Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

FIPIP: A novel fine-grained parallel partition based intra-frame prediction on heterogeneous many-core systems

Wenbin Jiang[a,*], Min Long[a], Laurence T. Yang[a,d], Xiaobai Liu[b], Hai Jin[a], Alan L. Yuille[c,e,f], Ye Chi[a]

[a] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[b] Department Computer Science, College of Sciences, San Diego State University, San Diego, CA, United States
[c] Department of Statistics, University of California, Los Angeles, Los Angeles, CA 90095, United States
[d] Department of Computer Science, St. Francis Xavier University, Antigonish, Canada
[e] Department of Cognitive Science, Johns Hopkins University, United States
[f] Department of Computer Science, Johns Hopkins University, United States

HIGHLIGHTS

- A fine-grained parallelism for intra prediction based on GPU is proposed.
- It is the first to promote intra prediction to pixel-level parallelism based on GPU.
- A new regular prediction formula is presented for parallelism.
- Two optimized encoding orders are adopted for multi-levels parallelism.
- An efficient self-synchronizing method is presented for task scheduling.

ARTICLE INFO

Article history:
Received 2 December 2015
Received in revised form
27 March 2016
Accepted 5 May 2016
Available online xxxx

Keywords:
Parallelism
Fine-grained partition
Intra-frame prediction
Fast mode decision
GPU
H.264/AVC

ABSTRACT

Intra-frame prediction is an important time-consuming component of the widely used H.264/AVC encoder. To speed up prediction, one promising direction is to introduce parallelism and there have been many heterogeneous many-core based approaches proposed. But most of these approaches are limited by their use of highly irregular prediction formulas, which require significant amount of branch instructions. They only use coarse-grained parallel partition, which considers blocks or sub-region of images as parallel processing units. In this paper, by contrast, we propose a *fine-grained intra-frame prediction approach based on parallel partition* (FIPIP) and implement it on *Graphics Processing Unit* (GPU) based heterogeneous many-core systems. The approach is characterized by the following aspects. First, our approach takes individual pixels as parallel processing units, instead of blocks. Imposing pixel-level parallelism is capable of fully exploiting the computational power of heterogeneous GPU-based systems and hence tremendously reduces the encoding time. Second, we unify irregular prediction formulas in intra-frame prediction into a well-designed uniform one, and propose a table-lookup method to efficiently perform intra-frame prediction. Our formula can eliminate unnecessary branch instructions by using a unified predictor array, which improves the efficiency of the fine-grained parallel partition significantly. Third, two optimized encoding orders assisted by an improved combined frame strategy are adopted to implement multi-level parallelism. Finally, an efficient self-synchronizing method is realized for fine-grained task scheduling on heterogeneous CPU–GPU architecture. We apply FIPIP to encode a set of benchmark videos under varying conditions and compare it with other popular intra-frame prediction methods. Results show that FIPIP outperforms existing state-of-the-art work with speedups factor of 2–6.

* Corresponding author.
E-mail address: wenbinjiang@hust.edu.cn (W. Jiang).

http://dx.doi.org/10.1016/j.future.2016.05.009
0167-739X/© 2016 Elsevier B.V. All rights reserved.

1. Introduction

H.264/MPEG-4 Part 10 or *Advanced Video Coding* (AVC) [1] is the most widely used standard for video compression. It was proposed by the ITU-T *Video Coding Experts Group* (VCEG) together with the ISO/IEC JTC1 *Moving Picture Experts Group* (MPEG). The encoder is characterized by the use of multiple references frames, improved motion estimation, and pixel-wise *intra-frame prediction* (intra prediction for short), etc. As the key component of H.264/AVC, intra prediction is one of most time-consuming steps in the whole pipeline of H.264/AVC. The software *Joint Model* (JM) [2] suggested by H.264 employs *rate-distortion optimization* [3] (RDO) to decide the optimal intra mode. Intra prediction with RDO can achieve good trade-off between encoding rate and compression distortion. As a result, the complexity and computation load increase drastically, because the encoder needs to calculate all prediction modes exhaustively to find the best one for a 4×4 pixels block.

In this paper, we investigate how to speed up intra prediction for encoding generic videos by using heterogeneous GPU-based many-core systems. In the literature, there have been many efforts in this direction. For example, Milani proposed to use spatial correlation between adjacent blocks to select a reduced set of prediction modes according to their probabilities, which were estimated adopting a belief-propagation procedure, and finally speed up the mode decision progress [4]. Pan et al. and Tsai et al. instead used the relationship between textures and prediction modes [5,6]. Other researchers implemented intra prediction on *Graphics Processing Units* (GPUs) [7,8] with the *Compute Unified Device Architecture* (CUDA) [9]. Diversified research works [7,9,10] have been done for coarse-grained intra prediction parallelism at frame-level, slice-level or block-level, which have obtained considerable speedups. However, the advantages of many-core resources such as GPUs are still not fully taken into account, because these methods usually employ a single thread to process one unit (such as a 4×4 block, a slice formed of several blocks, or even a frame). Note that a thread is the basic parallel unit of CUDA. All threads in the same grid execute the same kernel code simultaneously. To improve the parallelism degree further, a fine-grained parallel partition over pixel-level, instead of block-level, is potentially a good choice. This fine-grained parallelism, however, is likely to generate large amount of branches, and hence requires complex design. Other concerns include that several kinds of constraints over threads posed by the H.264 standard must be satisfied during encoding, which challenges the model of *Single Instruction Multiple Threads* (SIMT) of CUDA.

To address the aforementioned issues, we propose a *fine-grained intra-frame prediction approach based on parallel partition* (FIPIP) which aims to parallelize intra prediction at pixel-level on GPU-based many-core systems. To implement FIPIP, we present a unified formula to describe the prediction modes. We also propose a table-lookup based algorithm to efficiently eliminate the branches, which makes fine-grained parallelism much more efficient. Moreover, two encoding orders are presented to enable block-level parallelism to improve the parallelism degree further. Meanwhile, we introduce a combined frame strategy to sew some frames together to make a bigger frame for frame-level parallelism. Moreover, an efficient self-synchronizing method is realized for fine-grained task scheduling on heterogeneous CPU–GPU architecture.

By focusing on fine-grained parallelism, while cooperating with the other aforementioned methods, the proposed intra prediction method is capable of achieving high performance on heterogeneous GPU-based systems. The presented work is based on our previous work [11] and extends it by the following additions: (1) a 5-step sorting algorithm for the existing fast mode decision; (2) a selective encoding order neglecting intra modes 3 and 7; (3) more discussion about self-synchronizing task scheduling; (4) a more detailed evaluation on two more platforms, on frame combination, detailed speed-ups, and detailed PSNR and bit rate losses; (5) more introduction about the related work.

The rest of this paper is organized as follows. We first discuss the related work in Section 2, and introduce the background of intra prediction and CUDA architecture in Section 3. Section 4 analyzes the problems existing in multi-level parallelism methods. The proposed fine-grained parallel intra prediction is discussed in Section 5 in detail. The presented approach is evaluated in Section 6. Last, we conclude this paper and remark the future works in Section 7.

2. Related work

H.264 was approved as the video coding standard in 2003. Since then, lots of ways to accelerate the processing speed have emerged [7,12,13]. These solutions fall into two categories.

The first category reduces the computational overhead of the intra prediction, and is called fast mode decision. Pan et al. [5] built an edge map and a local edge direction histogram for each block by Sobel edge operators and chose the candidate modes according to their statistics. In a similar way, Tsai et al. [6] introduced an intensity gradient to select a subset of prediction modes. In [4], Milani S. proposed a method to determine the candidate modes based on their probabilities, by using belief-propagation. Researchers have also studied the transform domain as well as the pixel domain. Chen et al. [14] converted intra prediction from the pixel domain to the transform domain by matrix manipulation in order to obtain the transform domain predictions for various intra modes. Furthermore, [15] adopted the differences in the low frequency region to represent the total distortion of one 4×4 block. In [16], the authors even applied Support Vector Machines for Regression to intra prediction to improve the performance of H.264. Although the fast mode decision based solutions mentioned above have achieved good performance, they are limited to the serial computational process over CPU.

The second category for accelerating intra prediction is called parallel intra prediction. It is emerging in recent years and performs much faster than the ones with the fast mode decision mentioned above, as more and more parallelism-specific hardware devices (such as GPUs, stream processors and VLSI (*Very Large Scale Integration*) architectures) and related technologies spring up.

Fig. 2(d) shows a parallel intra prediction method by using an improved 10-step encoding order presented in [7,8]. It breaks through the restriction of a macroblock, which improves the GPU-based parallelism degree of the intra prediction. Fig. 2(c) depicts a 7-step order presented and used in [7,9]. These works promote the intra prediction to a higher parallelism degree by dividing a frame into a number of slices. Inside a macroblock, the 7-step order can improve the parallelism obviously. However, some prediction modes are reduced in this method, which decreases the quality of service to some extent. Ren et al. [10] presented a new real-time encoding method by applying stream processing model. Meanwhile, the 7-step encoding order is also used for high parallelism. It focuses on video sequences with 1080p resolution.

The aforementioned researches apply some revised encoding orders, which result in some loss of video quality. Moreover, the parallelism strategies used are still coarse-grained, which limits the speedup.

Aside from GPUs, some other many-core devices are also applied for the acceleration of the intra prediction. VLSI processors and stream processors are typical representatives. Wu et al. [12] implemented intra prediction on various stream processors by

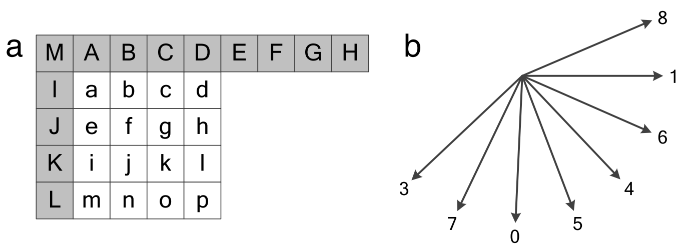


Fig. 1. Intra prediction of 4 × 4 block. (a) 4 × 4 block and its neighboring predictors. (b) Eight modes and the corresponding directions.

rewriting the X264 encoding in high-level language without special hardware/algorithm support. The experimental results showed that the presented approach can achieve significant speedup over the original X264 encoder on various programmable architectures such as X86, MIPS (*Microprocessor without interlocked piped stages*), DSP (*Digital Signal Processor*). Huang et al. [17] designed and implemented a VLSI based processor to reduce the process time of intra prediction with lots of parallel optimization measures. And in [18], Lin et al. applied the fast mode decision to the VLSI architecture, and introduced the sum of *absolute integer transformed differences* (SAITD) as part of intra prediction.

Overall, although many GPU based approaches have been proposed, fine-grained parallelism solutions are still very rare because of the high irregularity of intra prediction formulas and significant branch instructions.

The proposed algorithm in this paper is quite different from previous work. We propose a CUDA-based intra prediction algorithm that adopts the fine-grained parallel partition such as pixel-level and mode-level, whereas most conventional implementations on GPU only involve block-level or slice-level. Unlike existing works that only used single thread to deal with one block in the prediction, we employ two new encoding orders which utilize 64 threads for one block.

3. Background

3.1. RD optimized intra prediction in H.264

According to H.264, a frame can be divided into one or more slices, and each slice consists of many macroblocks whose size is 16 × 16 pixels. One macroblock can be further divided into sixteen 4 × 4 blocks or four 8 × 8 blocks. In each slice, intra prediction is performed in a raster scan order. The later blocks cannot be processed before the earlier ones in the same slice, but this restriction does not exist for blocks of different slices. H.264 supports three kinds of prediction strategies: intra-4 × 4, intra-8 × 8 and intra-16 × 16, which correspond to block size of 4 × 4 (subblock), 8 × 8 (subblock) and 16 × 16 (macroblock) pixels respectively. Among them, the intra-4 × 4 has the most complexity and is the most time-consuming one. For the intra-8 × 8 and intra-16 × 16, they have only four prediction modes. Their complexity is relative low, which makes them easy to be implemented. However, the intra-4 × 4 has nine prediction modes and the predictions are very different from each other, which make it very difficult to implement them, especially in parallel on GPUs. Many research works have been done to improve the efficiency of the intra-4 × 4 [4,6,7,9]. However, this is still far from the idea goal.

Here, we pay most attention to promoting intra prediction to a novel fine-grained level by improving the parallelism of the intra-4 × 4. For the intra-8 × 8 and intra-16 × 16, we do similar to existing work [6,7].

H.264 intra prediction uses the spatial correlation with neighboring macro blocks for prediction, i.e., a current macro block is predicted from the neighboring pixels of the previously reconstructed macro blocks. For the intra-4 × 4 prediction, as depicted in Fig. 1(a), 16 pixels of a 4 × 4 block are predicted by 13 neighboring reconstructed pixels (predictors). H.264 adopts 8 oriented modes, as shown in Fig. 1(b), and a special mode (mode 2) called DC. All these modes consider spatial correlation fully.

Only one mode will be chosen to encode the block using the RDO technique, which is the mode decision. RDO is widely used in many kinds of compression standards, and it is also an important part for mode decision in H.264. RDO makes a good tradeoff between the bit rate and the distortion for the encoded videos. The encoder computes the Lagrangian cost for each of the 9 modes as follows

$$RDvalue = Distortion + \lambda * Rate \tag{1}$$

where *Distortion* is the difference between original pixel block and the predicted one which is computed by the sum of *absolute transformed difference* (SATD) using the Hadamard transform. *Rate* means the estimated bit costs of the header information. And λ is a constant value which balances the distortion cost with the bit rate for different kinds of videos. The mode with the minimum cost is chosen as the optimum mode.

3.2. GPU features

GPUs are many-core co-processors and are capable of handling large scale parallelization by cooperating with CPUs. With large memory bandwidth and multithread capacity, it is attractive for tasks with intensive computation. To make GPUs more accessible for general purpose tasks, NVIDIA created a parallel computing platform and programming model, CUDA [19,20].

In the CUDA architecture, GPU is modeled by the SIMT architecture. SIMT instructions can specify the execution and branching behavior of a single thread. A scalable array of multithreaded *Streaming Multiprocessors* (SMs) is available in CUDA, and each SM has multiple *scalar processors* (SPs). All SPs in a SM handle the same instruction simultaneously. Threads are scheduled in a unit of 32 threads called a *warp*. In practice, a half *warp* (16 threads) is often the primitive schedule unit for one SM.

A program running on CPU launches a *kernel* which consists of a sequence of instructions to access GPU devices. In general, threads are grouped into thread blocks whose sizes are decided by the programmer. Each thread block is mapped into a SM and grouped in *warps*. If a *warp* includes multiple branches in the instruction set, different threads must wait for each other, which hurts running efficiency. Therefore we need to eliminate the branches in intra prediction by removing irregular prediction formulas.

CUDA has a three-level memory structure, i.e., global memory, shared memory and registers. These memories have different tradeoffs between capacity and speed of access. Among them, the global memory has the biggest memory capacity but the highest access delay, while the registers have the smallest capacity but the lowest access delay. Shared memories have intermediate capacity and access speed which makes it often a good choice to improve the efficiency of applications.

There are constraints for memories access. Every thread in CUDA has its own registers which other threads cannot access.

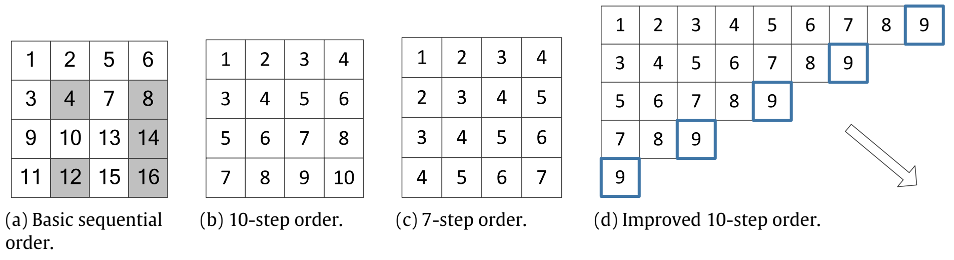


Fig. 2. Intra prediction 4 × 4 block encoding orders in H.264.

Shared memory is only visible to all threads of the same thread block. For global memory, the time cost of access is really high. We must reduce the number of the global memory calls. So we must take care of different features of different memories when we implement the novel fine-grained parallel mechanism for intra prediction.

Generally, in this paper, we pay most attention to two aforementioned features of GPU, although it has many others. One is the special memory structure of GPU, and the other is the SIMT architecture with many kernels. They will be discussed later along with the presentation of our proposed approach in more detail.

3.3. Traditional parallel solution on GPUs

Previous efforts usually implement intra prediction on GPUs with coarse-grained parallelism. They take slices or blocks as processing units and rearrange the encoding order of blocks in a frame to increase the degree of parallelism. There exist four possible encoding orders as shown in Fig. 2. Each square with a number denotes a macroblock that contains 4 × 4 pixels. The numbers in those figures indicate the encoding order of the corresponding 4×4 blocks. Sequential order is adopted as a standard in most serial intra prediction methods (see Fig. 2(a)). Kung et al. first presented a 10-step order as a parallel scheme [8] (see Fig. 2(b)). Su et al. utilized a 7-step method [9] to reduce data correlation between blocks and hence increase the degree of parallelism (see Fig. 2(c)). Cheung et al. [7] proposed to break the limitation of macroblock in the traditional 10-step encoding order (see Fig. 2(d)). Among these encoding orders, the 7-step order brings the encoded videos some quality losses, as well as the improved 10-step order.

4. Problem analysis

By aforementioned existing coarse-grained parallelism schemes, the intra prediction implemented on GPUs has almost reached its performance limitation. The introduction of fine-grained parallelism into intra prediction, such as mode-level or pixel-level, is imperative and necessary to overcome this bottleneck. Further, by combining coarse-grained with fine-grained parallel intra prediction we can better exploit the capability of many-core resources on GPUs.

What are the major problems that impede progress?

We take mode 2 (DC) and mode 3 as examples to explain the situations. Here, the situation of mode 2 is special, while mode 3 has a similar situation as other modes except mode 2. So the discussion about the two modes can cover the whole 9 modes.

Table 1 shows the prediction formulas of mode 2. The prediction value is the mean value of the adjacent available predictors. However, some special cases appear on the boundary where some adjacent predictors are unavailable for prediction. For example, for the blocks along the left edge of an image, the predictors of the left side (I, J, K and L) are unavailable.

However, the mode 3 (see Table 2) is quite different from the mode 2. We see that the prediction values of different pixels are calculated by different prediction equations in the same block. Those formulas are based on the direction of this mode. Each pixel refers to a related predictor according to corresponding direction and current position in a block.

Table 1
Formulas for prediction in mode 2 (DC).

Available predictors	Prediction formula
Upper side	$(A+B+C+D+2) \gg 2$
Both sides	$(A+B+C+D+I+J+K+L+4) \gg 3$
Left side	$(I+J+K+L+2) \gg 2$
None	128

$\gg$: left shift operation.

Table 2
Formulas for prediction in mode 3.

Position	Prediction formula
(0, 0)	$(A+(B \ll 1)+C+2) \gg 2$
(0, 1) (1, 0)	$(B+(C \ll 1)+D+2) \gg 2$
(0, 2) (1, 1) (2, 0)	$(C+(D \ll 1)+E+2) \gg 2$
(0, 3) (1, 2) (2, 1) (3, 0)	$(D+(E \ll 1)+F+2) \gg 2$
(1, 3) (2, 2) (3, 1)	$(E+(F \ll 1)+G+2) \gg 2$
(2, 3) (3, 2)	$(F+(G \ll 1)+H+2) \gg 2$
(3, 3)	$(G+H*3+2) \gg 2$

$\ll$: right shift operation.

In general, there are two major issues for intra-4×4 predictions, which make the intra prediction parallelism difficult.

(1) **Branch instructions**. As aforementioned, there are many diversified formulas for all different modes, which will bring a number of logic branches into the intra prediction implementation. The formulas in mode 3 (see Table 2) have similar forms. However, different formulas have different predictions. Moreover, in different modes, the formulas are very different. Even inside a mode, the formulas for different positions are also very different. In general, the predictors are determined by both the mode and the pixel position. We have mentioned that one CUDA thread is in charge of one pixel prediction in our fine-grained strategy. If all these prediction formulas are implemented according to their original forms, obviously, a lot of branch instructions will be produced in threads. In other words, different threads will have different logic flows. As we all know, branches in a *warp* are always big obstacles for CUDA to deal with, because CUDA will carry out all different branches sequentially among all threads in the thread block, which leads to unnecessary time cost. This kind of trouble is the major bottleneck when we try to promote the intra prediction parallelism to fine-grained level. So a new approach that eliminates the branches is urgently needed.

(2) **Particular spatial dependency**. We previously mentioned that the predictors which a current encoding block should refer usually fall on four blocks around it (see Fig. 1(a)). However, two exceptional cases should be noticed.

- **Special blocks marked in gray in** Fig. 2(a). At the upper-right of the block, there are no predictors for the intra predictions of the blocks according to the specific order. To solve

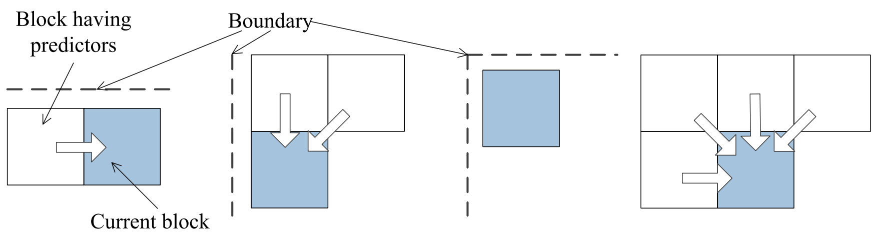


Fig. 3. Four conditions on the boundary. Some of predictors are absent for those blocks.

this problem, a suggested solution is made by H.264, which copies the value of the fourth pixel of the top side predictors to the upper right side predictors. For example, in Fig. 1(a), the values of 'E' to 'H' should equal the value of 'D' in such situation. In other words, such blocks should be differentiated from other normal blocks. This specific constraint posed by H.264 standard complicates the issue further.

- **Boundary blocks (see** Fig. 3). For those blocks along some boundary, some predictors are not available. In other words, for some prediction formulas in intra prediction modes, some predictors are absent. Therefore, we cannot make prediction for them. However, for mode 2, there is an exception. Different formulas can be selected according to the different absence of predictions (see Table 1).

In general, some special blocks that need to be particularly processed should be identified. This is one of key steps to make fine-grained parallelism successful and efficient.

To address the above two issues, we need to find a solution to make these irregular prediction formulas and particular spatial dependencies more uniform to make the GPU-based fine-grained parallelism possible and efficient.

5. Proposed parallel intra prediction

In this section, we present a fine-grained parallel partition for intra prediction on CUDA. It is also combined with a revised coarse-grained parallelism to further improve the efficiency of intra prediction.

In the following, we show how to promote parallelism of the intra prediction to fine-grained level based GPU by using CUDA. Section 5.1 discusses how to eliminate the branch instructions caused by diversified prediction formulas. Then, Section 5.2 presents some methods to improve the parallelism degree of block-level and frame-level. We introduce the task scheduling on heterogeneous CPU–GPU systems in Section 5.3, and discuss some tradeoff problems in Section 5.4.

5.1. Fine-grained parallelism

A natural solution for eliminating branch instructions is to convert the diverse prediction formulas into uniform ones, although this is a big challenge for intra prediction. Following this idea, we transfer the prediction formulas of H.264 and the predictor arrays, and then propose an efficient table lookup based algorithm.

We study all intra prediction formulas and find that it is possible to divide all of them into four classes as follows, when we use U, V and W to denote some predictors of A–M (seen in Fig. 1(a))

$$pred(x, y, m) = U \tag{2}$$

$$pred(x, y, m) = (U + V + 1) \gg 1 \tag{3}$$

$$pred(x, y, m) = (U + V * 3 + 2) \gg 2 \tag{4}$$

$$pred(x, y, m) = (U + (V \ll 1) + W + 2) \gg 2 \tag{5}$$

U, V and W are decided by variables of (x, y, m). (x, y) is the position of the predicted pixel. m means the mode number. $pred(x, y, m)$ denotes the value of the predicted pixel (x, y) with the mode m. Although for different pixels in different modes, U, V and W are different predictors, we can always find a suitable one from Eqs. (2)–(5) for a specific pixel in a specific mode (except mode 2) with proper predictors. For example, for the position (0, 1) in Table 2, Eq. (5) is suitable when U, V and W are set to B, C and D respectively. And for (3, 3), Eq. (4) is suitable by setting U and V to be G and H respectively.

Note that mode 2 is an exception. We cannot categorize it into any of the aforementioned classes. To solve this trouble, we compute the predicted value separately and store it as an independent variable named DC. Therefore, the formula (2) can be used for mode 2, where all prediction values in a block are inferred in the same way.

Eqs. (2)–(5) can be rewritten as follows, respectively,

$$pred(x, y, m) = (U + U + U + U + 2) \gg 2 \tag{6}$$

$$pred(x, y, m) = (U + U + V + V + 2) \gg 2 \tag{7}$$

$$pred(x, y, m) = (U + V + V + V + 2) \gg 2 \tag{8}$$

$$pred(x, y, m) = (U + V + V + W + 2) \gg \tag{9}$$

which share the same unified form:

$$pred(x, y, m) = (P_1 + P_2 + P_3 + P_4 + 2) \gg 2. \tag{10}$$

Herein, P_1, P_2, P_3 and P_4 are some ones of A–M shown in Fig. 1(a) or DC values for mode 2.

Now we discuss how to estimate the correspondence between individual pixels at (x, y), mode m and P_1, P_2, P_3 and P_4 exactly. This correspondence determines how to select specific values from predictors A–M and DC for different P_i ($i = 1$–4) according to different modes.

We first introduce a new organization method for the predictors. In previous research work [7–9], the predictors for each sub-block are organized as in Fig. 4(a), where the subscripts p, m and n can be obtained by Eqs. (11). A–M denote the predictors shown in Fig. 1(a).

$$\begin{aligned} n &= \text{the width of the frame}/4 \\ m &= \text{the height of the frame}/4 \\ p &= \text{MIN}\{m, n\}. \end{aligned} \tag{11}$$

For M_i ($i = 1 \sim p$), in the previous work, the number of blocks which can be processed in parallel does not extend the smaller value of m and n, so p is the minimum of them. Generally, all predictors are stored in three arrays: one for M_i, one for $I_j \sim L_j$ ($j = 1 \sim m$), and one for the rest.

Obviously, by the traditional organization of predictor arrays, due to their non-uniform styles, we still face a serious challenge to realize the parallelism of the intra prediction in fine-grained level efficiently. In order to make parallelization easier, we introduce a unified two-dimensional predictor array, as shown in Fig. 4(b), where p and A–M are the same ones in Fig. 4(a) respectively. DC is the mean value in mode 2, calculated according to the availability of predictors each time (see Table 1). Some positions

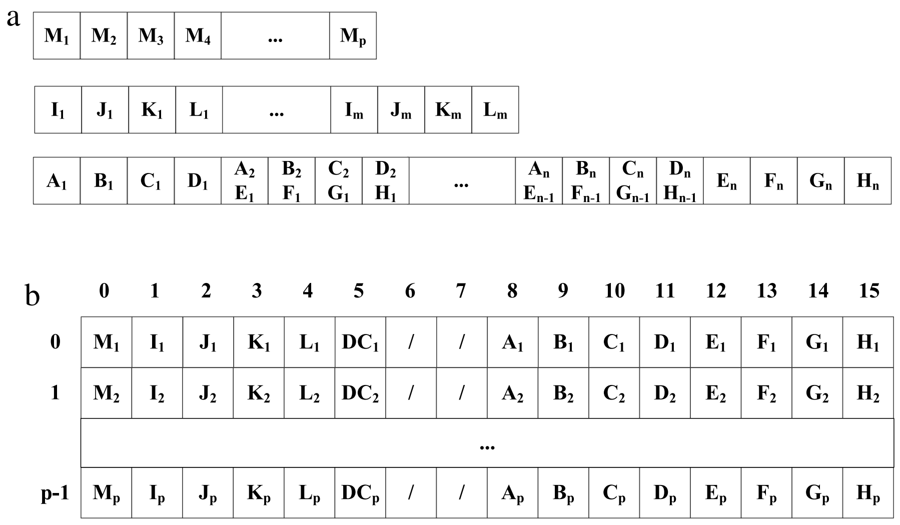

a
M1 M2 M3 M4 ... Mp
I1 J1 K1 L1 ... Im Jm Km Lm
A1 B1 C1 D1 A2 E1 B2 F1 C2 G1 D2 H1 ... An En-1 Bn Fn-1 Cn Gn-1 Dn Hn-1 En Fn Gn Hn
b
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 M1 I1 J1 K1 L1 DC1 / / A1 B1 C1 D1 E1 F1 G1 H1
1 M2 I2 J2 K2 L2 DC2 / / A2 B2 C2 D2 E2 F2 G2 H2
...
p-1 Mp Ip Jp Kp Lp DCp / / Ap Bp Cp Dp Ep Fp Gp Hp

Fig. 4. Organizations of (a) the traditional predictor arrays and (b) the proposed unified predictor array.

Table 3
The index table for the lookup of predictor array.

x	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
y	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Mode 0	8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
	8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
	8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
	8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
…																
Mode 2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Mode 3	8	9	10	11	9	10	11	12	10	11	12	13	11	12	13	14
	9	10	11	12	10	11	12	13	11	12	13	14	12	13	14	15
	9	10	11	12	10	11	12	13	11	12	13	14	12	13	14	15
	10	11	12	13	11	12	13	14	12	13	14	15	13	14	15	15
…																

are represented by '/', which means that the corresponding positions are reserved for aligning the predictor array. By this design, all predictors are organized and stored in a very regular two-dimensional array with size of $p \times 16$. This will provide easy access for GPU threads in parallel.

Moreover, the index of predictors in the array for different modes and pixel position will be stored in an index table whose size is $4 \times 9(modes) \times 16(pixels)$ (see Table 3). We demonstrate three exemplar modes and their tables in Table 3. By looking up the table, the irregular access of prediction formulas will be eliminated. Given a mode m and a pixel location (x, y), we can obtain the index by

$$index = m \times 64 + y \times 4 + x + k \times 16 \qquad (12)$$

where k is 0, 1, 2 and 3 corresponding to P_1, P_2, P_3 and P_4 in formula (10) respectively. Once we have solved the indexes of P_i (i = 1–4), we can use them to look up the unified predictor array in Fig. 4(b) to query the values of P_i, and then obtain *pred* by the formula (10).

According to the above strategy, a new algorithm based on lookup table is presented in this paper. It can move intra prediction to a novel fine-grained parallelism stage. Algorithm 1 displays the steps of the algorithm in detail. In this algorithm, the unified instruction set with only a few branches is used for the predictions of all pixels in a block. The branches brought by the diversified forms of different formulas are transformed to the irregular visits to the proposed unified predictor array. In other words, we convert the irregularity of the logic of computation flow into the irregularity of threads obtaining different predictor values from the predictor array. To guarantee the access efficiency, the shared memory, which is much faster than global memory, is used to save the predictor units.

Note that the algorithm is processed by all threads in a thread block. This improves over existing works which only employ a single thread to handle all nine modes.

However, for one pixel, computing all nine modes is still hard work. Fortunately, some fast ways have been proposed, which prune candidate modes before making mode decision. This strategy can reduce the computation cost while only introducing very limited loss of quality. To the best of our knowledge, none of these pruning methods have been combined with parallelism algorithms.

Here, we integrate the fast mode decision algorithm from [6] into our algorithm to prune the number of modes. As [6] shows, the intensity gradient can filter the modes effectively based on the probability information that mainly considers the spatial correlation between adjacent blocks. We use it to remove five modes in our algorithm.

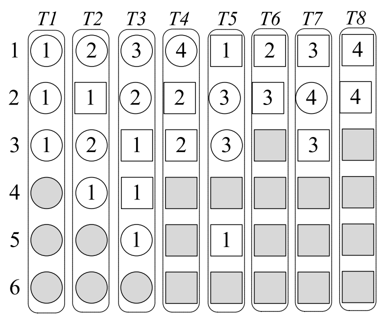


Fig. 5. 5-step algorithm. The left numbers are the steps of the algorithm. The circles and the squares of each row hold the values of the intensity gradients of the candidate modes.

Algorithm 1: the algorithm of table lookup for a pixel

```
Input:
(x, y): coordinates of a pixel in a block;
m: prediction mode;
Offset: the current predictor unit offset in the predictor unit array for
    an encoding unit;
Output:
pred: prediction value of current pixel in the mode m;
1: A1 = mode_para[$m * 64 + 4 * y + x$]; //mode_para is the index
    table for the predictor array lookup .
2: A2 = mode_para[$m * 64 + 4 * y + x + 16$];
3: A3 = mode_para[$m * 64 + 4 * y + x + 32$];
4: A4 = mode_para[$m * 64 + 4 * y + x + 48$];
5: P1 = pred_unit_array[A1 + offset]; // pred_unit_array contains
    eight predictor units which correspond to eight $4 \times 4$ blocks of an
    encoding unit respectively.
6: P2 = pred_unit_array [A2 + offset];
7: P3 = pred_unit_array [A3 + offset];
8: P4 = pred_unit_array [A4 + offset];
9: pred = (P1 + P2 + P3 + P4 + 2) >> 2;
10: return pred;
```

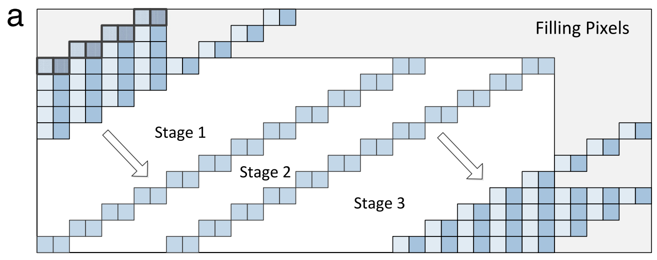
Note that every mode is dealt by one CUDA thread in our approach separately. This situation has never happened before in existing research work which dealt all modes in one thread. In this new situation, one big obstacle for parallelizing fast mode decisions is how to select the four best modes from the eight candidate modes based on the values of the intensity gradients of them efficiently. Because mode 2 can usually guarantee good efficiency, it is kept as the one of the best modes that we want. Then only other three best modes are left to be selected.

Here we propose a new 5-step algorithm to select the three best, as illustrated in Fig. 5. Every column (framed by dashed box and labeled as Ti, $i = 1$–8) denotes one thread that has the value of the intensity gradient of the candidate mode. The numbers on the left are the steps of the algorithm. The circles and the squares of each row hold the values of the intensity gradients of the candidate modes. The circle and the square which have the same number inside are treated as a pair.

In each step, the thread of the circle and the one of the square in the same pair exchange their values if necessary, to make sure that the circle holds the smaller value. For example, in the step 1, $T1$ and $T5$ constitute a pair. If the value of $T5$ is smaller than the one of $T1$, exchange their values. If not, do nothing. For other pairs in the step 1, the operations are similar. Then we go to step 2, the members of every pairs change, but the exchange rule keeps. Finally, the circles with the gray background are the modes we want, while the squares with the gray background are the ones to be removed. This algorithm can assure that the $T1$–$T3$ have the best three modes finally after 5 steps.

Although, the presented 5-step algorithm does not output the exact order of the selected three candidate modes, it exactly meets the requirement of making fast mode decisions in this preprocessing stage. Compared with some traditional sorting algorithms, it obviously has higher efficiency and is more suitable for CUDA kernel thread since it has few logical branches.

By applying the 5-step algorithm, for one pixel, only 4 modes are left for the intra prediction (adding mode 2). So we can employ 64 threads to deal with 4 modes of 16 pixels in a block. The 4 modes are calculated in parallel. In other words, for a 4×4 block, 64 threads are enough by using the Algorithm 1. Recall that the *warp* is the minimum scheduling unit whose size is 32 threads. Since every 64 threads obey the SIMT model, the branch instructions will not appear in a *warp* and the performance will not be affected. As aforementioned, in previous works [7–10], only one thread is used for dealing with one block, the degree of the parallelism is very limited. Compared with them, our proposed method with 64 threads for one block can improve the parallelism obviously.



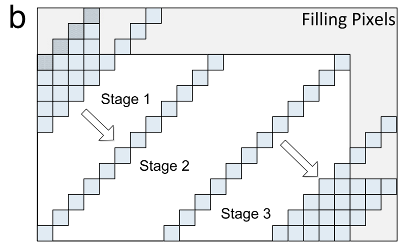


Fig. 6. The improved encoding orders. (a) The full encoding order. (b) The selective encoding order.

5.2. Coarse-grained parallelism

Although the presented novel fine-grained parallelism strategy can improve the performance of intra prediction, it is still necessary to take some coarse-grained parallelism into account. This can improve the performance further.

Here, two improved encoding orders for 4×4 block are presented to realize block-level parallelism, and a combined frame strategy to implement frame-level parallelism.

(1) Improved encoding orders

Fig. 6 shows two novel encoding orders which we presented based on the improved 10-step order (see Fig. 2(d)). They can take more advantage of GPUs.

The first one is called "full encoding order". This takes four out of all nine candidate modes into account for each pixel. As illustrated in Fig. 6(a), an encoding unit, which contains four block pairs (filled with oblique stripes) is processed by one thread block in CUDA. Why are four block pairs used in one encoding unit? Recall that there are 64 threads that are applied for one block, as mentioned in the part of showing 5-step algorithm in Section 5.1. But, in CUDA, 256 threads are generally assigned to a thread block, so it is appropriate to take four block pairs in one encoding unit which is assigned to one thread block. In a block pair, the left block will be processed first due to the dependence with the corresponding

Table 4
Average degree of parallelism for 4 × 4 block-level.

	Min	Max	Mean
CIF	1	72	40
1080p	1	270	173
2160p	1	540	345

right block, and then the right will be processed. In this way, the encoding order is compatible with the H.264 standard.

Moreover, note that there are some fragmented units on the boundaries of the frame, which have fewer block pairs (less than four). To unify the process, some meaningless blocks are padded to the frame to make every encoding unit have four block pairs. Of course, padding the frame will bring some additional computation. However, when the frame size is big enough, compared to the computation overhead for the image, the one for the padding area can be ignored. Moreover, we will introduce a combined frame strategy later, which builds a larger frame by combining some original frames. This strategy can reduce the negative effects of the padding further.

We must pay particular attention to synchronize the predictor units. As aforementioned (see Fig. 1(a)), the predictors for a block are distributed inside four neighbor blocks. In other words, a block usually produces predictors for the bottom left, lower, bottom right and right blocks. Therefore, the RAW (*read-after-write*) correlation should be noticed. To resolve the problem, the nature synchronization feature between CUDA kernels, namely self-synchronizing, is applied. We will discuss more about it later.

As shown in Fig. 6(a), when the wavefront of the process moves toward the bottom right, the number of thread blocks keeps increasing in stage 1, then reaches a peak in stage 2, and finally decreasing to one in stage 3.

The second encoding order is “selective encoding order” (see Fig. 6(b)) that removes candidate mode 3 and mode 7 directly in order to improve the efficiency of the encoding order. After removing the 3 and 7 modes, the dependence between the two blocks in the block pair of the full encoding order disappears. So the encoding unit handled by one thread block degrades into a unit of four blocks marked with oblique strips as shown in Fig. 6(b). Due to the reduction of the modes and the dependences, the computation complexity of every CUDA thread decreases. The whole speed can be improved to some extent. Of course, the elimination of two modes will result in some quality loss of the encoding. But the loss is acceptable in many situations, especially when speed is needed.

(2) The combined frame strategy

Table 4 displays the change of the average degree of parallelism as the frame size varies. The degree of parallelism means how many blocks can be processed simultaneously at a specific moment. And the average degree of parallelism denotes the average number of blocks that can be processed simultaneously during a specific time span. It is clear that bigger size is better for the average degree of parallelism. So the combined frame strategy is used, which stitches several frames into a bigger frame. A combined frame can be represented by (w, h), where w is the number of frames in vertical direction, while h is the number of frames in horizontal direction. It means that there are totally $w \times h$ frames in the combined frame. These frames are independent, so they can even be selected from different videos.

For some blocks on the boundaries of original frames, more considerations are needed. Generally, a block has predictors from the left, the upper, the upper left and the upper right blocks (see Fig. 7(a)). However, blocks on the boundaries will not obey the rule as shown in Fig. 7(c), because they may be no longer on boundaries of the combined frame. It means that these blocks refer to wrong predictors which probably lead to massive distortions in compressed videos. To solve this problem, a preprocessor is introduced to deal with the boundary blocks before starting the process. It removes some inappropriate prediction modes according to the position of the current block. For the blocks on the boundary, the modes that will cause the wrong predictions will be removed.

Before the prediction stage, all available modes of every 4 × 4 block in the combined frame will be computed and stored in global memory.

5.3. Task scheduling on heterogeneous CPU–GPU system

We implement parallel intra prediction on heterogeneous CPU–GPU system with CUDA. There is one host algorithm and two kernels in our solution. The host algorithm is responsible for the data allocation, memory transformation (transferring data between host memory and GPU memory), process control and kernel launch. The major steps are summarized in Algorithm 2. In the following, we mainly discuss about the two kernels.

(1) Preprocessor kernel

This kernel is for preprocessing. For each combined frame, the preprocessor only executes one time, and the time cost by this kernel is almost negligible compared with the prediction stage.

The preprocessor kernel is launched with a CUDA thread block size of $(4, 4, 16)$. It means that one thread block has $4\times4\times16 = 256$ threads. One thread block covers $4 \times 4 = 16$ subblocks. Every thread is for one pixel and 16 threads are responsible for one subblock. For the 5-step algorithm in the preprocessor, 8 threads selected from the 16 threads are used for every subblock.

The CUDA grid size is $(paddedWidth \gg 4, paddedHeight \gg 4)$. Here, *paddedWidth* and *paddedHeight* are the width and length of the padded combined frame.

The preprocessor generates four best candidate modes which are to be stored in global memory for all blocks in the combined frame by the parallel fast mode decision, and take special measures to handle the boundary of frames as mentioned above.

Algorithm 2: Host Algorithm of Intra Prediction

```
1: Initialize and allocate all the necessary memory and
parameters;
2: Prepare for the combinatorial frame;
3: Launch preprocessor kernel;
4: Decide the dimension of thread blocks;
5: Decide the pointer to related reference reconstructed
   array in global memory; // For the full encoding order,
      the pointer is the reference array for the 4 left blocks.
6: IntraPredictionKernel();
7: if the kernel is for full encoding order then,
    change the reference array to meet the 4 right blocks;
    IntraPredictionKernel();
8: end if
9: if it is not the last block in current frame then
10:     goto 4;
11: end if
12: Copy related data back to host memory;
13: Other processes of encoding.
14: if it is not the last frame then
15:    goto 2;
16: end if
17: Free all memory;
18: Return;
```

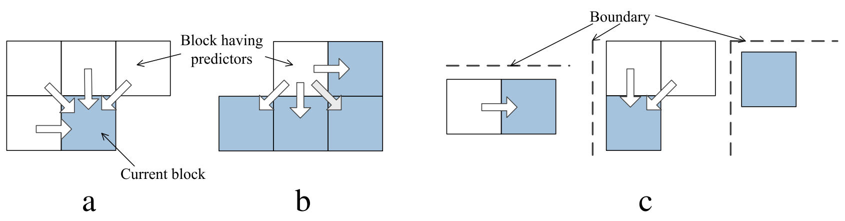


Fig. 7. (a) Dependent predictors of current block. (b) Predictors generated by current block for adjacent blocks. (c) Three cases for blocks on the Boundary. (The black line is the boundary.)

(2) Intra prediction kernel

This kernel is for intra prediction. It is responsible for pixel-level parallel intra prediction. One thread is in charge of the prediction of one pixel.

Algorithm 3 shows the intra prediction kernel of IntraPredictionkernel() in Algorithm 2. The size of the thread block in this kernel is also set to (4, 4, 16), namely 256 threads (the threadIdx in Algorithm 3 is the index of these threads). As mentioned in Section 5.2, one thread block covers 4 block (4×4 subblock) pairs (for full encoding order) or 4 blocks (for selective encoding order), Namely 64 threads cover one block pair or one block. Moreover, the grid size is set according to the number of blocks needed by the second stage shown in Fig. 6.

In the kernel, only the prediction sub-process can utilize all 256 threads, whereas others can only use 128 threads at most. For example, for the stage of preparing the unified predictor array for each thread block, one thread is responsible for one predictor copying. In every block pair, there are $16 \times 2 = 32$ predictors to be copied from global memory to shared memory (as Fig. 4(b) shows that every subblock has 16 predictors, among which, 13 ones are valid). Therefore, for 4 block pairs in the thread block, $4 \times 32 = 128$ threads are required for this copying operation.

For other sub-processes such as computing low complexity RD values, choosing the mode with minimum value as the best mode, there are only 16 threads required for the 4 block pairs. And for sub-processes such as integer transform, quantization and dequantization process, only 64 threads are needed.

In addition, most of sub-processes in the intra prediction kernel are based on the table lookup algorithm shown in Algorithm 1 if it is needed.

Here, we pay more attention to the intra prediction kernel for full encoding order. As aforementioned, in full encoding order, one thread block covers 4 block pairs. For two blocks in one block pair, the left block should be processed before the right one (see Fig. 6(a)) because of data dependence. If we use one kernel to deal all 4 block pairs, the kernel should be divided into two parts. The first part is for all four left blocks and the second part is for all four right blocks. Before starting the second part, all threads of the thread block must wait for all other threads to finish their first part computation. Some synchronization method is needed. In general, atomic functions provided by CUDA can be used to perform this synchronization. However, the atomic operations are very time-consuming for some GPUs that do not have a second level cache of global memory.

Here we make a self-synchronizing method by breaking the task in the original kernel into two sub-tasks packaged in two smaller kernels (see Fig. 8). One kernel is responsible for the four left blocks and the other kernel is for the four right blocks. Actually, we can see that the two kernels have the same logic flow except for different input data such as predictors.

Therefore, CPU launches the two small kernels to GPU one by one for every 4 block pairs. Then, self-synchronizing can be achieved between the two small sequential kernels efficiently. No other atomic functions are needed.

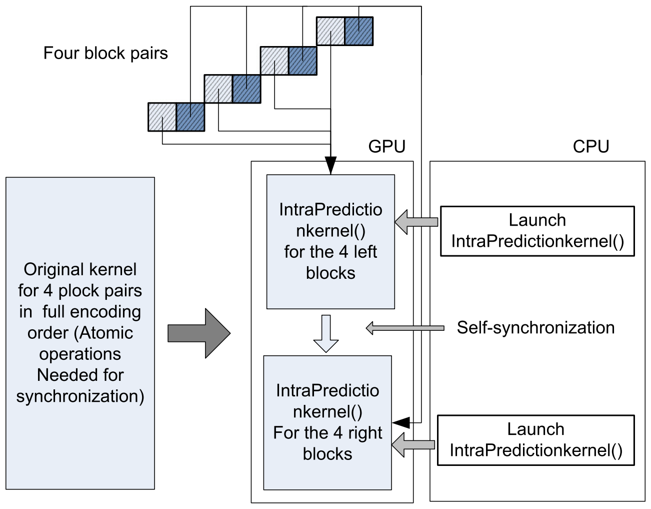

Four block pairs
GPU
CPU
Original kernel for 4 plock pairs in full encoding order (Atomic operations Needed for synchronization)
IntraPredictionkernel() for the 4 left blocks
Launch IntraPredictionkernel()
Self-synchronization
IntraPredictionkernel() For the 4 right blocks
Launch IntraPredictionkernel()

Fig. 8. A self-synchronizing method: breaking the task in the original kernel into two sub-tasks packaged in two smaller kernels.

5.4. Discussion

In this section, we will discuss some tradeoffs when implementing the proposed parallel intra prediction. There are three types of tradeoffs.

(a) **Bank conflicts**. In Section 5.1, we propose a table lookup algorithm which succeeds in eliminating branches of the irregular prediction formulas. In fact the irregular feature is relocated to the operations of block threads retrieving predictors from shared memory, which perhaps results in terrible bank conflicts. 16 threads in a half *warp* for Algorithm 1 which is scheduled in one SM in parallel, may simultaneously access some predictors. In other words, some predictors in the same banks may be accessed by different threads at the same time, which causes bank conflict in shared memory. Fortunately, our experimental results show that the bank conflicts are limited with few side effects on performance.

(b) **Computation of blank blocks in padding**. The padding area brings more computation. However, as the frame size becomes larger, the ratio of the additional computation to the original goes down. Therefore, the combined frame strategy can decrease the negative impacts introduced by the padding operations.

(c) **The effect of fast mode decision**. We adopt fast mode decision in the proposed algorithm to prune modes that are less likely to be the optimal one. This pruning step might result in loss of video quality and increase of the bit rate of encoding. However, many existing works with fast mode decision have shown that fast mode decision only generates a few side effects for intra prediction.

Table 5
Machines with different GPUs.

No.	1	2	3
CPU	Intel Xeon E5-2670 @ 2.60 GHz	Intel Core i7 950 @ 3.07 GHz	Intel Core i7 920 @ 2.67 GHz
GPU	NVIDIA Tesla K20	NVIDIA GTX460	NVIDIA GTX295
Memory	32 GB	16 GB	6 GB
OS	Redhat 6.2 (2.6.32–220)	Centos 5.9 (2.6.18–348)	Redhat 5.5 (2.6.18–194)
Compiler	GCC 4.4.6	GCC 4.1.2	GCC 4.1.2
CUDA runtime	5.0	4.0	2.3

```
Algorithm 3: Intra Prediction Kernel
1: **if** threadIdx < 128 **then**
2:     Prepare the unified predictor array for each thread block;
3: **end if**
4: Perform the prediction for 4 blocks in a thread block simultaneously;
5: Perform Hadamard Transform and sum up the results;
6: **if** threadIdx < 16 **then**
7:     Compute low complexity RD value;
8:     Choose the mode with minimum value as the best;
9: **end if**
10: if threadIdx < 64 **then**
11:     Perform integer transform;
12:     Perform quantization and dequantization process;
13:     Perform inverse integer transformation;
14:     Generate the unified predictor array for related blocks;
15: **end if**
16: Return;
```

Table 6
Videos for evaluation.

ID	Video name	Resolution	Frames	FPS
1	Bus	352 × 288	96	30
2	City	352 × 288	288	30
3	Football	352 × 288	192	30
4	Foreman	352 × 288	288	30
5	News	352 × 288	288	30
6	Cactus	1920 × 1080	480	50
7	Blue sky	1920 × 1080	192	25
8	Pedestrian area	1920 × 1080	288	25
9	Riverbed	1920 × 1080	192	25
10	Rush hour	1920 × 1080	480	25
11	Station2	1920 × 1080	288	25
12	Sunflower	1920 × 1080	480	25
13	Tractor	1920 × 1080	672	25
14	Crowd run	3840 × 2160	480	50
15	Ducks take off	3840 × 2160	480	50
16	In to tree	3840 × 2160	480	50
17	Old town cross	3840 × 2160	480	50
18	Park joy	3840 × 2160	480	50

6. Experimental evaluation

6.1. Experimental environment

To thoroughly evaluate the performance of the presented approach, three kinds of machines with different GPU devices are used to conduct experiments. The details of the hardware devices and the environments are shown in Table 5. We use three kinds of GPUs, which are K20, GTX460 and GTX295. They represent three kinds of popular architectures of GK110, GF104 and G200 respectively. NVIDIA nvcc and GCC compilers are applied.

We apply our approach to 18 standard video sequences with resolutions of CIF (352×288), 1080p (1920×1080) or 2160p (3840×2160) as shown in Table 6 [21]. These videos are captured for varying types of scenes and include various challenges such as static scenes with highly detailed structure, active motion blur with few details.

We compare our approach with other five algorithms quantitatively. To make fair comparison, the fast mode decision [6] is both involved in our proposed algorithm and some existing algorithms.

Serial Algorithm 1A is the baseline object for comparison. It is from JM 18.2 [2] and executed on CPU.

Serial Algorithm 1B adopts the fast mode decision [6] based on 1A algorithm.

Parallel Algorithm 2 takes the 7-step order (see Fig. 2(c)) into account and the strategies of slice-level and block-level parallelisms are realized.

Parallel Algorithm 3A takes the improved 10-step order (see Fig. 2(d)). It is realized based on the work of [7] on CUDA.

Parallel Algorithm 3B adds the fast mode decision to 3A.

Proposed Algorithm 4A applies all optimizations in Section 5 (the fast mode decision, the selective encoding order by removing the prediction modes 3 and 7, etc.). The combination scheme for CIF is (8, 12), as well as (8, 12) for 1080p and (4, 6) for 2160p.

Proposed Algorithm 4B is similar to the proposed Algorithm 4A. The version of 4B does not remove the prediction modes 3 and 7.

Three QP (quantization parameter) values are employed in the experiments. They are 20, 28 and 36, which represent high quality, normal quality and low quality respectively. The QP value is the quantization step size employed in the quantization and dequantization stages. With larger QP value, the bit rate of videos gets smaller and the video quality degrades.

6.2. Optimal combination scheme

We first introduce the combination scheme used for the proposed algorithms during evaluations. Recall that in Section 5, the maximal degree of parallelism depends on size of the combined frame. By increasing the maximal parallel scale, the performance of the proposed algorithm improves. So we need to compare different combination schemes to find the optimal one.

As shown in Fig. 9, the average encoding time of the proposed Algorithm 4B is tested for different combination schemes on Machine 1 in this experiment. The color bar stands for the average encoding time (in ms) of specific resolution. White areas mean that the combination schemes on corresponding coordinates are invalid. The coordinates with low encoding time occur in the center.

The x axis and the y axis represent the width and height (in frames) of the combined frame in normal frames. Good performance can be achieved with large combined frame. Note that in Fig. 9(c), the value of (5, 6) and (6, 5) is bigger than that of (4, 6) and (6, 4). That is because the proposed algorithm runs out of shared memory or registers. In other words, the size of the optimal combined frame is the one factor that may make the shared memory or registers reach its limit. Based on the experimental results, we choose scheme (8, 12) for CIF and 1080p and scheme (4, 6) for 2160p, according to the performance shown in the figure.

6.3. Performance comparison

We apply all seven algorithms mentioned above over the 18 video sequences in Table 6 and measure their respective

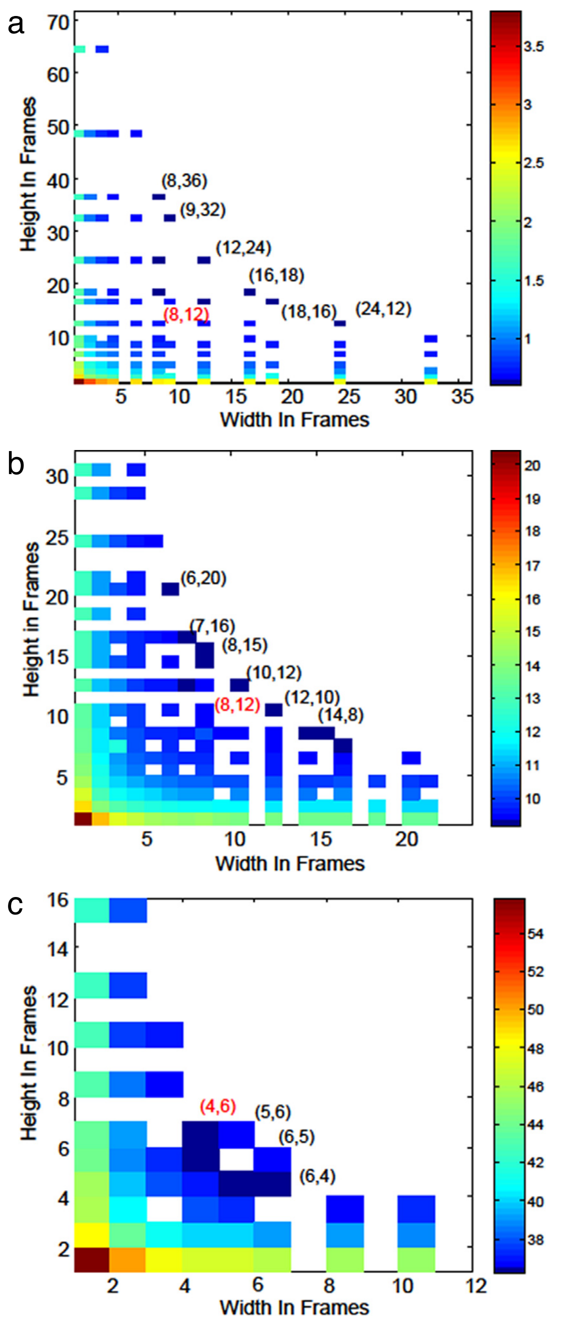


Fig. 9. Combination schemes map tested on machine 1. (a) CIF. (b) 1080p. (c) 2160p. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

encoding time. Firstly, we perform experiments on Machine 1 with different QPs to demonstrate how performance changes when using different video qualities. Secondly, we conduct experiments on three machines with the same QP to illustrate how different environments affect the encoding time. In this section, we regard Algorithm 1A as the baseline algorithm and use the speedup ratios to Algorithm 1A to compare performance of other algorithms.

Experimental results on Machine 1 (with different QPs) are depicted in Table 7. The speedups are reported based on the average values. The data transmission time between device memory on GPU and host memory has been considered. From the table, one can observe that Alg. 3B and Alg. 1B get higher speedups (both about 1.3 times) than Alg. 3A and Alg. 1A respectively. It demonstrates that fast mode decision can largely reduce computational complexity. Alg. 2 reduces the encoding time to the half of Alg. 1A approximately.

The speedups generally are low. The low degree of parallelism is the main reason. Even though the modes 3 and 7 have been removed from Algorithm 2, it makes little difference. Note that the Algorithm 3A and Algorithm 3B have widely changed speedups. The cause is the size of the frame, which is the key factor that affects the degree of parallelism. The poor performance of Algorithm 3A and 3B is caused by the small resolution.

Our proposed Alg. 4A and Alg. 4B outperform all the other algorithms under all conditions. In practice, Alg. 4B always performs better because of the better tradeoff between the encoding speed and the quality loss. The reason why Alg. 4A surpasses Alg. 4B in encoding time is that two modes for prediction are eliminated in Alg. 4A. Therefore, Alg. 4A will suffer from quality loss like Alg. 2, which only has an average speedup of 2× relative to Alg. 1A. Due to the low degree of parallelism, Alg. 2 cannot exploit the performance of Machine 1. Compared with Alg. 3B, Alg. 4B doubles the speedups in the 1080p and 2160p video sequences and has 6× speedup in CIF video sequences. The poor performance of Alg. 3A and Alg. 3B on CIF is caused by the small amount of parallelism. Alg. 4A and Alg. 4B overcome the problem by introducing the combined frame strategy. In general, the performance improves along with the increase of the frame size. The details of performance with different QP values are shown in Fig. 10.

Additionally, notice the relationship between the value of QP and the speedups. It is clear that the speedups of those CUDA-based algorithms relative to Algorithm 1A decrease as the value of QP increases. It is mainly because the traditional Algorithm 1A uses an optimization strategy that zero values will be detected and skipped by the encoder in quantization stage (it is reasonable in such kind of serial algorithms), while, this optimization cannot be used in the CUDA-based implementations. The main reason is that lots of logic branches would be produced to judge whether there are zero values in the quantization stage, then decide whether to skip them or not. This optimization also results in the fluctuation of processing time in traditional serial prediction algorithms. While, we cannot see this kind of fluctuation in parallel ones due to lack of this optimization. It is clear that the proposed algorithm is independent from the image content. In other words, the complexity of the image content cannot affect the effectiveness of the parallel algorithms.

The fast mode decision is used in our presented algorithm, which decreases the quality of videos and increases the bit rates to some extent. Table 8 gives the results in detail. Let ΔPSNR denote the deterioration of PSNR (*Peak Signal to Noise Ratio*), and ΔBR denote the deterioration of the *bit rate* (BR). They can be obtained by formulas (13) and (14).

$$\Delta\text{PSNR} = 1/n \times \sum(\text{PSNR}_k - \text{PSNR}_{ko}) \quad (13)$$

$$\Delta\text{BR} = 1/n \times \sum((\text{BR}_k - \text{BR}_{\text{ko}})/\text{BR}_{ko}). \quad (14)$$

Here the subscript k is used to indicate the corresponding variable is obtained from the algorithm that employs the fast mode decision, while the subscript ko is for one without fast mode decision. The n is the number of the videos with specific resolution.

It is clear that we can neglect the average PSNR loss that is less than 0.1 dB. However, for bit rates, the increases cannot be ignored because about half of them go over 5%. However, for some situations, such as cloud gaming service, which are more sensitive to the latency than to the bandwidth, the increases of bit rates are still acceptable.

Table 9 shows the speedups on different machines (the results on machine 1 were shown in Table 7) with the same QP (equal to 20). All algorithms have similar performance on different machines

Table 7
Speedups compared to Alg. 1A [2] (Machine 1).

	Alg. 1B [6]	Alg. 2	Alg. 3A [7]	Alg. 3B [7,6]	Alg. 4A	Alg. 4B
QP = 20						
CIF	1.33	2.14	1.05	1.38	9.38	7.82
1080p	1.35	1.93	3.87	4.89	13.08	12.34
2160p	1.31	2.70	5.65	7.17	14.84	14.05
QP = 28						
CIF	1.36	1.99	0.98	1.28	8.72	7.29
1080p	1.36	1.80	3.59	4.54	12.09	11.41
2160p	1.36	2.46	5.25	6.49	13.06	12.37
QP = 36						
CIF	1.36	1.84	0.89	1.16	7.96	6.65
1080p	1.35	1.68	3.32	4.20	11.15	10.52
2160p	1.38	2.32	4.87	5.89	12.28	11.59

Notice: Alg. XX denotes the Algorithms XX mentioned in Section 6.1.

Table 8
Average loss of video quality and the increase of bit rate (Alg. 4A to Alg. 1A).

QP	Resolution	ΔBR%	ΔPSNR
	CIF	2.16%	−0.016
20	1080p	3.10%	−0.023
	2160p	1.30%	−0.026
	CIF	3.67%	−0.029
28	1080p	7.89%	−0.047
	2160p	2.89%	−0.049
	CIF	6.59%	−0.022
36	1080p	12.02%	−0.080
	2160p	7.68%	−0.087

Table 9
Average speedup compared to Alg. 1A (QP = 20).

	Alg. 1B	Alg. 2	Alg. 3A	Alg. 3B	Alg. 4A	Alg. 4B
Machine 2						
CIF	1.39	1.81	1.97	2.70	11.71	9.66
1080p	1.41	2.11	3.16	4.58	13.12	11.40
2160p	1.38	2.92	3.81	5.57	14.45	12.41
Machine 3						
CIF	1.39	3.20	1.02	1.38	9.51	7.51
1080p	1.40	2.11	1.30	2.03	9.37	7.90
2160p	1.37	0.35	1.53	2.43	10.18	8.92

expect for abnormal results of Alg. 2 on Machine 3. The speedup decreases as the size of frame increases. This might be caused by the limited host memory of Machine 3 and the instability of Alg. 2. The details of performance for each video sequence on different machines are shown in Fig. 11. (The case on machine 1 can be seen in Fig. 10.) According to Table 7 and Table 9, we observe that the proposed algorithms are very stable and are able to achieve good performance on machines with different devices. They can handle video sequences from small size to very big size such as 8k × 4k. Compared with the state-of-the-art Alg. 3A and Alg. 3B, the proposed algorithm can achieve 2×–3× speedup in 1080p and 2160p and 5×–6× speedup in CIF.

6.4. *Evaluation of fast mode decision*

In Section 5, we introduced the fast mode decision algorithm into the proposed solution. The computational complexity decreases by pruning modes which are usually not the optimal mode. However, we have to accept side effects, *i.e.*, loss of video quality and the increase of bit rate of encoding. In Fig. 12, we present detailed statistics caused by fast mode decision.

For most of video sequences, the PSNR loss is less than 0.1 dB and the bit rate increases less than 10%. With lower QP values, the performance of the algorithm becomes better. Therefore, sometimes people tend to employ lower QP values to obtain a better quality video with larger bit rate.

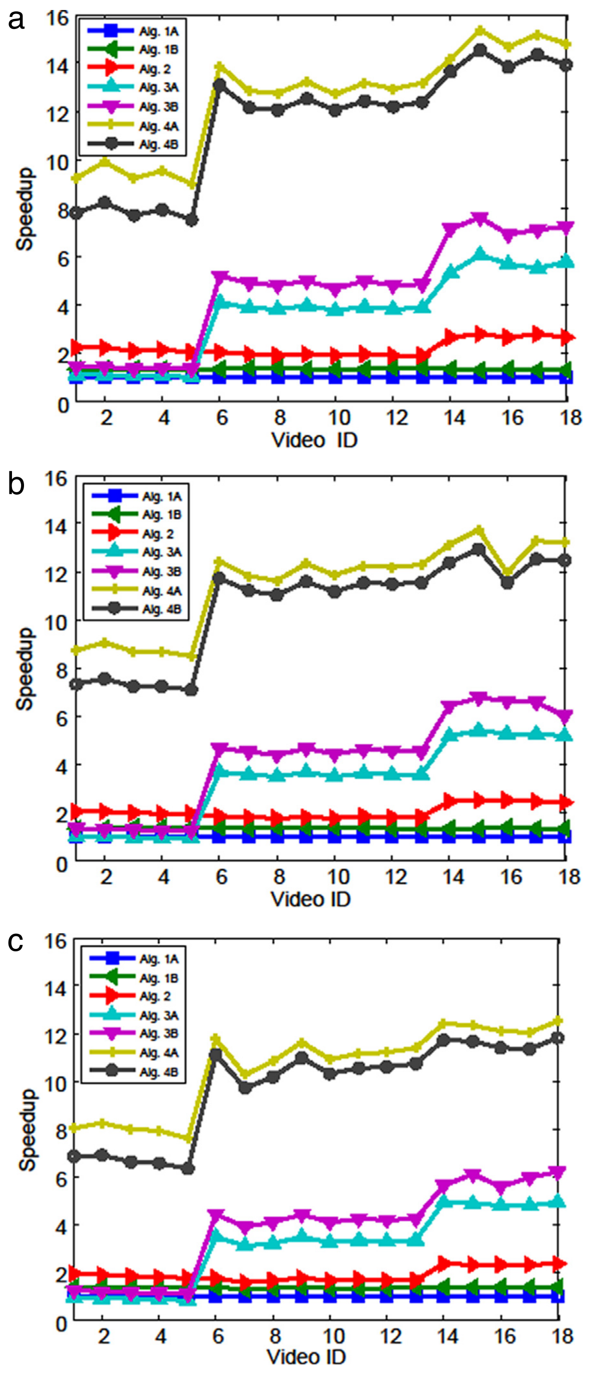


Fig. 10. Intra prediction on machine 1. (a) QP = 20. (b) QP = 28. (c) QP = 36.

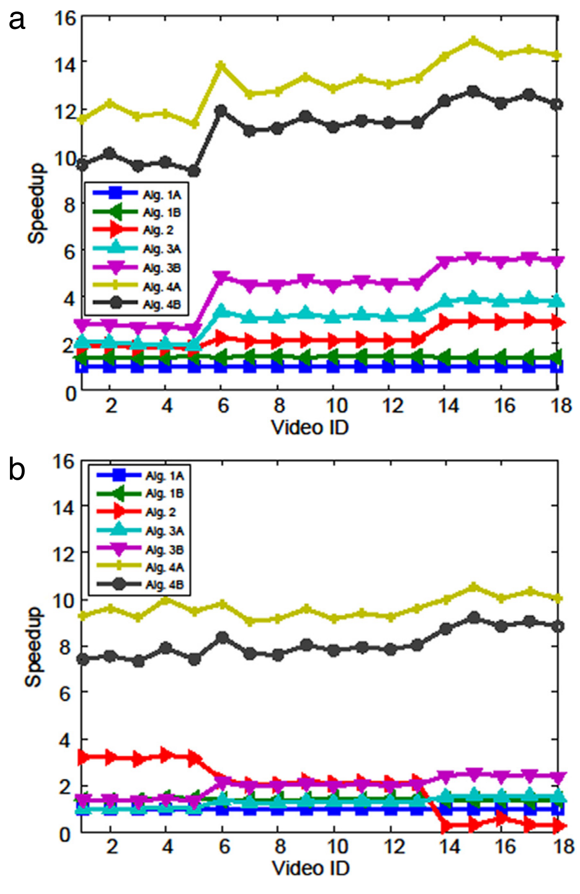


Fig. 11. Intra prediction with QP = 20. (a) Machine 2. (b) Machine 3.

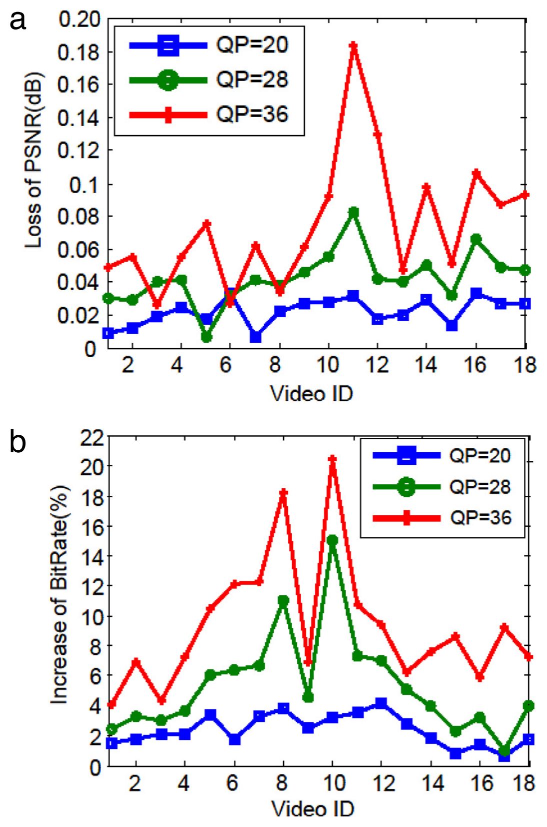


Fig. 12. Performance of fast mode decision. (a) PSNR loss. (b) Bit rate increase.

Note that there are some abrupt peaks in the figures. This is because the fast mode decision based on intensity gradient is not good at dealing with videos with many details. Fast mode decisions with different QP values have preference on video sequences of different styles. QP value should be chosen according to the video style to ensure high performance. For example, for the video of No. 11, low QP value resulted in obvious loss of PSNR. It is not a good idea to use fast mode decision with low QP value. However, for No. 13, the loss of PSNR and the increase of bit rate with low QP value are both acceptable.

Here one thing should be made clear that the fast mode decision algorithm is presented by previous research [6]. We just used it in our algorithm. Therefore, the loss of PSNR and the increase of bit rate are brought by the existing fast mode decision, not by our presented algorithm.

7. Conclusion

To promote intra prediction into a novel fine-grained parallelism stage, we present a novel parallel partition mechanism for it, named FIPIP based on GPUs. We find that the biggest obstacle for realizing fine-grained parallel intra prediction on GPU is the large number of logic branches brought by the diversity of prediction formulas and the constraints from H.264. To overcome the problem, in this paper, we propose three methods. First, transform irregular prediction formulas into a unified form. Second, introduce a parameter table and a predictor unit to switch diversified branches into irregular accesses to shared memory (the lookup of Table 3 and the unified predictor array in Fig. 4(b)). Third, a table lookup algorithm is proposed to ensure that threads can access irregular locations of predictors efficiently and perform unified intra prediction. Moreover, in order to build larger combined frame for coarse-grained parallelism, we propose two new encoding orders by taking the method of the improved 10-step encoding order and a combined frame strategy. Both of the fine-grained and coarse-grained methods are combined together to improve the performance of the intra prediction. An efficient self-synchronizing method is realized for the fine-grained task scheduling on heterogeneous CPU–GPU architecture. In addition to parallelization, some fast mode decision methods have been taken into account to accelerate intra prediction.

The experiment results on graphics cards GTX295, GTX460 and K20 demonstrated that the presented algorithm FIPIP has good performance. It outperforms the state-of-the-art works with a speedup of about 2–6 times, and has a speedup of about 6× to 14× compared to the JM18.2. In the future, we plan to probe some potential schemes for the parallelism of the intra prediction in P, and B frames (at present, the presented approach just works with I frame). Moreover, we will try to extend the proposed algorithm into the next generation standard of HEVC [22].

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant No. 61133008, and Scientific Research Foundation of Ministry of Education of China-China Mobile under Grant No. MCM20122041. We gratefully acknowledge the support of NVIDIA Corporation with the excellent GPUs used for this research.

References

[1] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Draft ITU-T recommendation and final draft international standard of joint video specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, March 2003.
[2] JM reference software. [Online]. Available: http://iphome.hhi.de/suehring/tml/.

[3] G. Sullivan, T. Wiegand, Rate-distortion optimization for video compression, IEEE Signal Process. Mag. 15 (6) (1998) 74–90.
[4] S. Milani, Fast H.264/AVC FRExt intra coding using belief propagation, IEEE Trans. Image Process. 20 (1) (2011) 121–131.
[5] F. Pan, X. Lin, S. Rahardja, K. Lim, Z. Li, D. Wu, S. Wu, Fast mode decision algorithm for intraprediction in H.264/AVC video coding, IEEE Trans. Circuits Syst. Video Technol. 15 (7) (2005) 813–822.
[6] A. Tsai, A. Paul, J.C. Wang, J.F. Wang, Intensity gradient technique for efficient intra-prediction in H.264/AVC, IEEE Trans. Circuits Syst. Video Technol. 18 (5) (2008) 694–698.
[7] N. Cheung, O. Au, M.C. Kung, P.W. Wong, C.H. Liu, Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors, IEEE Trans. Circuits Syst. Video Technol. 19 (11) (2009) 1692–1703.
[8] M. Kung, O. Au, P. Wong, C.H. Liu, Intra frame encoding using programmable graphics hardware, in: Proc. Pacific Rim Conf. Multimedia, PCM, 2007, pp. 609–618.
[9] H. Su, N. Wu, C. Zhang, M. Wen, J. Ren, A multilevel parallel intra coding for H.264/AVC based on CUDA, in: International Conference on Image and Graphics, 2011, pp. 76–81.
[10] J. Ren, Y. He, H. Su, M. Wen, N. Wu, C. Zhang, Parallel streaming intra prediction for full HD H.264 encoding, in: International Conference on Embedded and Multimedia Computing, 2010, pp. 1–6.
[11] W. Jiang, M. Long, H. Jin, P. Wang, Fine-grained CUDA-based parallel intra prediction for H. 264/AVC, in: Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV, 2014, pp. 97–102.
[12] N. Wu, M. Wen, W. Wu, J. Ren, H. Su, C. Xun, C. Zhang, Streaming HD H.264 encoder on programmable processors, in: Proceedings of the 17th ACM International Conference on Multimedia, 2009, pp. 371–380.
[13] L. Liu, Y. Chen, D. Wang, S. Yin, X. Wang, L. Wang, H. Lei, P. Cao, S. Wei, Implementation of multi-standard video decoder on a heterogeneous coarse-grained reconfigurable processor, Sci. China Inf. Sci. 57 (8) (2014) 1–14.
[14] C. Chen, P.H. Wu, C. Chen, Transform-domain intra prediction for H.264, in: IEEE International Symposium on Circuits and Systems, Vol. 2, 2005, pp. 1497–1500.
[15] C.W. Tien, H.Y. Lin, B.D. Liu, J.F. Yang, Transform-domain partial prediction algorithm for intra prediction in H.264/AVC, in: IEEE International Symposium on Circuits and Systems, 2009, pp. 1245–1248.
[16] C. An, T. Nguyen, Statistical learning based intra prediction in H.264, in: IEEE International Conference on Image Processing, ICIP, 2008, pp. 2800–2803.
[17] Y.W. Huang, B.Y. Hsieh, T.C. Chen, L.G. Chen, Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder, IEEE Trans. Circuits Syst. Video Technol. 15 (3) (2005) 378–401.
[18] H.Y. Lin, K.H. Wu, B.D. Liu, J.F. Yang, An efficient VLSI architecture for transform-based intra prediction in H.264/AVC, IEEE Trans. Circuits Syst. Video Technol. 20 (6) (2010) 894–906.
[19] T. Wiegand, G. Sullivan, G. Bjontegaard, A. Luthra, Overview of the H.264/AVC video coding standard, IEEE Trans. Circuits Syst. Video Technol. 13 (7) (2003) 560–576.
[20] CUDA toolkit documentation. [Online]. Available: http://docs.nvidia.com/cuda/index.html.
[21] Xiph.org video test media. [Online]. Available: http://media.xiph.org/video/derf/.
[22] G.J. Sullivan, J. Ohm, W.J. Han, T. Wiegand, Overview of the high efficiency video coding (HEVC) standard, IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1649–1668.

Wenbin Jiang is an Associate Professor at School of Computer Science and Technology of Huazhong University of Science and Technology (HUST). He received his Ph.D. degree from HUST in 2004. He was a visiting scholar at UCLA (University of California, Los Angeles (UCLA)) in 2014 for one year. His research interests include parallel computing, multimedia computing, machine learning and computer vision.

Min Long received his Master degree from Huazhong University of Science and Technology in 2014. He is a technician of Alibaba (China) Technology Co., Ltd. His research interests include distributed computing, video processing, and cloud computing.

Laurence T. Yang is a Professor at School of Computer Science and Technology of Huazhong University of Science and Technology and at St. Francis Xavier University. He graduated from Tsinghua University and got his Ph.D. degree from University of Victoria, Canada. His current research includes parallel and distributed computing, embedded and ubiquitous computing.

Xiaobai Liu is an Assistant Professor of Computer Science in the San Diego State University. He previously worked as a Postdoctoral Research Scholar at the University of California, Los Angeles. He received his Ph.D. degree from the Huazhong University of Science and Technology in 2012. His research interests include machine learning, multimedia computing, and computer vision.

Hai Jin is a Professor at School of Computer Science and Technology of Huazhong University of Science and Technology. He received his Ph.D. degree from Huazhong University of Science and Technology (HUST) in 1994. He was postdoctoral fellow at University of Southern California and The University of Hong Kong. His research interests include HPC, grid computing, cloud computing, and virtualization.

Alan L. Yuille holds a full-time position at Department of Statistics, University of California, Los Angeles. He also holds courtesy appointments at Department of Psychology, Department of Computer Science, Department of Psychiatry. His Ph.D. degree from the University of Cambridge, supervised by Prof. S.W. Hawking, was approved in 1981. His research interests include computational models of vision, mathematical models of cognition, and artificial intelligence and neural networks.

Ye Chi is a Master student at Huazhong University of Science and Technology. His research interests include distributed computing, video processing, and cloud computing.