed following the proposed greedy-based processing order, and also shown in the figure is the iteration when each block is processed. Note that a video block will be scheduled for processing *immediately* after all its parents have been processed. For example, in Figure 5(c), $B1$ is processed in the third iteration immediately after its parent $A2$ has been processed in the second iteration.

of instruction scheduling and GPU-based intra prediction, and there are also some differences between list scheduling and our independently developed encoding orders in various aspects. Specifically, the problem set-up in list scheduling (i.e., instruction scheduling) is much more general and challenging. For example, in instruction scheduling:

- 1) Different functional units may have different characteristics, e.g., some can process only integer instructions while other can process floating point instructions;
- 2) Different instructions could have different processing latencies;
- 3) The number of ready instructions may exceed the number of functional units, so there need to be some ways to select a subset of the ready instructions to execute.

On the other hand, in GPU-based intra prediction:

- 1) Different stream processors are identical;
- 2) The computations on different stream processors are also the same;
- 3) With hundreds of stream processors, in most scenarios, there would be enough processing units to handle the

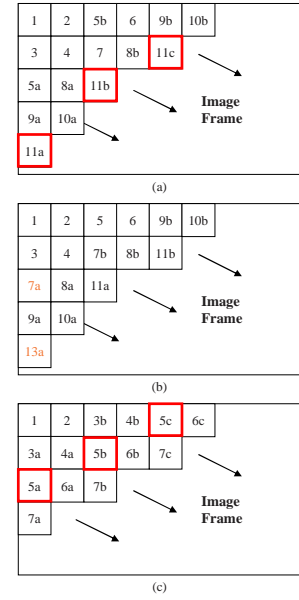


Fig. 6. (a) Our proposed greedy-based block encoding order for AVS encoding. Each square represents an 8×8 video block. Blocks with the same number (e.g., 11a, 11b, 11c) are to be processed in parallel in intra mode decision. (b) Modified version of the greedy-based block encoding order, for AVS encoding. Each square represents an 8×8 block. (c) Our proposed block encoding order, for H.264 encoding. Each square represents a 4×4 block.

ready blocks, so our work does not need to address the challenging issue of deciding how to select a subset of ready blocks for processing.

Note that these are valid assumptions for GPU computation. As a result, our problem (intra prediction on GPU) can be considerably simpler than instruction scheduling. And the consequence is that we can derive other results regarding our proposed solution to the intra prediction problem. For example, as we will discuss, we can prove that our proposed solution is optimal for RDO intra prediction, and we have analytical expression to characterize the performance (to obtain these results we also make use of some special properties of the dependency graphs pertaining only to intra prediction, as will be discussed). We do not aware similar results for list scheduling. In fact, according to [31], instruction scheduling is a NP-hard problem, and different ways to select ready instructions may lead to different list scheduling performance.

C. Optimality

In this section we present the proof that the greedy-based encoding order is optimal, i.e., under the constraints imposed by AVS/H.264 intra prediction, and among all encoding orders obeying the constraints, the greedy-based encoding order requires the minimum number of iterations to process all the blocks (i.e., determine all the optimal intra mode decisions).

Definition. We use \mathcal{G} to denote the DAG representation of dependency constraints when encoding a video frame (i.e., Figure 5(b) for AVS, where each node corresponds to a video block). We define a *path* \mathcal{P} as a sequence of blocks $\{K_1, K_2, \dots, K_n\}$, where K_i is the parent node of K_{i+1} [30]. The length of \mathcal{P} , $|\mathcal{P}|$, is n . Of all the possible paths in \mathcal{G} , we

define the *bottleneck path(s)* \mathcal{P}^* as the longest path(s) in \mathcal{G} with length n^* . It is clear that for any order, processing the entire \mathcal{G} would require at least n^* iterations. We will prove that our proposed greedy-based order can indeed process \mathcal{G} with n^* iterations.

We would like to remark that under the AVS/H.264 constraints all the paths of \mathcal{G} start out from a common *root block* (the top-left block of a frame, e.g., A_1 in Figure 5(c)) and end with a common block (the bottom-right block of a frame). These can be verified by tracing the dependency relationships (e.g., Figure 4). These properties are used in the proof.

Lemma 1. The proposed greedy-based encoding order can process all bottleneck path(s) \mathcal{P}^* with exactly n^* iterations⁴.

Proof of Lemma 1. Suppose to the contrary that the greedy-based order requires more than n^* iterations to process $\mathcal{P}^* = \{K_1, K_2, \dots, K_{n^*}\}$. Then there must exist at least one *processing gap* of length w iterations, $w > 0$, in which \mathcal{P}^* is not being processed, say between K_i and K_{i+1} (See Figure 5(b) for the gap between D_1 and D_2 of two iterations as an example). There would exist, however, an immediate parent block B_m of K_{i+1} , being processed immediately before K_{i+1} and belonging to another path. This is true because the greedy-based order would process a video block *immediately* after all its parents have been processed, and therefore any block (except the root block) would have at least one parent block being processed in the immediately preceding iteration. Likewise, there would be another immediate parent block B_{m-1} processed right before B_m (Figure 7(a)). Continuing with such backtracking eventually one would reach some block K_j in \mathcal{P}^* (one may reach \mathcal{P}^* at the root block, K_1 , the latest). The sub-path $\mathcal{P}_1 = \{K_j, B_1, B_2, \dots, B_m, K_{i+1}\}$ having no processing gap would be longer than the sub-path $\mathcal{P}_0 = \{K_j, K_{j+1}, \dots, K_i, K_{i+1}\}$ by at least w blocks. Therefore, the path $\{K_1, K_2, \dots, K_j, B_1, \dots, B_m, K_{i+1}, \dots, K_{n^*}\}$ formed by replacing \mathcal{P}_0 by \mathcal{P}_1 in \mathcal{P}^* would be longer than \mathcal{P}^* , contradicting that \mathcal{P}^* is the longest path in \mathcal{G} . \square

Theorem 1. The proposed greedy-based order can process all the video blocks in a frame in n^* iterations, where n^* is the length of the longest path in the DAG \mathcal{G} representing the dependency constraints imposed by AVS/H.264 intra prediction.

Proof of Theorem 1. The last block of a bottleneck path $\mathcal{P}^* = \{K_1, K_2, \dots, K_{n^*}\}$, K_{n^*} , would be processed in the n^* -th iteration by Lemma 1. Since all the paths would also end in K_{n^*} , all the blocks in \mathcal{G} could be processed with n^* iterations. \square

Based on the results we can also estimate the performance of the greedy-based ordering (which is also the minimum required iterations for any order). In particular, it would take n^* iterations to process a frame, where n^* is the length of the longest path(s). Such longest path can be readily found by inspecting the greedy-based order (e.g., Figure 6(c)) and tracing any path that does not have any processing gap (see the proof of Lemma 1). Figure 7(b) depicts one of the longest paths in the case of H.264 4x4 intra prediction.

⁴We remark that the result assumes there are enough processing cores to support multiple parallel computation threads, and any delay would be solely caused by inter-blocks dependency. Since GPUs with hundred cores are common nowadays [16], the assumption is reasonable.

paths in H.264 encoding. For a video frame of H pixels in width and V pixels in height (i.e., $\frac{H}{4}$ and $\frac{V}{4}$ 4×4 blocks in the horizontal and vertical directions respectively), the length can be found to be

$$n^* = \frac{V}{4} \times 2 + \frac{H}{4} - 2, \quad (3)$$

which is also the number of iterations required to process a video frame for H.264 4×4 intra prediction. The term $\frac{V}{4} \times 2$ in (3) corresponds to the zig-zag portion of the downward path.

In applications which multiple multi-core GPUs are available, a video frame may be partitioned into multiple sub-frames, with dependency between sub-frames ignored and each sub-frame to be processed by each individual GPU. In such situation our results can also help determine a more efficient partition. For example, in the case of two sub-frames, partitioning horizontally may reduce the length of the longest path (hence the required processing time) by $\frac{V}{4}$ (compare Figures 7(b) and (c)), while partitioning vertically may reduce the length by $\frac{H}{4}/2$ (compare Figures 7(b) and (d)). Therefore, horizontal partition may lead to better speed-up if $V > H/2$.

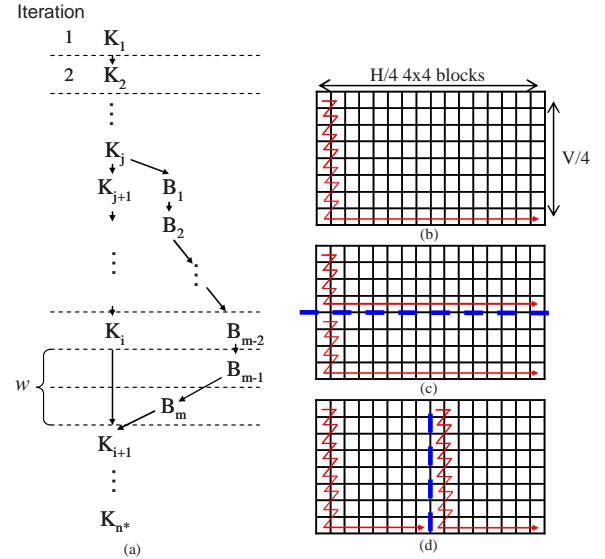


Fig. 7. (a) Proof of Lemma 1. We assume $\mathcal{P}^* = \{K_1, K_2, \dots, K_{n^*}\}$ is a bottleneck path with a processing gap of w iterations. (b) One of the longest paths in the case of H.264 4x4 intra prediction. (c) Horizontal partitioning in systems with multiple multi-core GPUs. Note the change in the longest path. (d) Vertical partitioning.

D. Bit Rate Estimation

When computing the RD costs of various modes, a straight forward approach to determine the required bit-rates would be to perform entropy coding on the transform coefficients. However, this approach may involve many branching instructions, and presents challenges to implementation on the pipeline-based GPU architecture [32]. Therefore, we choose to estimate the bit-rates by analytical expression. In particular, when calculating the H.264 intra RD costs, we adopt the model proposed in [33] to estimate the bit-rates:

$$\hat{R}_{res} = w_1 \times T_c + w_2 \times T_z + w_3 \sum_{k=1}^{T_c} |L_k| + w_4 \sum_{k=1}^{T_c} f_k. \quad (4)$$

Here T_c is the total number of nonzero coefficients, T_z is the total number of zeros before the last nonzero coefficients, $|L_k|$ is the absolute value of k th nonzero coefficient and f_k is the frequency of k th nonzero coefficient. For the weighting constants w_1, w_2, w_3, w_4 , [33] proposed to use 1, 1, 1, 0.3 respectively.

For AVS, we also use (4) for rate estimation. However, since AVS entropy coding is based on 8×8 blocks, the weighting constants could be different. Therefore, we conducted several experiments to empirically determine the weighting constants that could result in accurate estimation. Experiment results with video sequences of different resolutions and different QP values suggest, for CIF resolution, $w_1 = 1, w_2 = 1, w_3 = 1$ and $w_4 = 0.5$ can provide accurate estimation; while for 720p and 1080p resolutions, $w_1 = 2, w_2 = 1, w_3 = 2$ and $w_4 = 0.5$ can be used to estimate the bit-rates accurately.

IV. EXPERIMENTS

A. Encoding Bitrate Estimation

We first evaluate the accuracy of the bitrate estimation algorithms for AVS encoding. We use AVS RM 6.2 reference software in the simulation. Video sequences with CIF, 720p and 1080p resolutions are intra encoded. In RD optimized mode decisions, we compare the case when entropy coding is used to determine the exact rates, and the case when (4) is employed for rate estimation. Figure 8 shows the comparison results. In the figure, “With RDO” denotes the RD performance when we perform entropy coding to determine the rate, while “Proposed” denotes the cases when (4) is used for rate estimation. The figure suggests our proposed model can achieve accurate rate approximations with only negligible loss in coding performance, while it would facilitate RD cost computation on GPUs.

B. Parallel RD Optimized Intra Mode Decision

We also conduct experiments to evaluate the performance of the GPU-based RD optimized intra mode decision using the proposed encoding orders. We use a PC equipped with one GeForce 8800 GTS PCIe graphics card with 96 stream processors [16], and an Intel Pentium 4 3.2 GHz processor with 1 GB DDR2 memory, as the simulation platform. We use OpenGL API and programming language Cg to program the GPU.

We conduct experiments using the H.264 JM 14.0 reference software to evaluate the performance of our proposed encoding orders depicted in Figure 6(c) for 4×4 intra prediction. Here we compute the RD cost as in (1). Moreover, we perform prediction and reconstruction on the GPU. We use one parallel thread to compute all the costs of the nine different modes of a 4×4 block, and the subsequent prediction/reconstruction. Therefore, all the (intra-prediction related) processing of a 4×4 block is carried out on the same stream processor (and different stream processors process different 4×4 blocks)⁵.

⁵As will be discussed, in the AVS experiments, we examine a different parallelizing strategy where the processing of a block is partitioned and carried out in multiple stream processors.

We compare the average running time using our proposed encoding order with that using the conventional (sequential) raster scan order, when performing intra prediction on the GPU. Table I shows the speedups using our proposed encoding order, at different quantization parameter (QP) settings. The speedups are the ratios of GPU running time using the raster scan order to that using our proposed encoding order. As shown in the table, more than eighty times reduction in the running time can be achieved by using our proposed order, since our solution can efficiently harness the massive parallel capability of GPU. The results also suggest QP setting does not have any significant effect on the speedups. It is because intra prediction related processing is largely the same for different QP settings. We also compare our algorithm with the parallel intra prediction proposed in [28]. Note that [28] exploits only parallelism within a macroblock, which tends to be very limited. Therefore, [28] cannot fully harness the massive parallel processing capability of GPUs. As shown in Table II, our proposed solution can outperform [28] significantly⁶.

Table III shows the comparison between our proposed encoding order and the raster scan order, when RD optimization is disabled in intra mode selection. In this case, cost computation uses only distortion, i.e., SATD in (1). The results suggest our proposed encoding order can achieve similar speedups over the raster scan order when RD optimization is disabled. Table IV compares our proposed parallel solution running on the GPU with the raster scan based intra prediction running on the CPU. Note that in our platform the CPU runs at a higher clock rate, and CPU computation does not incur any CPU-GPU data transfer overhead, while GPU offers parallel processing opportunity. The results in Table IV suggest GPU-based intra prediction using our proposed encoding order compares favorably to CPU-based intra prediction. Therefore, GPU could be utilized to offload the task of intra prediction to accelerate intra frame encoding in a CPU-GPU system. We remark that it is essential to exploit the massive parallel capability of GPU in order to achieve competitive intra prediction. For example, Table V shows similar comparisons using the parallel algorithm in [28] on GPU. As suggested by the results, the GPU-based intra prediction using [28] could be considerably slower than the CPU counterpart; therefore, it may not result in any noticeable improvement in the overall performance.

We also conduct experiments using AVS RM 6.2 reference software. Following our proposed order as depicted in Figure 6(b), multiple 8×8 blocks are processed in parallel. In addition, for each block, the RD costs of each of the five candidate modes (as defined in (2)) are computed concurrently in different processing threads. Therefore, each 8×8 block uses five processing threads⁷. We use (4) to estimate the rate cost in (2), and quantized transform coefficients are used to estimate

⁶Note that, unlike our work, [28] was not proposed in particular for GPU.

⁷Note that this is different from the H.264 experiments, where each 4×4 block uses one thread to calculate all the RD costs of different modes. Since AVS 8×8 intra prediction has a smaller number of individual block than H.264 4×4 intra prediction, we use a different strategy in AVS experiments to increase the number of parallel processing threads so that the processing capability of the GPU can be sufficiently utilized.

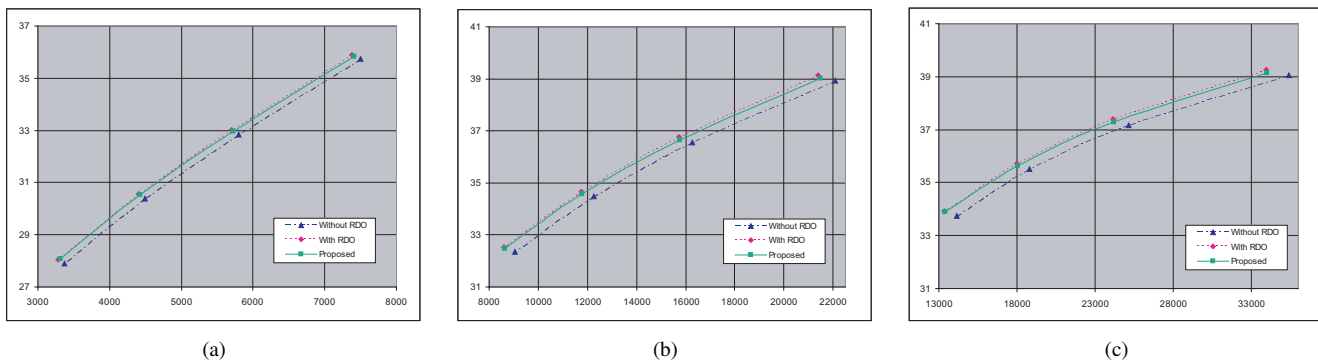


Fig. 8. RD performance comparisons using the proposed rate estimation in RD optimized intra mode decision. The horizontal axis represents the bit rate in kbps, and the vertical axis represents PSNR in dB: (a) Mobile (CIF), (b) Night (720p), (c) Riverbed (1080p). The results are the average of encoding 300 frames. For reference we also show the results when RD optimized mode decision is disabled, denoted by “Without RDO”.

the model parameters, i.e., T_c , T_z , L_k and f_k . For fair comparison, both the GPU and CPU implementations use the same rate estimation. Table VI compares our proposed order and the raster scan order. The results suggest our proposed order can accelerate AVS intra prediction by more than ten times. As discussed, in AVS experiments, we choose to compute the RD costs of the five different modes in parallel even in the raster scan order. Therefore, the improvement (over raster scan) using our proposed order is smaller in the AVS experiments comparing with that in the H.264 experiments. Table VII compares our proposed parallel AVS intra prediction on GPU and raster scan based AVS intra prediction on CPU. The results suggest GPU-based AVS intra prediction compares favorably to the CPU counterpart. Therefore, GPU can potentially be utilized to offload AVS intra prediction from CPU.

We also examine how the speedups (the ratios of CPU running time to GPU running time) change with different number of concurrent 8×8 block processing in AVS. The results are shown in Figure 9. The results suggest when GPU can accommodate all the parallel processing the speedups would increase steadily with the number of block processed in parallel⁸. However, when the number of parallel processing exceeds the GPU capability (when the number of block processed in parallel exceeds 39 in this case) the system would suffer some performance penalty due to excessive context switching. Note that, before saturation, as the number of block increases, GPU running time would remain almost constant since there are enough stream processors to handle all the processing in a single iteration, while CPU running time would increase steadily since CPU processes the blocks sequentially. Note also that in our testing platform each stream processor has the capacity to accommodate two concurrent processing threads, beyond which context switching would incur substantial overhead. Therefore, steady improvement in speedup is possible as long as the number of concurrent processing is less than twice the number of stream processors (i.e., 96×2). Recall that each block would consume five concurrent processing for each of the five prediction modes. Therefore, the maximum number of block before saturation is

⁸Note that since the measurement unit is microsecond some small random variation is inevitable due to measurement error.

TABLE I
COMPARISON BETWEEN OUR PROPOSED ORDER AND THE CONVENTIONAL RASTER SCAN ORDER, IN H.264 INTRA PREDICTION. THE NUMBERS ARE THE RATIOS OF GPU RUNNING TIME USING THE CONVENTIONAL RASTER SCAN ORDER (SEQUENTIAL) TO THAT USING OUR PROPOSED GREEDY-BASED ORDER.

	QP= 28	QP= 36	QP= 44
CIF:			
flower_cif	25.54	25.71	25.93
paris_cif	25.44	25.68	25.52
mobile_cif	25.56	25.73	25.84
Average (CIF)	25.51	25.71	25.76
1280 × 720:			
crew	61.34	60.48	61.14
night	60.81	60.62	61.09
city	60.94	60.50	60.80
Average (1280 × 720)	61.03	60.53	61.01
1920 × 1080:			
blue_sky	83.60	84.39	84.26
riverbed	84.07	83.97	84.36
station	83.96	84.49	84.02
Average (1920 × 1080)	83.88	84.28	84.21

$96 \times 2/5$, or 38.4. It should however be noted that even after saturation GPU still remains competitive and could be used to offload intra mode decision from CPU.

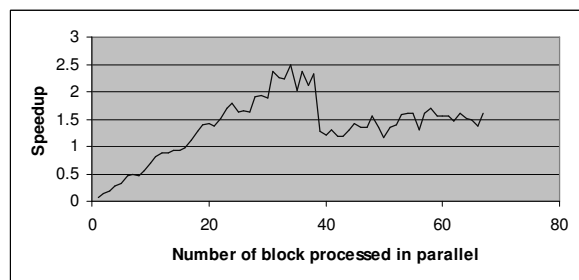


Fig. 9. Speedup with different number of 8×8 block processed in parallel, in AVS intra mode decision.

V. CONCLUSIONS AND DISCUSSION

We have discussed our proposed GPU-based RD optimized intra mode decision. Based on the dependency analysis of

TABLE II

COMPARISON BETWEEN OUR PROPOSED ORDER AND THE PREVIOUSLY PROPOSED PARALLEL INTRA PREDICTION IN [28], IN H.264 INTRA PREDICTION. THE NUMBERS ARE THE RATIOS OF GPU RUNNING TIME USING [28] TO THAT USING OUR PROPOSED GREEDY-BASED ORDER.

	QP= 28	QP= 36	QP= 44
CIF:			
flower_cif	15.99	16.12	16.25
paris_cif	15.94	16.08	15.96
mobile_cif	15.99	16.11	16.18
Average (CIF)	15.97	16.10	16.13
1280 × 720:			
crew	38.35	37.80	38.22
night	38.01	37.90	38.19
city	38.10	37.82	38.01
Average (1280 × 720)	38.15	37.84	38.14
1920 × 1080:			
blue_sky	52.25	52.75	52.66
riverbed	52.55	52.49	52.72
station	52.47	52.81	52.52
Average (1920 × 1080)	52.43	52.68	52.64

TABLE III

COMPARISON BETWEEN OUR PROPOSED ORDER AND THE CONVENTIONAL RASTER SCAN ORDER, IN H.264 INTRA PREDICTION, USING DISTORTION ONLY FOR COST COMPARISON. THE NUMBERS ARE THE RATIOS OF GPU RUNNING TIME USING THE CONVENTIONAL RASTER SCAN ORDER (SEQUENTIAL) TO THAT USING OUR PROPOSED GREEDY-BASED ORDER.

	QP= 28	QP= 36	QP= 44
CIF:			
flower_cif	25.50	25.70	25.95
paris_cif	25.37	25.62	25.55
mobile_cif	25.57	25.67	25.91
Average (CIF)	25.48	25.66	25.80
1280 × 720:			
crew	61.31	60.51	61.16
night	60.83	60.66	61.08
city	60.93	60.52	60.80
Average (1280 × 720)	61.02	60.56	61.01
1920 × 1080:			
blue_sky	83.62	84.38	84.25
riverbed	84.11	84.00	84.35
station	83.98	84.46	84.03
Average (1920 × 1080)	83.90	84.28	84.21

intra mode decision, we proposed to encode the video blocks following the greedy orders, leading to highly parallel RD cost computations. Simulation results suggested our proposed strategy can lead to more than eighty times speedup for GPU-based intra prediction, and with such performance improvement GPU can be utilized to offload intra prediction from CPU. Since intra frame encoding in general would spend a significant portion of computation on RD optimized intra prediction, our strategy can potentially reduce the overall execution time of intra frames encoding significantly. To facilitate implementation on GPU, we also extended a bit-rate approximation method to estimate the rate in RD cost computation. Simulation results suggested with our method the approximation errors in rate have only a small impact to the coding performance: no more than 0.12 dB loss in PSNR and 0.98% bit rate increase.

Our future work includes investigating the optimal partition

TABLE IV

COMPARISON BETWEEN OUR PROPOSED PARALLEL H.264 INTRA PREDICTION ON GPU AND CONVENTIONAL H.264 INTRA PREDICTION ON CPU. THE NUMBERS ARE THE RATIOS OF CPU RUNNING TIME TO GPU RUNNING TIME. NOTE THAT GPU RUNNING TIME INCLUDES ALL THE DATA TRANSFER OVERHEAD.

	QP= 28	QP= 36	QP= 44
CIF:			
flower_cif	1.14	1.12	1.14
paris_cif	1.12	1.14	1.12
mobile_cif	1.14	1.12	1.12
Average (CIF)	1.13	1.13	1.13
1280 × 720:			
crew	1.38	1.40	1.37
night	1.49	1.42	1.39
city	1.48	1.47	1.43
Average (1280 × 720)	1.45	1.43	1.39
1920 × 1080:			
blue_sky	1.90	1.82	1.73
riverbed	1.93	1.82	1.76
station	1.89	1.81	1.80
Average (1920 × 1080)	1.91	1.82	1.76

TABLE V

COMPARISON BETWEEN THE PREVIOUSLY PROPOSED PARALLEL H.264 INTRA PREDICTION [28] ON GPU AND THE CONVENTIONAL H.264 INTRA PREDICTION ON CPU. THE NUMBERS ARE THE RATIOS OF CPU RUNNING TIME TO GPU RUNNING TIME. NOTE THAT GPU RUNNING TIME INCLUDES ALL THE DATA TRANSFER OVERHEAD.

	QP= 28	QP= 36	QP= 44
CIF:			
flower_cif	0.03	0.03	0.03
paris_cif	0.03	0.03	0.03
mobile_cif	0.03	0.03	0.03
Average (CIF)	0.03	0.03	0.03
1280 × 720:			
crew	0.03	0.03	0.03
night	0.03	0.03	0.03
city	0.03	0.03	0.03
Average (1280 × 720)	0.03	0.03	0.03
1920 × 1080:			
blue_sky	0.03	0.03	0.03
riverbed	0.03	0.03	0.03
station	0.03	0.03	0.03
Average (1920 × 1080)	0.03	0.03	0.03

of the encoding flow between CPU and GPU in intra frame compression. Note that this topic could be very involved, and it requires serious evaluation on many issues. Some of them could be: (i) to investigate how to allocate the tasks between CPU and GPU such that the computation can be overlapped as much as possible; (ii) to investigate how to minimize the data transfer between main memory and GPU memory; (iii) to determine which modules can be efficiently offloaded to GPU. Given that GPU-based intra prediction is competitive, and that entropy coding may require conditional statements/branching, which are inefficient to be executed on GPU, our preliminary partition approach is to use GPU for RD optimized intra prediction, and CPU for entropy coding.

TABLE VI

COMPARISON BETWEEN OUR PROPOSED ORDER AND THE CONVENTIONAL RASTER SCAN ORDER, IN AVS INTRA PREDICTION. THE NUMBERS ARE THE RATIOS OF GPU RUNNING TIME USING THE CONVENTIONAL RASTER SCAN ORDER TO THAT USING OUR PROPOSED GREEDY-BASED ORDER. NOTE THAT IN THE CASE OF CONVENTIONAL RASTER ORDER THE COSTS OF DIFFERENT MODES OF THE SAME BLOCK ARE COMPUTED IN PARALLEL.

	QP= 28	QP= 36	QP= 44
1280 × 720:			
crew	11.90	12.58	12.54
night	12.38	12.51	12.54
city	12.31	12.48	12.56
Average (1280 × 720)	12.19	12.52	12.55
1920 × 1080:			
blue_sky	13.15	13.54	13.14
riverbed	13.28	13.38	13.24
station	13.37	13.48	13.43
Average (1920 × 1080)	13.27	13.46	13.27

TABLE VII

COMPARISON BETWEEN OUR PROPOSED PARALLEL AVS INTRA PREDICTION ON GPU AND CONVENTIONAL AVS INTRA PREDICTION ON CPU. THE NUMBERS ARE THE RATIOS OF CPU RUNNING TIME TO GPU RUNNING TIME. NOTE THAT GPU RUNNING TIME INCLUDES ALL THE DATA TRANSFER OVERHEAD.

	QP= 28	QP= 36	QP= 44
1280 × 720:			
crew	1.54	1.58	1.53
night	1.64	1.57	1.52
city	1.54	1.53	1.52
Average (1280 × 720)	1.58	1.56	1.52
1920 × 1080:			
blue_sky	1.68	1.69	1.63
riverbed	1.68	1.71	1.64
station	1.69	1.78	1.79
Average (1920 × 1080)	1.68	1.73	1.68

ACKNOWLEDGMENT

This work has been supported in part by the Innovation and Technology Commission (project no. GHP/048/08) and the Research Grants Council (project no. RPC07/08.EG22) of the Hong Kong Special Administrative Region, China. The authors would also like to thank the editor and the anonymous reviewers for their comments which helped improve the paper significantly.

REFERENCES

- [1] M. Kung, O. Au, P. Wong, and C.-H. Liu, "Intra frame encoding using programmable graphics hardware," in *Proc. Pacific Rim Conference on Multimedia (PCM)*, 2007.
- [2] N.-M. Cheung, O. Au, M.-C. Kung, and X. Fan, "Parallel rate-distortion optimized intra mode decision on multi-core graphics processors using greedy-based encoding orders," in *Proc. Int'l Conf. Image Processing (ICIP)*, 2009.
- [3] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, Nov. 1998.
- [4] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, 1998.
- [5] G. J. Sullivan and R. L. Baker, "Rate-distortion optimized motion compensation for video compression using fixed or variable size blocks," in *Proc. Global Telecommunications Conference (GLOBECOM)*, 1991.

- [6] A. Ortega and K. Ramchandran, "Forward-adaptive quantization with optimal overhead cost for image and video coding with applications to MPEG video coders," in *Proc. SPIE Digital Video Compression: Algorithms and Technologies*, Feb. 1995.
- [7] J. Wen, M. Luttrell, and J. Villasenor, "Trellis-based R-D optimal quantization in H.263+," *IEEE Trans. Image Processing*, vol. 9, no. 8, Aug. 2000.
- [8] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 7, July 2003.
- [9] C.-L. Yang, L.-M. Po, and W.-H. Lam, "A fast H.264 intra prediction algorithm using macroblock properties," in *Proc. Int'l Conf. Image Processing (ICIP)*, 2004.
- [10] R. Su, G. Liu, and T. Zhang, "Fast mode decision algorithm for intra prediction in H.264/AVC," in *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, 2006.
- [11] M. Liu and Z.-Q. Wei, "A fast mode decision algorithm for intra prediction in AVS-M video coding," in *Proc. International Conference on Wavelet Analysis and Pattern Recognition*, Nov. 2007.
- [12] J. Kruger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *Proc. International Conference on Computer Graphics and Interactive Techniques*, 2005.
- [13] G. Shen, G. Gao, S. Li, H. Shum, and Y. Zhang, "Accelerate video decoding with generic GPU," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 15, pp. 685 – 693, 2005.
- [14] O. Fialka and M. Cadik, "FFT and convolution performance in image filtering on GPU," *Information Visualization*, pp. 609 – 614, 2006.
- [15] A. Brunton and J. Zhao, "Real-time video watermarking on programmable graphics hardware," in *Proc. Canadian Conference on Electrical and Computer Engineering*, 2005.
- [16] NVIDIA, NVIDIA GeForce 8800 Architecture Technical Brief, 2006.
- [17] G. Wen, W. Qiang, and M. Siwei, "Digital audio video coding standard of AVS," *ZTE Communications*, no. 3, 2006.
- [18] L. Yu, F. Yi, J. Dong, and C. Zhang, "Overview of AVS-Video: tools, performance and complexity," in *Proc. Visual Communications and Image Processing (VCIP)*, 2005.
- [19] L. Fan, S. Ma, and F. Wu, "An overview of AVS video standard," in *Proc. Int'l Conf. Multimedia and Exhibition*, 2004.
- [20] T. Wiegand, "Joint final committee draft for joint video specification H.264," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Tech. Rep. JVT-D157, 2002.
- [21] F. Yi, J. Wang, J. Dong, and L. Yu, "An improvement of intra prediction mode coding," China AVS Standardization, Tech. Rep. AVS-M1456, 2004.
- [22] Y. Shen, D. Zhang, L. Yu, C. Huang, and S. Lin, "A simplified intra prediction method," China AVS Standardization, Tech. Rep. AVS-M1419, 2004.
- [23] J. Munkberg, P. Clarberg, J. Hasselgren, and T. Akenine-Moller, "High dynamic range texture compression for graphics hardware," *ACM Transactions on Graphics*, vol. 25, no. 3, 2006.
- [24] Y. Lin, P. Li, C. Chang, C. Wu, Y. Tsao, and S. Chien, "Multi-pass algorithm of motion estimation in video encoding for generic GPU," in *Proc. IEEE International Conference on Circuits and Systems*, 2006.
- [25] C.-W. Ho, O. Au, G. Chan, S.-K. Yip, and H.-M. Wong, "Motion estimation for H.264/AVC using programmable graphics hardware," in *Proc. Int'l Conf. Multimedia and Exhibition*, 2006.
- [26] G. Jin and H.-J. Lee, "A parallel and pipelined execution of H.264/AVC intra prediction," in *Proc. IEEE International Conference on Computer and Information Technology*, 2006.
- [27] W. Lee, S. Lee, and J. Kim, "Pipelined intra prediction using shuffled encoding order for H.264/AVC," in *Proc. IEEE Region 10 Conference (TENCON)*, 2006.
- [28] K.-W. Yoo and H.-H. Kim, "Intra prediction method and apparatus," *U.S. Patent Pub. No. US 2005/0089094*, 2005.
- [29] Y.-K. Chen, W. Li, J. Li, and T. Wang, "Novel parallel Hough transform on multi-core processors," in *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.
- [30] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Addison-Wesley, 1985.
- [31] T. C. Mowry, List Scheduling - Lecture Notes in Carnegie Mellon University CS745: Optimizing Compilers.
- [32] M. Macedonia, "The GPU enters computing's mainstream," *IEEE Computer*, pp. 106 – 108, 2003.
- [33] M. G. Sarwer and L.-M. Po, "Fast bit rate estimation for mode decision of H.264/AVC," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 17, no. 10, Oct. 2007.