Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2008, Article ID 542735, 19 pages
doi:10.1155/2008/542735

Research Article

Early Termination and Pipelining for Hardware Implementation of Fast H.264 Intraprediction Targeting Mobile HD Applications

Jin-Su Jung, Genhua Jin, and Hyuk-Jae Lee

School of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-742, South Korea

Correspondence should be addressed to Hyuk-Jae Lee, hjlee_paper@capp.snu.ac.kr

Received 2 March 2008; Revised 16 June 2008; Accepted 7 August 2008

Recommended by Liang-Gee Chen

H.264/AVC adopts aggressive compression algorithms at the cost of increased computational complexity. To speed up the H.264/AVC intraframe coding, this paper proposes two novel techniques: early termination and pipelined execution. In P slices, intra 4×4 and 16×16 predictions are early terminated with the threshold determined by the cost of motion estimation. In I slices, intra 4×4 prediction is early terminated with the threshold derived from intra 16×16 prediction. The threshold function is chosen as a monotonically decreasing linear function with its optimal coefficients determined by experiments. For the pipelined execution of 4×4 intrapredictions, the processing order of 4×4 blocks is changed to reduce the dependencies between consecutively processed blocks. In I slices, computation for 4×4 intraprediction is reduced by 19 percent with the proposed early termination. In P slices, computations for 4×4 and 16×16 intrapredictions are reduced by more than 81 and 91 percents, respectively. The pipelined execution reduces the computation time by 41 percent. In spite of the speed-up by the proposed methods, degradation in rate-distortion performance is negligible. The proposed pipelined execution is integrated with other H.264/AVC hardware accelerators and fabricated as an SoC using Dongbu 0.13 μm technology.

1. INTRODUCTION

The joint video team (JVT) of ISO/IEC MPEG and ITU-T VCEG recently proposed a new video compression standard known as H.264/AVC, advanced video coding (AVC). In order to improve the compression efficiency, H.264/AVC employs aggressive compression techniques such as variable block size and 1/4-pel accuracy motion compensation. Intraprediction is one of the aggressive techniques employed to improve the coding efficiency of H.264/AVC. In the baseline and main profiles, intraprediction is performed with two block sizes (a 4×4 block and a 16×16 block) for 13 prediction modes (9 modes for 4×4 blocks and 4 modes for 16×16 blocks), and the best intramode is selected among the 13 modes. As a result, intraprediction requires a large amount of computation, which is comparable to the computation amount of JPEG2000 [1]. Accordingly, there has been extensive research into a faster computation of intraprediction for the real-time processing of H.264/AVC video compression.

Extensive research efforts have been made to speedup the computation of intraprediction [1–8]. One of the most popular approaches reduces the computational complexity of intraprediction by evaluating the RD cost not for the entire set of prediction modes but only for a subset of them and selecting the best mode among the estimated modes. A number of previous studies take advantage of the fact that an intraprediction mode is strongly related to the edge, or texture, direction of the block. Therefore, this direction is evaluated first, and then the RD cost is estimated only for a subset of prediction modes according to the evaluated direction [2, 9–18]. In [1, 19], the best prediction mode is selected from candidate modes that are derived from the prediction modes of neighboring blocks. In these techniques, the compression efficiency is sacrificed because one of the excluded prediction modes may be the optimal mode. In another technique for fast intraprediction, one of the 4 × 4 or 16 × 16 intrapredictions is skipped [20–22]. The smoothness of a macroblock is estimated, and 16 × 16 intraprediction is chosen when the block is smooth while 4 × 4 intraprediction is performed in the other case. An early decision to skip intraprediction has been proposed by several researchers as another efficient method for fast intraprediction [3, 23–26]. In the reports by the researchers, the results of interprediction (i.e., motion estimation) is often used as the criterion for the decision. The elimination of both or either of the 4 × 4 and 16 × 16 intrapredictions reduces the compression efficiency because the skipped intraprediction may generate the optimal compression result. Another technique for fast intraprediction uses only a part of the pixels in a block when evaluating the prediction error. For example, 8 pixels, instead of 16 pixels, are used for the error evaluation of the 4 × 4 intraprediction [1]. This technique also suffers from a loss of compression efficiency when the error evaluation with the selected pixels leads to a suboptimal decision.

Intraprediction is often implemented by a hardware accelerator as its computational complexity is large. Not all of the algorithms explained above are suitable for hardware implementation which requires a relatively regular computation structure. Thus, a number of studies have been published on the algorithms and architecture for the hardware implementation of intrapredictions [1, 4, 27–29]. In [4], the DC components of 4 × 4 DCT coefficients are precalculated when 16 × 16 intraprediction is performed. This technique enables pipelining of transform and quantization (TQ) and inverse quantization and inverse transform (IQIT) of the results from 16 × 16 intraprediction. In [27, 28], an effective architecture is suggested for intraprediction and mode decision. For these hardware implementations, the hardware resources for 4 × 4 intraprediction and reconstruction are often idle and wasted. Intraprediction of a 4 × 4 block depends on the reconstructed pixels in its neighboring blocks, and therefore, intraprediction must remain idle while the reconstruction of the neighboring blocks is being completed. Hence, the execution of intraprediction and reconstruction is often serialized. In [1], the idle cycles are avoided by interleaving 4 × 4 intraprediction and 16 × 16 intraprediction. This architecture is pretty effective when a single prediction module is shared by both 4 × 4 and 16 × 16 predictions. A recently proposed architecture employs separate prediction modules to further speedup intraframe coding [29]. This architecture employs three-step mode decision algorithm to avoid the idle cycles at the sacrifice of a slight degradation of R-D performance.

This paper proposes two novel techniques to speedup the execution of intraprediction. The first technique is early termination of intraprediction. The proposed early termination performs intraprediction for each 4 × 4 block and accumulates the cost of intraprediction for the 4 × 4 block. At the end of every 4 × 4 block processing, the accumulated cost is compared with a predetermined threshold, and intraprediction is stopped when the cost is higher than the threshold. The cost of motion estimation is used to derive the threshold in a P slice. In an I slice, motion estimation is not performed, so that its cost is not available for the early termination. In this case, the results from 16 × 16 intraprediction are used as the threshold for the early termination of 4 × 4 intraprediction. An efficient threshold for early termination is investigated to obtain the best tradeoff between coding efficiency and computation reduction. The second technique is pipelined execution of intraprediction and reconstruction which minimizes the waste of hardware resources by serialized execution. To allow the pipelined execution, the processing order of 4 × 4 intrapredictions is rearranged so that neighboring blocks are not processed consecutively. An optimal processing order is derived to achieve the pipelining of intraprediction without a significant reduction of compression efficiency. Early termination combined with pipelined execution significantly reduces the computation time of intraprediction without much sacrifice of the coding efficiency. Experimental results show that the proposed early termination achieves about 19 percent of savings of computation for the 4 × 4 intraprediction in I slices and over 81 percent savings for the 4 × 4 and 16 × 16 intrapredictions in P slices. The pipelined execution also reduces the computation time of the 4 × 4 intraprediction by about 41 percent. In spite of this large computation savings by early termination and pipelining, the loss of compression efficiency is negligible. The proposed pipelined architecture is integrated into an H.264/AVC encoder and fabricated as an SOC using the Dongbu 0.13 μm technology.

The rest of this paper is organized as follows. Section 2 proposes an early termination algorithm for intraprediction. Section 3 proposes a simple pipelined execution of the 4 × 4 Luma prediction with the original H.264/AVC order and explains the reduction of the compression efficiency resulting from the pipelined execution. Section 4 proposes a new processing order of intraprediction that improves the pipelining efficiency. Evaluation results are shown in Section 5, and the hardware implementation of the pipelined architecture is presented in Section 6. Section 7 presents conclusions.

2. VARIABLE THRESHOLD EARLY TERMINATION

2.1. Early termination of 4 × 4 intra prediction

In this section, an early termination scheme for intraprediction is proposed to reduce the amount of computation for intraprediction while minimizing the loss of compression efficiency. The early termination algorithm for 4 × 4 intraprediction is shown in Figure 1. The first step is the predictor generation step in which the neighboring blocks of a 4 × 4 block are used to generate the predictors. The next step calculates the SATD that is the sum of the differences between the predictors and the original pixels in the Hadamard transformed domain. Then, the cost of 4 × 4 intraprediction is derived from SATD as follows:

$$\text{Cost}_{\text{intra\_4}\times 4} = (K + 4M) \times \lambda(\text{QP}) + \text{SATD}, \tag{1}$$

where K is a constant to be determined and M is the number of 4 × 4 blocks of which the best mode is not the most probable mode which is derived from the best modes of the

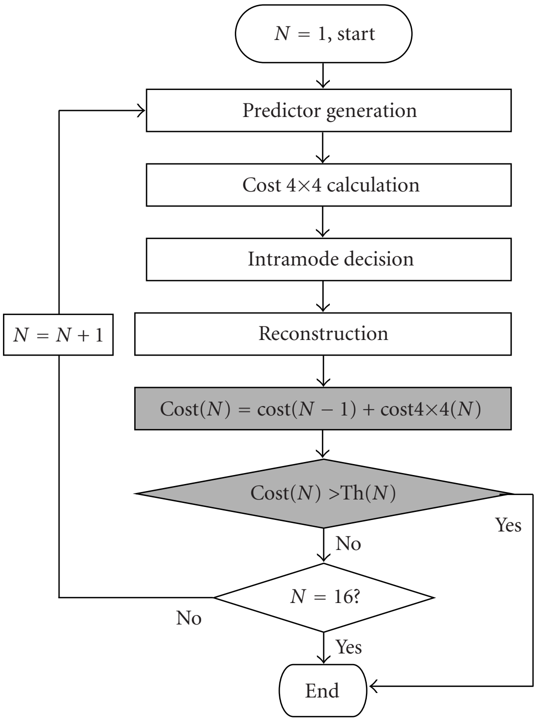


Figure 1: Early termination of 4×4 intraprediction.

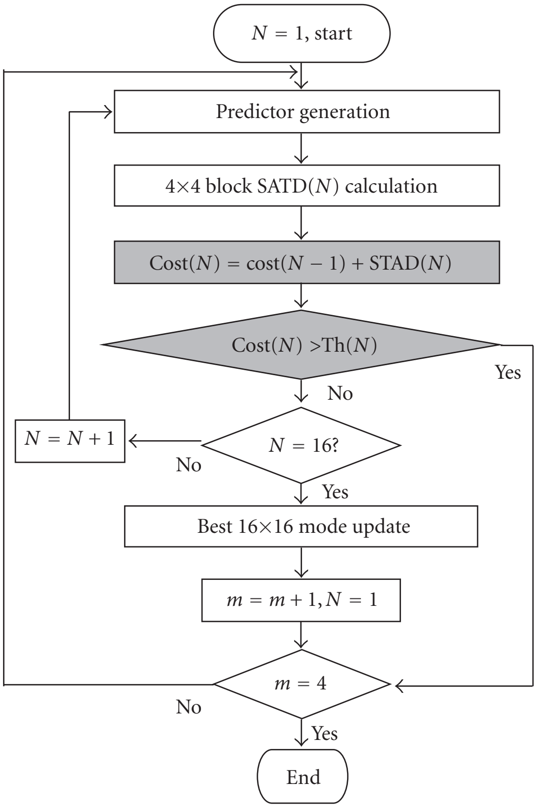


Figure 2: Early termination of 16×16 intraprediction.

neighboring 4×4 blocks [30]. $\lambda(\mathrm{QP})$ is a function of the quantization parameter, QP. This cost is the Lagrangian cost defined in the H.264/AVC standard in which the constant K is defined as 24. For the derivation of $\mathrm{Cost}_{\mathrm{intra\_4\times4}}$, SAD, instead of SATD, can also be used. The use of SAD instead of SATD reduces the computation time because Hadamard transform needs not to be performed. However, the accuracy of the cost estimation by SAD is slightly degraded.

The cost of a single 4×4 block for intra 4×4 prediction, denoted by $\mathrm{Cost}_{4\times4}$, is derived from (1) as follows:

$$\mathrm{Cost}_{4\times4} = \left(\frac{K}{16} + 4M'\right) \times \lambda(\mathrm{QP}) + \mathrm{SATD}, \qquad (2)$$

where K and $\lambda(\mathrm{QP})$ are same as (1) and M' is 1 when the selected prediction mode is not the most probable mode derived from the neighboring 4×4 blocks [30] and M' is 0 otherwise. SATD in (2) represents the summation over a single 4×4 block (i.e., the 16 differences between the predictors and the original pixels in the Hadamard transformed domain) while SATD in (1) represents the summation over a 16×16 macroblock. The $\mathrm{Cost}_{4\times4}$ is derived for 9 prediction modes; then, the next step, intramode decision, selects the best mode among the nine possible modes. In the next step, the 4×4 block is reconstructed and saved as a reference frame. This reconstructed block is also used to generate the predictors for its neighboring blocks. After the reconstruction, the accumulated cost, $\mathrm{cost}(N)$, is calculated by adding $\mathrm{cost}_{4\times4}(N)$ to $\mathrm{cost}(N-1)$ where $\mathrm{cost}_{4\times4}(N)$ represents the cost of the Nth 4×4 block (i.e., the current block), and $\mathrm{cost}(N-1)$ is the accumulated cost from the 1st to the $(N-1)$th blocks. The next step determines whether intraprediction is terminated early or not. If $\mathrm{cost}(N)$ is larger than the threshold function, $\mathrm{Th}(N)$, then intraprediction is terminated early. Otherwise, intraprediction proceeds to the next 4×4 block. For an efficient early termination without a degradation of compression efficiency, a proper selection of the threshold function is important, and Section 2.3 discusses in detail the derivation of the threshold function. The above steps are repeated until intraprediction is terminated early or all 16 blocks are processed.

For P slices, the threshold function is derived from the result of motion estimation (see Section 2.3 for details). For an I slice, motion estimation is not performed, and consequently, the result of motion estimation is not available for the threshold function. Instead, the result of 16×16 intraprediction is used to obtain the threshold function for early termination of 4×4 intraprediction (see details also in Section 2.2). It is also possible to early terminate 16×16 intraprediction using the result of 4×4 intraprediction. However, the early termination of 4×4 intraprediction is more efficient than that of 16×16 intraprediction because the amount of computation of 4×4 intraprediction is much larger than that of 16×16 intraprediction.

2.2. *Early termination of 16 × 16 intra prediction*

The early termination scheme is also employed for 16×16 intraprediction in P slices with its threshold derived from the cost of motion estimation. A 16×16 macroblock is decomposed into sixteen 4 × 4 blocks, and the SATD of each 4 × 4 block is added up and compared with a predefined threshold. If this accumulated cost is greater than the threshold, the 16 × 16 intraprediction is early terminated. Figure 2 shows the early termination algorithm for 16 × 16 intraprediction. Predictor generation is performed first, and then, a macroblock is decomposed into sixteen 4 × 4 blocks. The SATD is calculated for one 4 × 4 block after another and defined as the cost of the Nth 4 × 4 block:

$$\mathrm{Cost}(N) = \mathrm{SATD}(N). \tag{3}$$

Then, the SATD of the Nth block (i.e., the current block) is added to cost$(N - 1)$, the accumulated cost from the 1st to the $(N - 1)$th block. The SATD is simply chosen as the cost of each 4 × 4 block because H.264/AVC standard defines the SATD of a 16 × 16 block with a slight adjustment as the cost of the block for 16 × 16 intraprediction. Note that the cost defined in the H.264/AVC standard is slightly different from SATD because the DC coefficients of the sixteen 4 × 4 blocks in the transformed domain are transformed again by the Hadamard transform [30]. Intraprediction is early terminated if the accumulated cost, cost(N), is larger than the threshold. Otherwise, the algorithm iterates the SATD calculation and comparison for early termination until all sixteen 4 × 4 blocks are processed. Unlike 4 × 4 intraprediction, 16 × 16 intraprediction determines the mode only after all sixteen 4 × 4 blocks are processed. After deriving the cost of the sixteen 4 × 4 blocks for one mode if intraprediction for the mode is not terminated early, the cost is compared to that of the other modes that are not early terminated. Then, the mode with the minimum cost is selected as the best 16 × 16 mode.

In I slices, 16 × 16 intraprediction is not terminated early because a proper threshold cannot be obtained.

2.3. Variable threshold

The threshold function, Th(N), determines the amount of computation savings by early termination. The selection of Th(N) is important for an effective tradeoff between computation savings and compression efficiency. In a P slice, the threshold is derived from the Lagrangian cost of the motion estimation of the corresponding macroblock as follows:

$$\mathrm{Th}(N) = \frac{\mathrm{Cost}_{\mathrm{inter}}}{16} \times (N + M(N)), \tag{4}$$

where $\mathrm{Cost}_{\mathrm{inter}}$ represents the Lagrangian cost of interframe prediction (i.e., motion estimation) defined in the H.264/AVC standard,and $M(N)$ is a margin for considering the cost variation of the remaining 4 × 4 blocks. N denotes the number of the 4 × 4 blocks including the current block for which intraprediction has been finished.

M is defined as a function of N because the cost variation is proportional to the number of the remaining 4 × 4 blocks. As the cost variation decreases with decreasing number of the remaining blocks, $M(N)$ should be a monotonically decreasing function of N. Thus, a simple linear function is chosen as follows:

$$M(N) = \frac{M(1)}{15} \times (16 - N), \tag{5}$$

where N varies from 1 to 16. Equation (5) is finally chosen as the margin function employed for the early termination proposed in this paper. As a margin is not necessary when all 16 blocks are processed, $M(N)$ is defined to be 0 for $N = 16$, that is, $M(16) = 0$. The optimal $M(1)$ is chosen experimentally as explained in Section 5.

In (4), $\mathrm{Cost}_{\mathrm{inter}}/16 \times N$ implies the amount of temporal correlation for the N 4 × 4 blocks while Cost(N) in Figure 1 represents the spatial correlation for the corresponding N 4 × 4 blocks. Therefore, the comparison of cost(N) with Th(N) implies the comparison of the spatial correlation with the temporal correlation (with a certain margin) of the N 4 × 4 blocks. Using the threshold function of (4), intraprediction is early terminated when the accumulated $\mathrm{Cost}_{4\times4}(N)$ is larger than Th(N) (i.e., the spatial correlation of the N blocks is smaller than the temporal correlation of the corresponding N blocks).

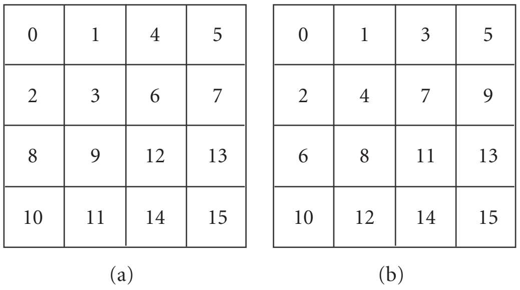
As the threshold is derived from the result of motion estimation, the above early termination assumes that motion estimation is processed before intraprediction. This assumption is true, in general, for the software implementation of H.264/AVC [30]. In some hardware implementations, this assumption is also true [31–33] while another type of possible hardware implementation executes motion estimation and intraprediction in parallel [31]. Even in this type of hardware implementation, motion estimation is, in general, decomposed into integer and fractional motion estimations, and integer motion estimation is performed earlier than fractional motion estimation. In this case, the result of integer motion estimation is available before the execution of intraprediction so that it can be used as the threshold for intraprediction. Although the result of integer motion estimation is not exactly the same as that of the fractional motion estimation, it can still be used effectively for the early termination criterion of intraprediction. The experimental results with both integer and fractional motion estimations are presented in Section 5.

In I slices, 4 × 4 intraprediction is early terminated based on the results of 16 × 16 intraprediction. In this case, the threshold function for early termination of 4 × 4 intra prediction is

$$\mathrm{Th}(N) = \frac{\mathrm{Cost}_{\mathrm{intra\_16\times16}}}{16} \times (N + M(N)), \tag{6}$$

where $\mathrm{Cost}_{\mathrm{intra\_16\times16}}$ represents the cost of 16 × 16 intraprediction, and $M(N)$ is the same as (5). Note that this threshold function is the same as that for P slices as given by (4).

For Chroma intraprediction, its execution is determined by the early termination of 4 × 4 and 16 × 16 Luma intrapredictions. If the proposed early termination is implemented in software, Chroma intraprediction is performed later than both Luma 4 × 4 and 16 × 16 predictions. If both Luma intrapredictions are early terminated, Chroma



0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

(a)

0	1	3	5
2	4	7	9
6	8	11	13
10	12	14	15

(b)

Figure 3: The processing order of 4 × 4 intraprediction. (a) Original order, (b) optimal order minimizing the cost of (7).

prediction is skipped. Otherwise, Chroma prediction is performed for all blocks. In the hardware implementation of this paper (see Figure 10(a)), Chroma intraprediction is performed after 16 × 16 Luma intraprediction and in parallel with the later part of 4 × 4 Luma intrapredictions. If both Luma intrapredictions are early terminated before the time when Chroma intraprediction begins, then Chroma prediction is skipped. If both Luma intrapredictions are early terminated while Chroma intraprediction is performed, then Chroma prediction is also early terminated at the same time. Otherwise, Chroma intraprediction is performed to the end.

3. PIPELINED EXECUTION OF INTRAPREDICTION AND RECONSTRUCTION FOR 4 × 4 LUMA BLOCKS

Figure 3(a) shows the processing order of the 4 × 4 intraprediction employed by the reference software of H.264/AVC [30]. In Figure 3, each box represents a 4 × 4 block of pixel data, and the number inside a box is the processing orders of these 16 4 × 4 blocks. For example, the block in the upper-left corner (labeled 0) is processed first, and the next block to the right (labeled 1) is processed next. The intraprediction of block 1 uses the rightmost four pixels of block 0 as the left-side predictors. Note that these predictors should be given in the reconstructed frame. Therefore, block 1 can be processed after block 0 is reconstructed. To reconstruct block 0, additional operations such as integer transform (T), quantization (Q), inverse quantization (Q^{-1}), and inverse integer transform (T^{-1}) should be performed after intraprediction. This implies that the intraprediction of block 1 can start only after all the operations for the reconstruction of block 0 are finished.

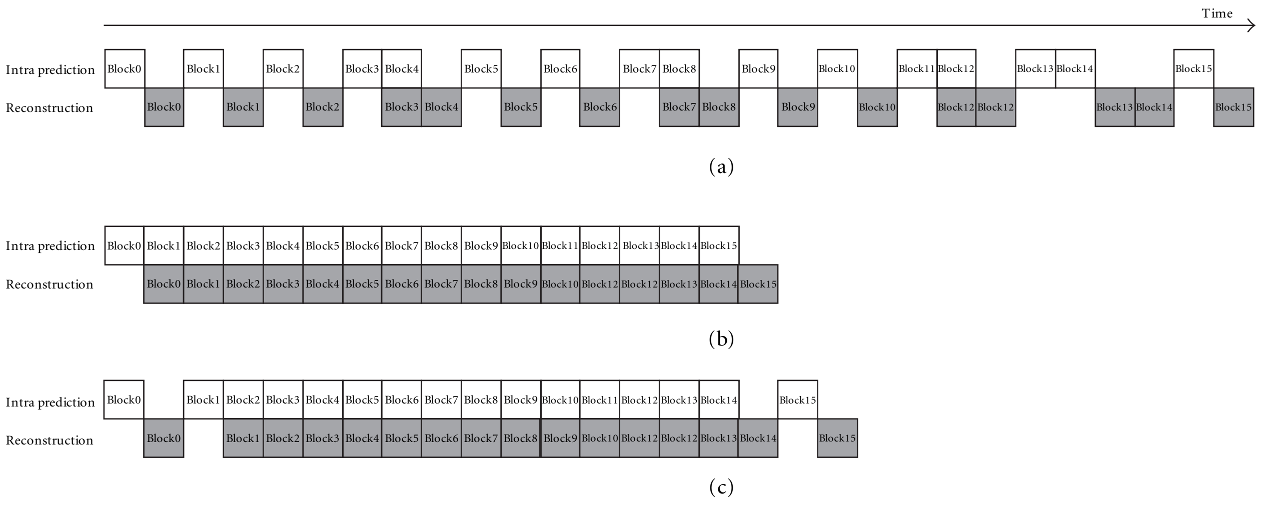
The execution order as shown in Figure 3(a) does not efficiently use hardware resources because all operations are serialized and only one hardware module among those designed for intraprediction, T, Q, Q^{-1}, and T^{-1} is utilized at a given time. To achieve a higher utilization, a pipelined execution of these hardware modules is desirable. For example, consider the intraprediction of block 4. Note that block 4 depends on block 1 but does not depend on the previously computed block 3. Therefore, block 4 can be pipelined with block 3. Among sixteen blocks, twelve blocks are dependent on their previously computed blocks in the order given in Figure 3(a). These blocks are blocks 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, and 15; operations for these blocks must be serialized. For the remaining blocks 4, 8, and 12, intrapredictions can be pipelined. Figure 4(a) shows the execution sequence of the sixteen 4 × 4 blocks where blocks 4, 8, and 12 are pipelined with their previous blocks. In this figure, "reconstruction" represents the operations of T, Q, Q^{-1}, and T^{-1}, and it is assumed that the execution time of "reconstruction" is the same as that of intraprediction. The numbers inside the boxes are designations for the 4 × 4 blocks. Let T_{ip} denote the execution time of intraprediction. Then, the total execution time of the sixteen blocks is $29 \times T_{ip}$ cycles.

The execution speed of intraprediction can be increased with the sacrifice of compression efficiency by excluding the support of certain intraprediction modes. For example, consider the intraprediction of block 1 shown in Figure 3(a). The reconstructed pixels of block 0 are necessary to process six prediction modes: the Horizontal mode, the DC mode, the diagonal down-right mode, the vertical-right mode, the horizontal-down mode, and the horizontal-up mode. If these six modes are excluded, block 1 can be processed before the reconstruction of block 0 and therefore can be pipelined with block 0 in the same way as block 4. However, the exclusion of these six modes may reduce the compression efficiency. Therefore, this pipelining results in the tradeoff of computation speed with compression efficiency. Similar to block 1, the other blocks except blocks 0, 4, 8, and 12 can be pipelined at the sacrifice of compression efficiency. Figure 4(b) shows a fully pipelined execution sequence of the sixteen blocks. In this case, the execution time is reduced to $17 \times T_{ip}$ cycles. However, compression efficiency is severely reduced due to the exclusions of many modes. The simulation results evaluating the reduction of compression efficiency due to the pipelined execution is shown in Section 5.

4. OPTIMAL PROCESSING ORDER AND PARTIALLY PIPELINED EXECUTION

4.1. The optimal order for encoding intra 4 × 4 blocks

In order to improve the compression efficiency even with the pipelined execution, this paper proposes a change in the processing order of 4 × 4 intraprediction. One such change for optimal processing order is given in Figure 3(b) (the derivation of this order is to be discussed later). The proposed ordering has two objectives. The first objective is to separate the processing of dependent blocks so that consecutively processed blocks are independent and executed in a pipelined manner. For an example, in the proposed order, block 3 does not depend on its previous block, block 2, so that block 3 does not need to wait for block 2 to be reconstructed. In this new order, 3, 4, 5, 6, 7, 9, 10, 11, 12, and 13 are independent of their previously processed blocks and can be pipelined without the reduction of the compression efficiency. Note that blocks 4 and 12 are processed before their upper-right blocks, blocks 3 and 11, respectively. However, these two blocks do not use the upper-right blocks in the original H.264/AVC reference software,



(a)

(b)

(c)

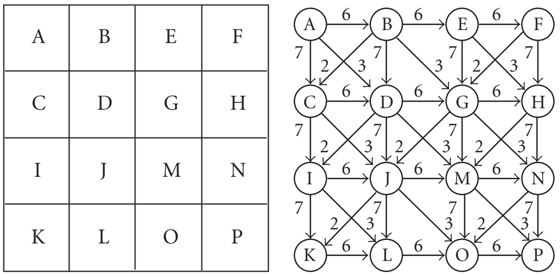
Figure 4: Difference in processing time by pipelined execution of 4 × 4 intra prediction. (a) Partially pipelined execution of 16 4 × 4 blocks in the original processing order. (b) Fully pipelined execution. (c) Partially pipelined execution in the proposed order shown in Figure 3(b).

either. Therefore, these two blocks are treated independently of their upper-right blocks. Hence, the new order has seven more independent blocks than that in the original order shown in Figure 3(a).

The second objective of the proposed ordering is to select the excluded modes which affect the compression efficiency to a lesser degree and to reduce the number of excluded modes if the exclusion of certain modes is unavoidable. Consider block 8 in Figure 3(b). Block 8 depends on block 7 because the pixels of block 7 are used as predictors of block 8 for the diagonal down-left and vertical-left modes. If the intraprediction of block 8 in this order cannot perform the two modes that are dependent on block 7, the exclusion may reduce the compression efficiency. However, its impact is less significant than the exclusion of other important modes that depend on either the left block or the upper block. When the left block is not available, six modes should be excluded and these six modes, in general, impact more significantly than the two modes that depend on the upper-right block. Seven modes are dependent on the upper block, so that the block gives an even more significant impact than the left block. Thus, the exclusion of the two modes that depend on the upper-right block may affect the compression efficiency least when compared to the exclusion of the left block or the upper block.

To derive the processing order that satisfies the two goals discussed above, an optimization problem is formulated. To this end, a dependency graph is defined as shown in Figure 5(b). The sixteen 4 × 4 blocks labeled from A to P in Figure 5(a) are represented by the sixteen labeled nodes in Figure 5(b). An edge between two nodes represents the dependence between the source and destination nodes of the edge. The weight of an edge represents the number of intraprediction modes that depend on the source node. For example, the edge weight of 6 from node A and B represents the number of intraprediction modes that require the predictors from A. Note that these six modes are the horizontal, DC, diagonal down-right, vertical-right, horizontal-down, and the horizontal-up modes. Any block depends on its left blocks, and the number of dependent modes is 6. Thus, the weight of any edge from a node to its right node is always 6 in this graph. On the other hand, any node depends on its upper node for 7 modes (i.e., vertical, DC, diagonal down-left, diagonal down-right, vertical-right, horizontal-down, and vertical-left modes). Thus, the weight of any edge from a node to its lower node is assigned to 7. Similarly, any node depends on its upper-left node for 3 modes (i.e., diagonal down-right, vertical-right, and horizontal-down modes), and the weight of an edge from a node to its lower-right node is assigned to 3. All of these weights are assigned to the edges in Figure 5(b). A block depends on its upper-right block for 2 modes (i.e., diagonal down-right and vertical-left modes). Thus, a weight of 2 is assigned to the edge from a node to its lower-left node. Note that there is no edge from E to D although E is the upper-right block of D. This edge is absent because block D is processed earlier than block E even in the original processing order (as shown in Figure 3(a)), so that block D does not use its predictors from block E for the diagonal down-right and vertical-left modes even in the original H.264/AVC reference software. For the predictors of these two modes, the bottom-rightmost pixel of block B is copied to the right to become the predictors. The edge from M to L is also absent for the same reason as from E to D. Other than these two edges, all edges from nodes to its lower-left nodes have their weights assigned to 2.

The processing order is defined as a one-to-one function, order(), that maps an integer (from 0 to 15) to a 4 × 4 block. For example, suppose that function order() is defined such as order(0) = A, order(1) = C, and order(2) = B. Then, the processing order of the first three blocks is A, C, followed by B. The objective of the order selection is to minimize the excluded prediction modes. Consider the case when the *i*th block (i.e., the block processed in the *i*th order, also denoted



(a) Sixteen 4 × 4 blocks (b) Dependence graph

Figure 5: The dependency graph for cost formulation.

by order(i)) is dependent on the jth block and the ith block is processed earlier than the jth block, that is, $i < j$. In this case, by the time the ith block is processed, its predictors that belong to the jth block are not available. Thus, among the intraprediction modes of the ith block, the modes that depend on the jth block must be excluded. The number of excluded modes is represented by weight(order(j), order(i)) which is the weight of the edge from node j to node i. All the blocks on which the ith block depends need to be processed earlier than the ith block. Otherwise, the prediction modes of the ith block must be excluded. Let pred(order(i)) denote the set of blocks on which the ith block depends. Then, the number of the excluded prediction modes of the ith block is expressed as

$$\Sigma_{\text{order}(j)\in\text{pred}(\text{order}(i)),\ i<j}\ \text{weight}(\text{order}(j), \text{order}(i)). \tag{7}$$

Consider another case in which the ith block is dependent on the $(i-1)$th block. For the intraprediction of the ith block, T-Q-Q^{-1}-T^{-1} operations for the $(i-1)$th block are not completed. In this case, the number of excluded modes is represented by

$$\text{weight}(\text{order}(i-1), \text{order}(i)). \tag{8}$$

This is because the predictors that belong to the $(i-1)$th block are not available, by the time the ith block is processed. From (7) and (8), the total number of the excluded modes is given as follows:

$$\begin{aligned}\text{cost(order)} = {} & \Sigma_{(i=0\ \text{to}\ 14)}\Sigma_{\text{order}(j)\in\text{pred}(\text{order}(i)),\ i<j}\\ & \text{weight}(\text{order}(j), \text{order}(i))\\ & + \Sigma_{(i=1\ \text{to}\ 15)}\ \text{weight}(\text{order}(i-1), \text{order}(i)).\end{aligned} \tag{9}$$

The cost function of (9) is design to estimate the number of excluded modes when all blocks are processed in a pipelined manner. Note that this cost may not always be a precise estimation of the number of excluded modes because some excluded modes may be counted twice in (9). For example, suppose that the blocks shown in Figure 5(a) are processed in the order such that order(0) = C, order(1) = A, and order(2) = B. Then, block C depends on both blocks A and B, but C is processed earlier than blocks A and B. Thus, cost(order) includes the weight(order(1), order(0)) (= weight(A,C)) and weight(order(2), order(0)) (= weight(B,C)). Note that the 7 excluded modes in weight(A,C) include the 2 excluded modes in weight(B, C). Therefore, the two excluded modes are counted twice. This example shows a possibility that the cost of (9) may not estimate the exact number of excluded modes because certain modes can be counted multiple times. However, such a case does not occur unless a node is processed earlier than the node that it depends on. For both the original order and the optimal order in Figure 3, the cost estimated by (9) does not have any excluded modes counted twice. Thus, the cost function of (9) is a reasonable approximation of the number of excluded modes, and in most cases, it is the exact number of excluded modes.

With an exhaustive search, the optimal solution that minimizes the cost function of (9) is derived as shown in Figure 3(b). In this optimal order, the value of the cost function is 18. Note that the cost of the original H.264/AVC order is 56, and the improvement following the proposed order is over 68%.

In the above discussion, it is assumed that the processing time of intraprediction is greater than or equal to that of T-Q-Q^{-1}-T^{-1} operations. This assumption is true in the implementation in this paper as discussed in Section 6, and consequently the hardware implementation in this paper follows the processing order given by Figure 3(b). However, this assumption may not always be valid for other implementations. The cost function of (9) can be easily extended for a general case. For example, consider another case in which the processing time of intraprediction is less than that of T-Q-Q^{-1}-T^{-1} operations. In this case, if the ith block depends on the $(i-2)$th block, the prediction modes that depend on the $(i-2)$th block must be excluded. This is because the predictors that depend on the $(i-2)$th block are not available, by the time the ith block starts intraprediction. Therefore, the term weight(order($i-2$), order(i)) needs to be added to the cost function. As a result, the cost function of (9) is modified as follows:

$$\begin{aligned}\text{cost(order)} = {} & \Sigma_{(i=0\ \text{to}\ 14)}\Sigma_{\text{order}(j)\in\text{pred}(\text{order}(i)),\ i<j}\\ & \text{weight}(\text{order}(j), \text{order}(i))\\ & + \Sigma_{(i=1\ \text{to}\ 15)}\text{weight}(\text{order}(i-1), \text{order}(i))\\ & + \Sigma_{(i=2\ \text{to}\ 15)}\text{weight}(\text{order}(i-2), \text{order}(i)).\end{aligned} \tag{10}$$

If the processing time of T-Q-Q^{-1}-T^{-1} operations is larger than twice the execution time of intra prediction, then the $(i-3)$th block is not available, by the time the ith block is processed. In this case, the prediction modes that depend on the $(i-3)$th block also needs to be excluded for the intraprediction of the ith block. Thus, according to the processing times of T-Q-Q^{-1}-T^{-1} operations and intraprediction, the number of excluded modes is determined.

Although the number of excluded modes is significantly reduced by the optimal order of Figure 3(b), the pipelined execution of intraprediction and reconstruction (T-Q-Q^{-1}-T^{-1} operations) may still suffer from a substantial loss of compression efficiency. This is because a large number of modes are excluded in the intrapredictions of block 1 and

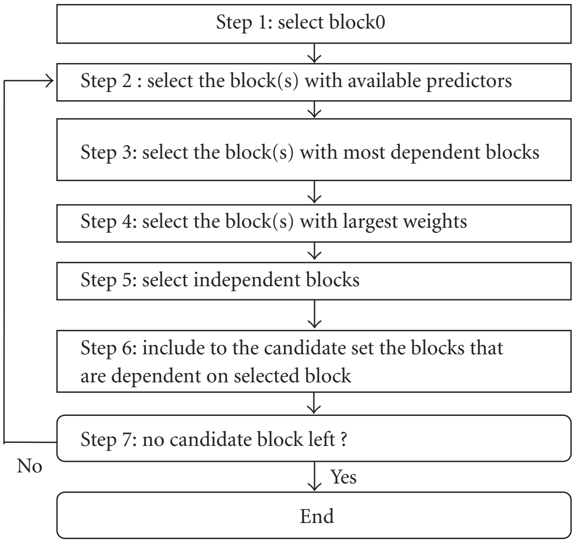

Step 1: select block0
Step 2 : select the block(s) with available predictors
Step 3: select the block(s) with most dependent blocks
Step 4: select the block(s) with largest weights
Step 5: select independent blocks
Step 6: include to the candidate set the blocks that are dependent on selected block
Step 7: no candidate block left ?
No
Yes
End

FIGURE 6: Processing order selection algorithm.

block 15. Other than these two blocks, at most two modes are excluded, so that the loss of compression efficiency is minor. For a reasonable tradeoff between the compression efficiency and the execution time, this paper proposes a partial pipeline such that all blocks are executed in a pipelined manner except for block 1 and block 15, which are executed only after their previous blocks are reconstructed. Figure 4(c) shows the execution sequence of the partial pipeline. As two blocks are not pipelined, the total execution time is $19 \times T_{ip}$ cycles. Thus, the processing time increases by $2 \times T_{ip}$ cycles (11.7%) when compared to a fully pipelined execution (see Figure 4(b)). However, the compression efficiency is significantly improved as block 0 and block 15 do not exclude any prediction modes. The evaluation results of the compression efficiency for this partially pipelined execution are presented in Section 5.

4.2. The optimal order for decoding intra 4×4 blocks

This subsection focuses on decoding and discusses the processing order of intraprediction. The discussion in this subsection is made with the assumption that all data in a 16×16 macroblock are available before the start of intraprediction. If all data are not available, intraprediction must follow the decoding order of CAVLD, so that the processing order cannot be changed arbitrarily. Like encoding, the intraprediction of a 4×4 block can start only after the block containing the predictors is reconstructed. To allow a pipelined execution for decoding, two back-to-back blocks must be independent. A decoder needs to process only one prediction mode while an encoder needs to process all modes to select the best one. Thus, a predefined order is not possible for a decoder. Instead, the algorithm shown in Figure 6 is proposed for selecting the processing order.

In the algorithm in Figure 6, step 1 makes block 0 an element of the set of candidate blocks for the next processing. Then, the algorithm moves to the next step. Step 2 selects the block whose predictors are available, that is, the blocks that contain the predictors are already processed and reconstructed. If more than one block is selected in this step, then the algorithm moves to the next step. Otherwise, the next step becomes step 6. Among the blocks selected in step 2, step 3 selects the block that has the most dependent blocks that are not yet processed. Recall that one block is dependent on the other block if its predictors belong to the other block. Thus, for a given block, its dependent blocks are the one in the right, down, down-left, and down-right if they exist. Thus, one block has at most four dependent blocks. For example, block 2 in Figure 7 has four dependent blocks, 1, 4, 5, and B. while block 1 has only three dependent blocks, 3, 4, and C. Note that the down-left block of block 1 does not exist. If more than one block is selected, then the algorithm moves to the next step. Otherwise, the next step becomes Step 6. For the blocks selected in step 3, step 4 calculates the summation of all the weights for these dependent blocks. Then, step 4 chooses the one with the maximum weight. The weight is the same as that given in the corresponding edge of a dependence graph shown in Figure 5(b). For example, the weight of the edge from B to C in Figure 7 is 2 which corresponds to the edge weight from G to J in the dependence graph of Figure 5(b). For another example, the weight of the edge from B to E in Figure 7 is 6 which corresponds to the edge weight from G to H in Figure 5(b). If more than one block has the same maximum weight, then the algorithm moves to the next step. Otherwise, the next step becomes step 6. Among the blocks selected in step 4, step 5 selects the block that is not dependent on the other blocks selected in step 4. If more than one block is selected, choose any block among them. Then, the algorithm moves to the next step. Step 6 makes the neighboring blocks of the selected block as elements of the set of candidate blocks for the next processing. Then, the algorithm moves to the next step. In step 7, if there is a candidate block, the algorithm moves to step 2. Otherwise, the algorithm ends.

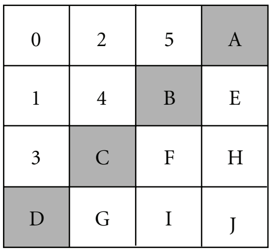


FIGURE 7: A processing order example for decoding.

The proposed algorithm is explained with Figure 7 in which six blocks numbered 0 to 5 are already processed in the order given by the numbers, and the remaining blocks are not yet processed. The basic idea of the proposed algorithm is that a block having more dependent blocks is selected earlier than the block with less dependent blocks. In this way, the algorithm has more choices in the selection of the blocks to be processed in the next steps. Consider an example that one block between A and B needs to be selected. In this example, block A has two dependent blocks, B and E, while block B has

Table 1: Bitrate and PSNR comparison of various pipelined executions.

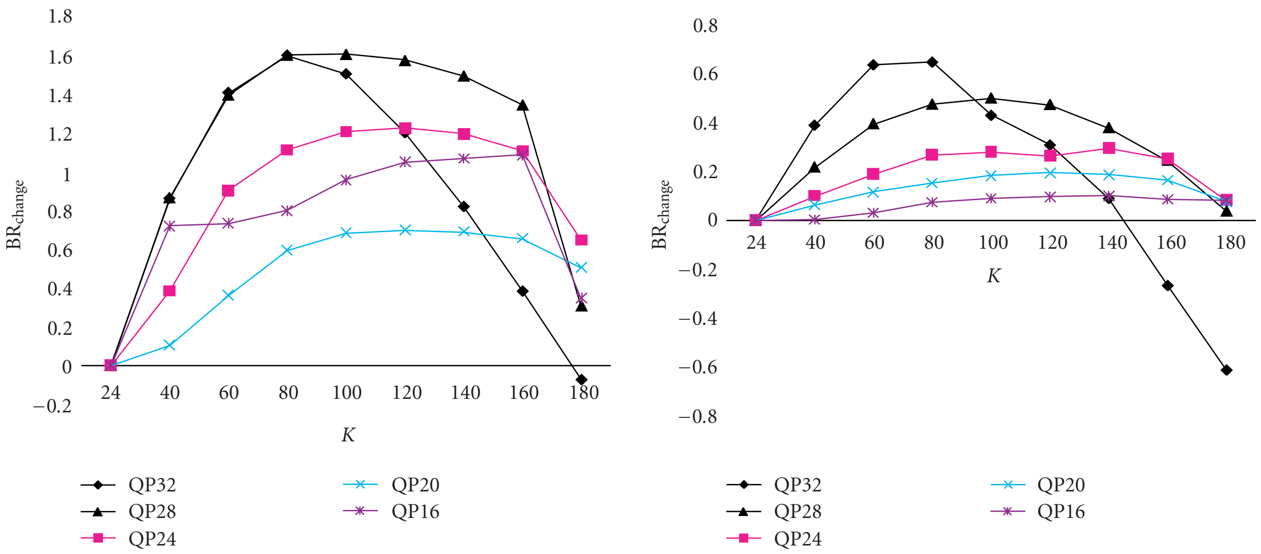
Sequence	QP	Original H.264 without pipeline	Original order + Full pipeline	Proposed order + Full pipeline	Proposed order + Partial pipeline	Original H.264 without pipeline	Original order + Full pipeline	Proposed order + Full pipeline	Proposed order + Partial pipeline
		Bit rate				PSNR			
Akiyo	16	4268.21	4585.53	4402.53	4297.93	47.36	47.3	47.34	47.36
	20	2872.88	3138.17	2995.07	2908.32	44.84	44.83	44.83	44.84
	24	2045.43	2242.07	2138.39	2067.89	42.53	42.53	42.52	42.55
	28	1439.71	1584.1	1496.24	1459.21	40.17	40.16	40.2	40.16
Mother and daughter	16	4413.6	4789.94	4569.89	4452.68	47.2	47.16	47.18	47.2
	20	2940.71	3248.39	3073.19	2975.96	44.38	44.34	44.36	44.37
	24	1980.31	2217.13	2088.11	2012.35	41.88	41.83	41.86	41.87
	28	1327.41	1517.17	1406.12	1353.44	39.48	39.41	39.45	39.46
Stefan	16	10592.21	11278.49	10874.35	10604.33	46.39	46.36	46.38	46.39
	20	8206.43	8781.96	8450.64	8217.89	42.96	42.95	42.96	42.96
	24	6264.22	6750.28	6467.23	6275.59	39.6	39.59	39.6	39.6
	28	4652.08	5059.1	4814.59	4662.62	36.36	36.36	36.37	36.37
Foreman	16	7741.97	8319.95	7963.94	7755	46.24	46.21	46.23	46.23
	20	5474.14	5997.32	5677.29	5486.95	42.86	42.83	42.85	42.86
	24	3744.46	4215.47	3928.1	3756.52	39.89	39.83	39.87	39.89
	28	2528.31	2918.27	2684.19	2538.77	37.19	37.09	37.16	37.19

four dependent blocks C, E, F, and H. Thus, block B has more dependent blocks than A so that it is selected earlier than block A. As block B is selected, four dependent blocks can be selected in the next steps. On the other hand, if block A is selected instead, only two dependent blocks can be selected in the next steps. Thus, the selection of block B gives more choices for the selection of blocks to be processed in the next steps.

Suppose that the algorithm is at step 2 attempting to find the next block to be processed. The candidate blocks are the neighboring blocks of the processed blocks, that is, $\{A, B, C, D\}$ is the set of the current candidate blocks. Assume that the intraprediction modes of blocks A, B, C, and D are 1 (horizontal mode), 1, 4 (diagonal down-right mode), and 0 (vertical mode), respectively. Step 2 checks blocks A, B, C, and D to see if their predictors are available. For block A, its prediction mode is 1 that requires the predictors from block 5. Note that block 5 is processed in the previous step so that its data is not available. Therefore, these predictors are not available, so that block A is not selected. For block B, its prediction mode is mode 1 (horizontal mode) that requires the predictors from block 4. Note that block 4 is not the previously processed block so that it is already reconstructed and all data are available for block B. Thus, block B is selected in step 2. Similarly, blocks C and D are also selected because their predictors are available. In step 3, the number of dependent blocks is counted. Block B has four dependent blocks which are blocks C, E, F, and H. Block C also has four dependent blocks (D, F, G, and I), and block D has one (G). Thus, step 3 selects blocks B and C. Step 4 calculates the summation of the weights. Block B has four dependent blocks: C, E, F, and H whose weights are 2, 6, 7, and 3, respectively, as shown in the dependence graph of Figure 5(b). Thus, the summation of these weights is 18. Block C also has four dependent blocks: D, F, G, and I whose weights are 2, 6, 7, and 3. Thus, the summation of the four weights is also 18. Thus, both blocks B and C are selected in step 4 and the algorithm moves to the next step. Step 5 selects block B because block C depends on block B.

5. EXPERIMENTAL RESULTS

This section evaluates the compression efficiency and execution speed of the proposed pipelined execution and early termination. Table 1 shows the coding efficiency of the pipelined execution of 4 × 4 intraprediction with the processing order proposed in Section 3. Four video sequences, Foreman, Akiyo, Mother and daughter, and Stefan are used. Each of these video sequences is tested for three hundred CIF (352 × 288 pixels) size frames. The reference software version 7.3 is used for this comparison, and the rate distortion optimization for the high complexity mode decision [1] is not used in the simulation. For mode decision, SAD, instead of SATD, is used in the simulation. The QP values of 16, 20, 24, and 28 are tested, and all frames in a video sequence are encoded as I slices. The bit rate of the partially pipelined execution with the proposed order as shown in Figure 3(b) is compared with the original H.264/AVC encoder without pipelining. The compression efficiencies are also presented for the fully pipelined execution as shown in Figure 4(b). For fully pipelined execution, the results of both the original order and the proposed order are shown. Table 1 clearly shows that the proposed order outperforms the original order when the full-pipeline is applied.



(a) CIF (b) QCIF

Figure 8: Bit rate change averaged over 11 sequences.

Table 2: Number of nonpipelined blocks in a decoder.

Sequence	Original order	Proposed order
Foreman	6786	1535
Mother and daughter	5770	1302
Akiyo	4473	1087
Stefan	6879	1596

Rate-distortion performance is further improved by the partial pipeline and the performance remains almost same as that of the original algorithm without pipelining. With this performance comparison, the partial pipeline is finally chosen for hardware implementation presented in Section 6.

Table 2 shows the efficiency of the order selection algorithm for a decoder presented in Section 4.2. The efficiency is measured with the number of nonpipelined blocks. The order selection algorithm decreases the number of blocks that depend on their previous blocks, and consequently decreases the number of nonpipelined blocks. For simulation, the reference software version 7.3 is used with both of the rate distortion optimization and Hadamard transform for cost estimation turned off. The intraframe period is chosen as 10, and QP is fixed as 28. The number of reference frames is chosen as 1 for motion estimation. Four QCIF-sized sequences, Foreman, Mother and daughter, Akiyo, and Stefan, are used as test sequences. The number of 4×4 blocks that are prevented from the pipelined execution is given in the table. On average, the number of nonpipelined blocks is reduced to only 23.1% with the proposed order when compared to the original order.

The next simulation results show the best constant K in (1) when SAD is used to evaluate the cost of distortion in I slices. Note that 24 is assigned to K in the original H.264/AVC reference software, and experiments show that this value is optimal when SATD is used. On the other hand, 24 is not the best value when SAD is used in an I slice as explained next. The bit rate is measured with various values of K. Eleven video sequences, Foreman, Akiyo, Coastguard, Container, Hall monitor, Mother and daughter, News, Stefan, Table tennis, and Weather are used. Both the CIF size and the QCIF size (176×144 pixels) test vectors are used, and the number of frames is 300 for all test vectors. Five QP values, 16, 20, 24, 28, and 32 are used, and SAD is used instead of SATD. As the optimal constant K for only I slices is examined in these simulations, frames are encoded in the intraprediction mode. For the evaluation criterion, the bit rate change is defined as

$$BR_{change} = \frac{BR_{orig} - BR_{new}}{BR_{orig}} \times 100, \qquad (11)$$

where BR_{orig} denotes the bit rate of the reference software, and BR_{new} is the bit rate with a new constant K ranging from 24 to 180. Figure 8 shows the bit rate changes averaged over 11 CIF-sized and QCIF-sized test sequences. In Figure 8, value 80 achieves the best BR_{change} for QP equal to 32 while 100 generates a reasonably good result for other QP values. The PSNR variation depending on the value of K is also measured by experiments which show that the effect of K on the PSNR variation is negligible. As a result, the PSNR variation is not considered for the selection of K. From the results shown in Figure 8, 80 is chosen as the constant K for QP greater than 28 while 100 is chosen as K for QP less than or equal to 28. Note that the value of K is changed only for the case when SAD is used for cost evaluation. The value of K remains 24 when SATD is used for cost evaluation.

The next simulation is performed to find the optimal value of $M(1)$ in (6). With simulations with 11 video sequences, the value of $M(1)$ for the early termination of 4×4 and 16×16 intrapredictions in P slices is chosen as 2 and 0,

Table 3: Skipped block ratio [%].

Early termination type	4×4 in P ($M(1) = 2$)	16×16 in P ($M(1) = 0$)	4×4 in I ($M(1) = 2.5$)
Foreman	79.53	91.25	11.16
Akiyo	85.66	92.38	40.61
Coastguard	82.2	93.26	1.57
Container	82.51	91.77	20.82
Hall monitor	83.47	92.31	27.68
Mobile	85.08	91.42	1.75
Mother and daughter	83.42	91.66	32.3
News	83.38	91.88	30.31
Stefan	81.96	90.05	17.1
Table tennis	81.55	92.57	11.87
Weather	86.81	88.18	15
Average	83.23	91.52	19.11

respectively, because the numbers make a reasonable tradeoff between computational reduction and prediction accuracy. For the early termination of 4×4 intraprediction in I slices, the value of $M(1)$ is chosen as 2.5.

For evaluation of the computation savings by early termination, the number of early terminated 4×4 blocks is counted. Table 3 shows the ratio of the early terminated blocks against the total number of blocks. The simulation options are mostly same as the ones used for Table 1 except that QP is fixed as 28, and the intraframe period is chosen as 5 when evaluating early termination of 4×4 and 16×16 intrapredictions in P slices. For mode decision, the cost derived from SAD, instead of SATD, is used in the simulation. Eleven video sequences are simulated for this evaluation. The second column (4×4 in P) shows the early termination ratio for 4×4 intraprediction in P slices, and the third column shows the ratio for 16×16 intraprediction in P slices. The last column shows the early termination ratio for 4×4 intraprediction in I slices. A great amount of computation is saved in P slices as shown in the table. On average, 19.11% and 83.23% of 4×4 intrapredictions are terminated early in I and P slices, respectively. For 16×16 intraprediction, 91.52% of 4×4 blocks are early terminated in P slices. The overhead of early termination is the calculation of threshold in (4) and comparison of the threshold with the cost. Thus, the overhead is negligible compared to the amount of saved calculation for intra prediction.

Both pipelined execution and early termination are employed by the H.264/AVC encoder, and the combined effect is evaluated. Table 4 shows the early termination ratios for which the simulation options are same as the ones used for Table 3. Only the average of eleven sequences is shown in this table. The early termination ratios are slightly different from those in Table 3 as they are affected by the execution order of the 4×4 intraprediction optimized for pipelining. Although the early termination ratios are slightly reduced with the new order, the difference is negligibly small, and the ratios of early termination are still very large; consequently, the amount of computation is significantly reduced by early termination.

Table 4: Skipped block ratio [%] with the new order shown in Figure 3(b).

	4×4 in P ($M(1) = 2$)	16×16 in P ($M(1) = 0$)	4×4 in I ($M(1) = 2.5$)
Average	81.85	91.53	19.09

Table 5: Skipped block ratio [%] with integer motion estimation results.

	4×4 in P slices ($M(1) = 2$)	16×16 in P slices ($M(1) = 0$)
Average	78.55	80.71

The hardware implementation in this paper employs pipelined execution in such a way that integer motion estimation is performed one stage earlier than intraprediction while fractional motion estimation is performed in the same pipeline stage as intraprediction. Therefore, by the time when intraprediction begins, the result of fractional motion estimation is not available, but only the integer motion estimation is available. In this case, the cost of integer motion estimation is used for the threshold function of (4). Table 5 shows the skipped block ratio using the cost of integer motion estimation. The simulation options are same as the ones used for Table 3. The average skipped block ratio of 4×4 intraprediction in P slices is 78.55 percent. When compared with the fractional motion estimation (i.e., the result with Table 3), the ratios are reduced by 4.68 percent. For the early termination of 16×16 prediction, the skipped block ratio is reduced to 80.71 percent. Although the skipped block ratio is reduced by using the cost of integer motion estimation, it is still large enough to be used effectively for the early termination criterion of intraprediction.

As early termination is determined by comparing the cost of intraprediction with that of motion estimation, the skipped ratio is affected by the accuracy of motion estimation. A practical H.264 encoder often adopts a motion estimation with reduced complexity in order to reduce the hardware cost at the sacrifice of the accuracy. As a result, the skipped block ratio in a practical H.264 encoder may be not as large as that shown in Tables 3 or 4. However, to maintain a reasonably efficient compression, most practical H.264 encoders employ a motion estimation engine with a reasonably good accuracy. Then, the skipped block ratio is still pretty large as far as the accuracy of motion estimation is reasonably good. Therefore, the proposed early termination is effective for most practical H.264 encoders.

The bit rate and PSNR changes, by the proposed intraprediction, are measured and shown in Tables 6 and 7. These values are obtained by Bjontegaard's method presented in [34]. Table 6 shows the bit rate and PSNR changes for I slices when the 4×4 intraprediction is partially pipelined with the proposed processing order and also early terminated with the proposed threshold function. The simulation options are same as those for Table 1.

Table 6: Bit rate and PSNR change by pipeline and early termination in I slices.

Sequence	Pipeline	Early termination	Pipeline + early termination	Pipeline	Early termination	Pipeline + early termination
	Bit rate change [%]			PSNR change [dB]		
Foreman	0.2967	0.0105	0.336	−0.0236	−0.0006	−0.026
Akiyo	1.1516	−0.0749	1.155	−0.0741	0.0045	−0.0741
Coastguard	0.0846	0.0203	0.0964	−0.0087	−0.0021	−0.0099
Container	0.0872	−0.0071	0.0827	−0.0078	0.0008	−0.0072
Hall monitor	0.3597	0.0103	0.3554	−0.0225	0.0004	−0.0213
Mobile	0.1958	0.0068	0.209	−0.0281	−0.001	−0.0301
Mother and daughter	1.5653	0.5831	1.7049	−0.0983	−0.0365	−0.107
News	0.5627	0.1192	0.5662	−0.0455	−0.0097	−0.0459
Stefan	0.1607	0.0589	0.2198	−0.0195	−0.007	−0.0265
Table tennis	0.1629	0.029	0.1869	−0.015	−0.0026	−0.0171
Weather	0.6654	0.1395	0.7951	−0.0926	−0.0203	−0.1113
Average	0.4812	0.0814	0.5188	−0.0396	−0.0067	−0.0433

Table 7: Bit rate and PSNR change by pipeline and early termination in I and P slices.

Sequence	Early termination	Pipeline + early termination	Early termination	Pipeline + early termination
	Bit rate change [%]		PSNR change [dB]	
Foreman	0.1123	0.2695	−0.0055	−0.0135
Akiyo	−0.0505	0.8769	0.0021	−0.0427
Coastguard	0.052	0.0752	−0.0034	−0.0047
Container	−0.0258	0.0404	0.0014	−0.0024
Hall monitor	−0.0237	0.2848	0.0005	−0.0097
Mobile	−0.0096	0.0998	0.0006	−0.0092
Mother and daughter	0.5203	1.1445	−0.0215	−0.0467
News	0.1012	0.5277	−0.0061	−0.0316
Stefan	0.3991	0.4509	−0.03	−0.032
Table tennis	0.0697	0.1552	−0.0045	−0.0098
Weather	0.1729	0.7112	−0.0198	−0.0805
Average	0.1198	0.4215	−0.0078	−0.0257

The column, named as "Early Termination," shows the simulation results when only early termination is employed while the column named as "Pipeline" represents the results with only pipelined execution. The column "Pipeline + early termination" shows the results when both early termination and pipelined execution are employed. As shown in the table, the bit rate and PSNR changes are negligible for all test sequences. Table 7 shows the bit rate change and the PSNR change for both I and P slices when both 4 × 4 and 16 × 16 intrapredictions are terminated early, and the partial pipelined execution of 4 × 4 intraprediction is also employed. In this simulation, the I frame period is set to 5, so that frames are encoded as both I and P slices. Other simulation options are same as those for Table 6. The bit rate change by early termination is very small, even smaller than that in Table 6 because very small percentage (between 1% and 1.24%) of macroblocks is encoded as intramode in P slices [35]. As shown in Tables 6 and 7, the proposed algorithm suffers little degradationof either compression efficiency or image quality. These simulation results show that dramatic computation savings are achieved with little degradation in coding performance by the proposed pipelining and early termination.

All previous simulations are performed again with SATD-based mode decision instead of SAD-based decision. The result is similar to that obtained with SAD except that the early termination results in a slightly larger bit rate increase. It is because the SATD-based mode decision finds the best candidate more accurately than the SAD-based decision, and the penalty of wrong early termination becomes relatively expensive. However, thanks to the modification for the SAD-based cost evaluation proposed by this paper, the bit-rate increase by SAD-based decision is reduced, so that the bit rate is not significantly larger than that with SATD-based cost evaluation. Moreover, the selection of cost evaluation between SAD or SATD is not a main topic of this paper. Therefore, this paper employs a simple SAD-based cost evaluation.

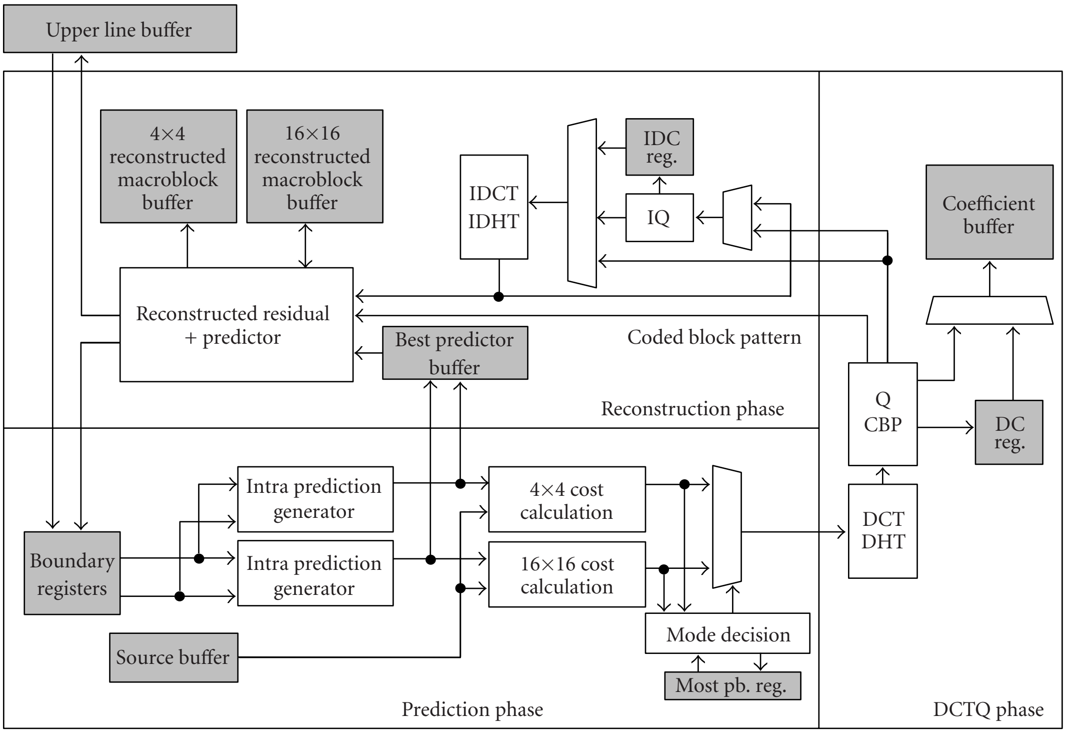


Figure 9: Block diagram of hardware implementation.

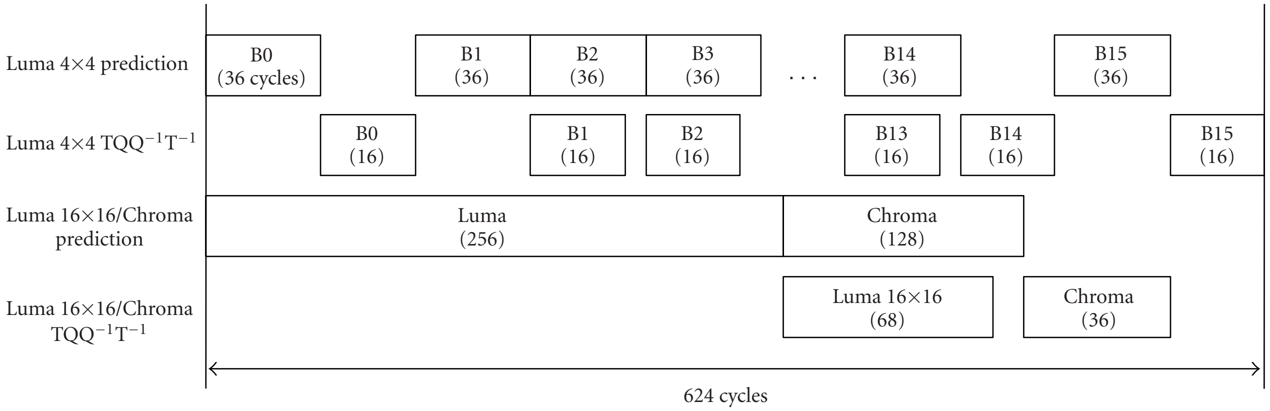
Table 8: Execution time comparison.

Method	Proposed			Huang et al. [1]			Li et al. [29]
Slice/mode	I and P slice	I slice	P slice	I and P slice			I and P slice
Processing option	No early termination	19.09% early termination	78.55% early termination	4 × 4 mode	16 × 16 mode	Average (2.54% in 16 × 16 mode)	—
cycles	624	497	143	544	612	546	<560

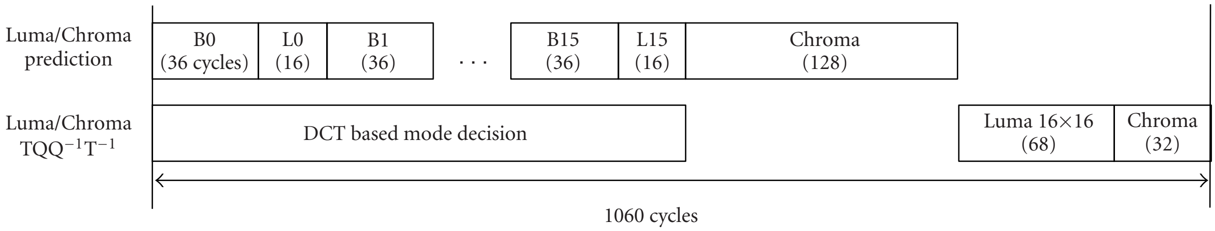
6. HARDWARE IMPLEMENTATION

This section presents the hardware implementation of the proposed pipelined execution and early termination. Figure 9 shows a block diagram of the hardware implementation which is similar to that presented in [29]. As shown in the figure, two separate prediction modules are implemented to simultaneously process Luma 4 × 4 and 16 × 16 predictions. The Luma 16 × 16 prediction module is shared by the Chroma 8 × 8 prediction because Luma 16 × 16 prediction requires a less amount of computation than Luma 4 × 4 prediction. Single hardware modules are implemented for DCT, quantization, inverse Quantization, and inverse DCT, each, as shown in Figure 9. To achieve the maximum throughput, these modules are pipelined. Hadamard transform is not implemented in the mode decision module because the cost of intraprediction is estimated not with SATD but with SAD.

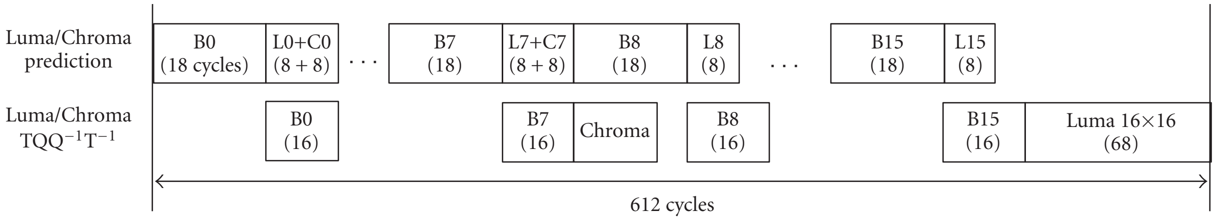
Figure 10(a) shows the proposed processing schedule that adopts the early termination and pipeline schedule presented in Sections 2 and 4.1, respectively. The hardware in Figure 9 is similar to that in [29], but the schedule shown in Figure 10(a) shows how the proposed early termination and pipeline schedule can efficiently utilize the hardware shown in Figure 9. The Luma 4 × 4 prediction and 4 × 4 reconstruction (T, Q, Q^{-1}, and T^{-1} operations) are partially pipelined as explained in Section 4.1, so that all predictions are overlapped with reconstructions of the previous blocks except block 1 and block 15 which are not pipelined due to the data dependence with the previous blocks. Luma 16 × 16 prediction is performed in parallel with Luma 4 × 4 prediction. After the execution of Luma 16 × 16 prediction, Chroma 8 × 8 prediction is executed by the same prediction module as Luma 16 × 16 prediction. The reconstruction operations for 16 × 16 Luma and 8 × 8 Chroma blocks are performed as soon as their predictions are completed. Although the reconstruction operations for 16 × 16 Luma and 8 × 8 Chroma blocks are shown below the 16 × 16 intrapredictions in this figure, these operations are executed by the same hardware module for the reconstruction of 4 × 4 block. As Luma 4 × 4 prediction



(a) Proposed intraprediction and reconstruction scheduling of Figure 9



(b) Intraprediction and reconstruction scheduling in [1]

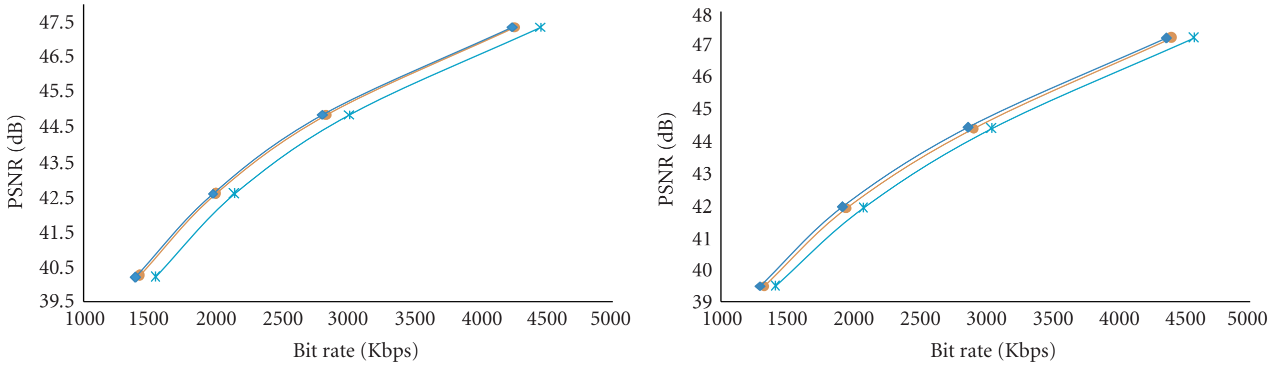


(c) Modified scheduling of (b) for a prediction module with eight-pixel parallelism

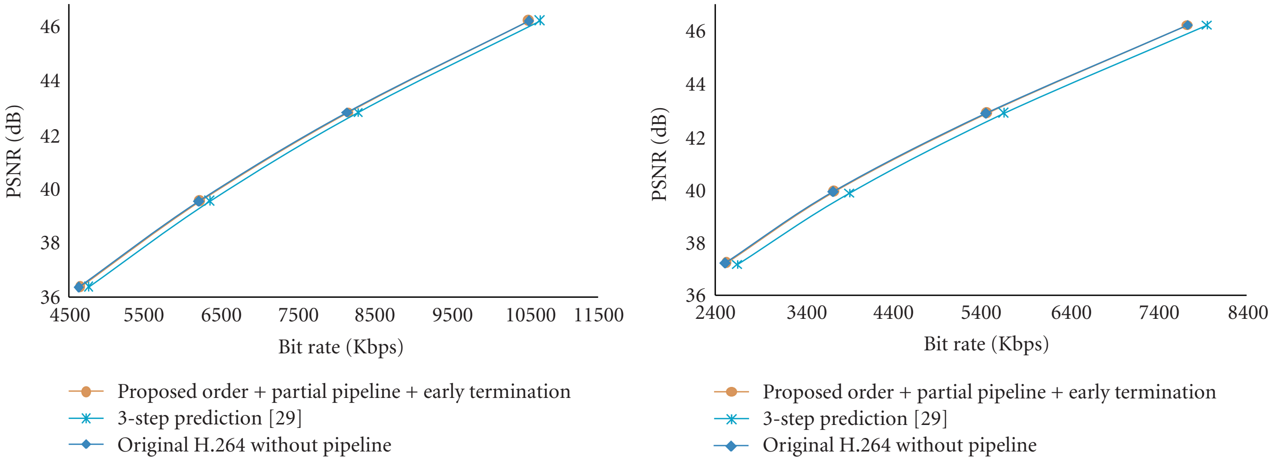
FIGURE 10: Comparison of intraprediction and reconstruction scheduling.

is the performance bottleneck, a higher priority is given to Luma 4 × 4 reconstruction than Luma 16 × 16/Chroma 8 × 8 reconstruction. Thus, the reconstruction of Luma 16 × 16/Chroma 8 × 8 is performed when Luma 4 × 4 reconstruction is not performed. A single hardware module is enough to process all Luma 4 × 4, Luma 16 × 16, and Chroma 8 × 8 because the computation amount for reconstruction operations is relatively small when compared with prediction operations.

In order to compare the efficiency of the proposed processing schedule with the previous work proposed in [1], the processing schedule presented in [1] is shown in Figure 10(b). The hardware in [1] uses only a single prediction module that performs Luma 4 × 4 and 16 × 16 predictions in an interleaved manner as shown in Figure 10(b). During the idle time between consecutive Luma 4 × 4 blocks, Luma 16 × 16 prediction is performed so that the prediction module is fully utilized. The prediction module is designed to exploit four-pixel parallelism and consequently process 4 pixels at every cycle. For Luma 4 × 4 prediction, each 4 × 4 block requires to process 16 pixels and 9 prediction modes. As a result, 126 pixels need to be processed for a single 4 × 4 block which consumes 36 cycles with the module with four-pixel parallelism. The execution time is shown in Figure 10(b) as the number inside parenthesis of each box. For the Luma prediction of each 4 × 4 block, 36 is assigned. Each 4 × 4 block for a Luma 16 × 16 prediction requires 16 cycles because 16 × 16 prediction requires 4 prediction modes. The same prediction module performs Chroma 8 × 8 prediction which is followed by Luma prediction and executed in 128 cycles. The DCT and quantization are followed by Chroma 8 × 8 prediction. The hardware module for DCT and quantization also exploits four-pixel parallelism so that it processes four pixels per a single cycle. As a result, four cycles are required to process a single 4 × 4 block. A Luma macroblock requires 16 4 × 4 DCT operations followed by 1 4 × 4 Hadamard transform for the 4 × 4 DC coefficients. Thus, a Luma macroblock requires 17 4 × 4 transforms which consume 68 cycles. For Chroma, 8 4 × 4 DCT operations and 2 2 × 2 Hadamard transforms are required. The 2 × 2 Hadamard operation is relatively simple, and its execution time is ignored. Then, 32 cycles are required



(a) Akiyo (b) Mother and daughter



(c) Stefan (d) Foreman

FIGURE 11: Rate distortion performance comparison.

to process for Chroma DCT operations. In total, 1060 cycles are required to process a single macroblock. In [1], the total execution time for a single macroblock is about 1300 cycles which include initial data loading time. For fair comparison with the proposed architecture, this initial loading time is excluded in Figure 10(b).

For comparison, the execution time of the proposed schedule shown in Figure 10(a) is estimated assuming that the same hardware modules as [1] are used. Thus, 36 cycles are required for a single 4 × 4 block of Luma 4 × 4 prediction and 68 and 32 cycles are, respectively, required for Luma 16 × 16 and Chroma 8 × 8 predictions. These execution cycles are given as the numbers inside the parentheses in each box in Figure 10(a). For reconstruction operation, 16 cycles are assigned because it requires T, Q, Q^{-1}, T^{-1}, operations, each of which requires to process 16 pixels with four-pixel parallelism. Note that 4 cycles are assigned to a single 4 × 4 block of T and Q operations in [1] because these operations are performed for the entire macroblock so that they can be pipelined and the throughput becomes 4 cycles per every 4 × 4 block. However, in this case, T, Q, Q^{-1}, T^{-1}, operations are performed for a single 4 × 4 block, so that they cannot be pipelined for multiple 4 × 4 blocks. Thus, it is reasonable to assign 16 cycles for these four operations. Then, total execution time is 624 cycles. When early termination is applied, then the execution time is reduced. Experimental results in Table 6 shows that 19.09 percent of 4 × 4 intraprediction is terminated early in I slices. Then, the three last 4 × 4 blocks (blocks 13, 14, and 15) are expected to be not performed by early termination while the probability of the fourth last 4 × 4 block (block 12) to be processed is 95%. All other blocks (from block 0 to 11) are expected to be processed. With the pipelined execution as shown in Figure 10(a), the processing time of the first 12 blocks is 448 cycles. The expected cycles of the 13th block are 49 cycles ($= (36 + 16) \times (0.95)$). Thus, the expected execution time with 19.09% early termination is 497 cycles. In P slices, 78.55 percent of 4 × 4 intraprediction is terminated early by the cost derived from the result of the integer motion estimation as shown in Table 5. Thus, the expected processing time is 143 ($= 52 + 36 + 36 + 52 \times 0.43$) cycles. As 4 × 4 intraprediction requires the longer time than 16 × 16 prediction, 143 is the expected execution cycles of intraprediction.

The direct comparison of the execution times of the schedules given in Figures 10(a) and 10(b) is not fair because the hardware implementation in [1] employs four-pixel parallelism for prediction modules while the implemented

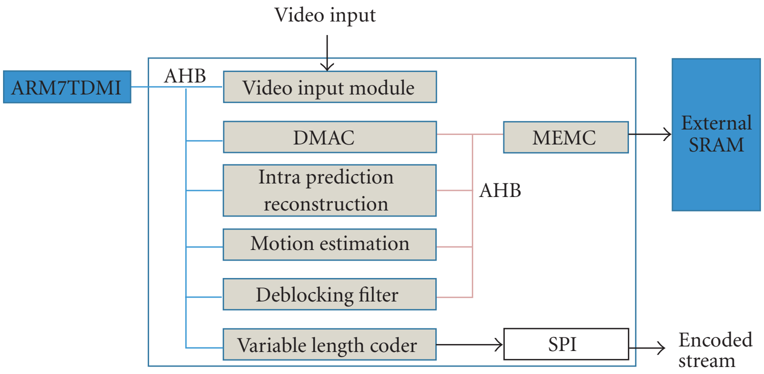


FIGURE 12: H.264/AVC encoder block diagram.

architecture in this paper employs eight-pixel parallelism. For fair comparison, the hardware implementation in [1] is modified to exploit eight-pixel parallelism, so that the execution speed for prediction is doubled. Then, Figure 10(b) is modified as Figure 10(c) in which a single 4 × 4 block requires 18 and 8 cycles (i.e., halves of the execution time in Figure 10(b)) for Luma 4 × 4 and 16 × 16 predictions, respectively. The DCT and quantization unit remain unchanged, so that the execution time for DCT and quantization is the same as that in Figures 10(a) and 10(b). Thus, 16 cycles are assigned to the reconstruction operation. In this case, the execution time for the reconstruction is larger than that of Luma 16 × 16 prediction for a single 4 × 4 block. Then, the execution time for the reconstruction cannot be hided by the overlapped execution of a single 4 × 4 block of 16 × 16 Luma prediction. To avoid bubble cycles, it is necessary to interleave Chroma prediction as shown in Figure 10(c). Thus, from block B0 to B7, Luma 4 × 4, Luma 16 × 16, and Chroma 8 × 8 predictions are interleaved. Then, the execution of reconstruction operations is completely overlapped with Luma 16 × 16 and Chroma 8 × 8 predictions so that there exist no bubble cycles. As there are only eight 4 × 4 blocks for Chroma 8 × 8 prediction, there exist no Chroma 8 × 8 predictions from block B8 to B15. Thus, bubble cycles are unavoidable. The total execution time of intra predictions is 544 cycles. After intrapredictions are completed, then the DCT and quantization are performed. As Chroma 8 × 8 predictions are finished early, the DCT and quantization for Chroma pixels also begin righter after the completion of Chroma prediction so that they can be interleaved with the reconstruction operation for 4 × 4 Luma prediction. The DCT and quantization for Luma 16 × 16 begin after the completion of Luma 16 × 16 prediction, and this operation requires 68 cycles. Thus, the total execution time is 612 cycles. Note that DCT and quantization for Luma 16 × 16 need to be performed only for the case when Luma 16 × 16 mode is selected. With experiments, it is observed that 16 × 16 mode is selected over 4 × 4 mode for about 2.54 percent when K in (1) is 24. If K is chosen as 100 from the results of Figure 8, the percentage of 16 × 16 mode is 23.65. Thus, $K = 24$ is the better choice than K=100 for this architecture because the selection of $K = 24$ decreases the probability to compute the extra DCT and quantization for the 16 × 16 prediction mode. The average execution cycles with $K = 24$ for the DCT and quantization for 16 × 16 mode are 2 (= 68 × 0.0254). Thus, the average execution time of the process schedule shown in Figure 10(c) is 546 (= 544 + 2) cycles.

Other efficient hardware implementations are presented in [29, 36]. The implementation in [29] is an improvement of [36] and adopts eight-pixel parallelism for prediction modules and four-pixel parallelism for reconstruction operation. Thus, the hardware complexity is about the same as that used for Figures 10(a) and 10(c). In addition, this implementation employs a fast three-step algorithm for intraprediction to speedup the computation at the sacrifice of the compression efficiency. As a result, the execution cycles for a single 4 × 4 block operation of Luma 4 × 4 prediction are reduced to 20 cycles, and the total execution time is less than 560 cycles.

The rate-distortion performances of JM H.264 reference software version 7.3 [30] and the three step mode decision algorithm in [29] are compared with the partial pipeline with early termination proposed in this paper. Four video sequences, Akiyo, Mother and daughter, Stefan, and Foreman are used for the comparison, and the results are shown Figure 11. All frames are encoded as I slices and four QP values, 16, 20, 24, and 28 are used. Other simulation options are same as those for Table 1. The value of K in (1) is chosen as 100 for all three methods according to the experimental results shown in Figure 8. As shown in these four figures, the performance of the proposed intraprediction is slightly worse than the original H.264 reference software while it achieves the better performance than the algorithm in [29] for all four test sequences. On average, the proposed methods are 4.6% better than [29] in terms of bit rate.

Table 8 summarizes the execution time of the three implementations, the one proposed in this paper, another one proposed in [1] with its prediction module modified to exploit eight-pixel parallelism as shown in Figure 10(c), and the other one proposed in [29]. Note that the execution cycle is obtained with the hardware of the same complexity (eight-pixel parallelism for intraprediction and four-pixel parallelism for DCT and quantization) for all three implementations. The proposed architecture requires the least execution time with early termination employed. For I slices, the proposed architecture requires about 49 and

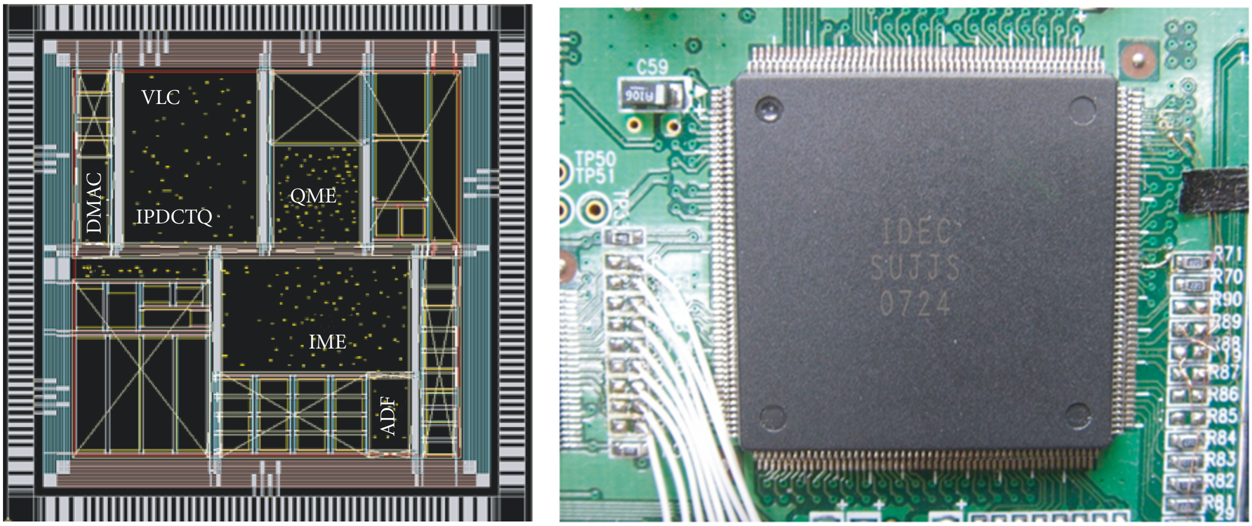


Figure 13: Chip layout and photograph.

63 cycles less than the other two architectures in [1, 29], respectively. For P slices, the proposed methods dramatically speedup the processing time with the high probability of early termination, so that it requires about 400 cycles less than the other two architectures.

The intraprediction and reconstruction modules are implemented in VHDL and integrated with the other computation units for an H.264/AVC encoder to constitute a hardware-based encoder chip. Figure 12 shows a block diagram of the encoder. In addition to the intraprediction and reconstruction modules, motion estimation, deblocking filter, and variable length coder are also implemented in hardware, and the remaining part of computation is processed by the ARM7TDMI processor. Video input module (VIM) accepts image data from an image sensor and SPI interface outputs the encoded stream. Direct memory access controller (DMAC) and memory controller (MEMC) are designed for efficient data communication with an external SRAM. Two AMBA AHB buses are used for the communication between modules. One AHB bus is mainly used for the control of the hardware modules by ARM7TDMI processor, and the other AHB bus is mainly used for data communication between hardware modules and external memory. Figure 13 shows the layout and the chip photograph of the H.264/AVC encoder shown in Figure 12. The die area of the H.264/AVC encoder is 5 mm × 5 mm using the Dongbu 1P6M 0.13 μm CMOS technology.

For the proposed processing sequence shown in Figure 10(a), the best hardware utilization is achieved when the execution time of Luma 4 × 4 prediction and reconstruction (T, Q, Q^{-1}, and T^{-1} operations) is the same. In the chip shown in Figure 13, the intraprediction hardware modules are designed such that a single 4 × 4 block requires 25 cycles for Luma 4 × 4 prediction while the 16 × 16 Luma prediction and 8 × 8 Chroma prediction require 95 and 84 cycles, respectively. The reconstruction unit processes one 4 × 4 block in 28 cycles while three kinds of reconstructions, the 4 × 4 reconstruction and two 16 × 16 or Chroma reconstructions, can be overlapped so that the reconstruction unit can start reconstruction of another 4 × 4 block for every eight cycles because T, Q, Q^{-1}, and T^{-1} operations require eight execution cycles each. The cost evaluation is made with SAD, not SATD, and the 4 × 4 Luma prediction is performed in the order as shown in Figure 3(b). As a result, 532 cycles are required to complete all sixteen blocks without any early termination. The intraprediction and reconstruction modules are synthesized by Synopsys Design Compiler and the gate count of the proposed intraprediction and reconstruction logic is 126 617. These modules also require 4992 bytes of SRAM. This hardware size is relatively large when compared with the previous work (e.g., the previous work in [29] which requires 72 K gates with 1632 bytes of SRAM). The intraprediction module in the chip shown in Figure 13 is designed to perform not only for intraframe encoding but also for interframe encoding. For example, the cost evaluation of the DCT and quantization outcome for interframe encoding is also performed by the intraprediction module, and additional buffer to store the result of interframe DCTQ for ADF is also included. Therefore, direct comparison of the gate counts of the intraprediction module in Figure 13 with a previous intraprediction module designed only for intraframe encoding is not appropriate, and therefore it is not presented in this paper. Note that the additional hardware cost to implement the proposed early termination is negligible because only simple logics for comparison of two values and derivation of the threshold function are necessary for early termination while no additional hardware is necessary for pipeline schedule because it is fixed when the chip is implemented.

7. CONCLUSIONS

This paper proposes an early termination and pipelined execution of H.264/AVC intraframe processing. The early termination saves over 19 and 81 percent of computation for 4 × 4 intraprediction in I and P slices, respectively. For 16 × 16 intraprediction, about 91 percent of computation is saved by the early termination in P slices. Through pipelined execution, the computation time is reduced by 41 percent when the execution time of 4 × 4 intraprediction and reconstruction is about the same. The speedup by the proposed algorithm is achieved at the sacrifice of the R-D performance although the degradation of the performance is not significant as shown in the experimental results.

The proposed early termination and pipelined execution do not adopt a complex control flow or data structure which is often difficult to implement in hardware. Thus, they are suitable for hardware implementation. Table 8 shows that the proposed early termination and pipelining make it possible to design a hardware intraprediction module that processes a single macroblock in an average of 497 cycles. With this processing speed, HD-size video can be processed in 30 fps with a low operating frequency of 54 MHz. Thus, the proposed early termination and pipelining can be efficiently used for an H.264/AVC encoder targeting Mobile HD (1280 × 720) size video applications, such as HD-size camcorder and notebook camera.

ACKNOWLEDGMENTS

This work was funded by Seoul R&BD Program (BU070067) of Seoul Development Institute, and CAD tools were supported by IDEC (in South Korea).

REFERENCES

[1] Y.-W. Huang, B.-Y. Hsieh, T.-C. Chen, and L.-G. Chen, “Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 3, pp. 378–401, 2005.

[2] F. Pan, X. Lin, S. Rahardja, et al., “Fast mode decision algorithm for intraprediction in H.264/AVC video coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 7, pp. 813–822, 2005.

[3] B. Meng, O. C. Au, C.-W. Wong, and H.-K. Lam, “Efficient intra-prediction mode selection for 4 × 4 blocks in H.264,” in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME ’03)*, vol. 3, pp. 521–524, Baltimore, Md, USA, July 2003.

[4] K. Suh, S. Park, and H. Cho, “An efficient hardware architecture of intra prediction and TQ/IQIT module for H.264 encoder,” *ETRI Journal*, vol. 27, no. 5, pp. 511–524, 2005.

[5] B. Meng, O. C. Au, C.-W. Wong, and H.-K. Lam, “Efficient intra-prediction algorithm in H.264,” in *Proceedings of the IEEE International Conference on Image Processing (ICIP ’03)*, vol. 3, pp. 837–840, Barcelona, Spain, September 2003.

[6] Y.-K. Lin and T.-S. Chang, “Fast block type decision algorithm for intra prediction in H.264 FRext,” in *Proceedings of the IEEE International Conference on Image Processing (ICIP ’05)*, vol. 1, pp. 585–588, Genoa, Italy, September 2005.

[7] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, “Performance analysis of hardware oriented algorithm modifications in H.264,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP ’03)*, vol. 2, pp. 493–496, Hong Kong, April 2003.

[8] F. Fu, X. Lin, and L. Xu, “Fast intra prediction algorithm in H.264/AVC,” in *Proceedings of the 7th International Conference on Signal Processing (ICSP ’04)*, vol. 2, pp. 1191–1194, Beijing, China, August-September 2004.

[9] A.-C. Tsai, J.-F. Wang, W.-G. Lin, and J.-F. Yang, “A simple and robust direction detection algorithm for fast H.264 intra prediction,” in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME ’07)*, pp. 1587–1590, Beijing, China, July 2007.

[10] J. Jung and D. N. Kwon, “DCT based fast 4 × 4 intraprediction mode selection,” in *Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference (CCNC ’07)*, pp. 332–335, Las Vegas, Nev, USA, January 2007.

[11] C.-L. Lim, K.-H. Thung, and P. Raveendran, “Edge vector based mode decision for H.264/AVC intra prediction,” in *Proceedings of the 1st Asia International Conference on Modelling & Simulation (AMS ’07)*, pp. 308–312, Phuket, Thailand, March 2007.

[12] G. Hwang, J. Park, B. Jung, et al., “Efficient fast intra mode decision using transform coefficients,” in *Proceedings of the 9th International Conference on Advanced Communication Technology (ICACT ’07)*, vol. 1, pp. 399–402, Gangwon-Do, Korea, February 2007.

[13] A.-C. Tsai, A. Paul, J.-C. Wang, and J.-F. Wang, “Efficient intra prediction in H.264 based on intensity gradient approach,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS ’07)*, pp. 3952–3955, New Orleans, La, USA, May 2007.

[14] Z. Wang, J. Yang, Q. Peng, Z. Ma, and C. Zhu, “A fast transform domain based algorithm for H.264/AVC intra prediction,” in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME ’07)*, pp. 1563–1566, Beijing, China, July 2007.

[15] J.-F. Wang, J.-C. Wang, J.-T. Chen, A.-C. Tsai, and A. Paul, “A novel fast algorithm for intra mode decision in H.264/AVC encoders,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS ’06)*, pp. 3498–3501, Island of Kos, Greece, May 2006.

[16] R. Su, G. Liu, and T. Zhang, “Fast mode decision algorithm for intra prediction in H.264/AVC,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP ’06)*, vol. 2, pp. 921–924, Toulouse, France, May 2006.

[17] Z. Yong, D. Feng, and L. Shou, “Fast 4 × 4 intra-prediction mode selection for H.264,” in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME ’04)*, vol. 2, pp. 1151–1154, Taipei, Taiwan, June 2004.

[18] Z. Wei, H. Li, and K. N. Ngan, “An efficient intra mode selection algorithm for H.264 based on fast edge classification,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS ’07)*, pp. 3630–3633, New Orleans, La, USA, May 2007.

[19] C.-C. Wang, T.-S. Chen, and C.-W. Tung, “Fast intra-mode decision in H.264 using interblock correlation,” in *Proceedings of the IEEE International Conference on Image Processing (ICIP ’06)*, pp. 1345–1348, Atlanta, Ga, USA, October 2006.

[20] Z. Kun, Y. Chun, L. Qiang, and Z. Yuzhou, “A fast block type decision method for H.264/AVC intra prediction,” in *Proceedings of the 9th International Conference on Advanced Communication Technology (ICACT ’07)*, vol. 1, pp. 673–676, Gangwon-Do, Korea, February 2007.

[21] J. B. Song, B. Li, W. Li, and L. Jiang, "A novel fast intra prediction algorithm applied in H.264/AVC," in *Proceedings of the 8th International Conference on Signal Processing (ICSP '06)*, vol. 1, pp. 16–20, Beijing, China, November 2006.

[22] C.-L. Yang, P. Lai-Man, and W.-H. Lam, "A fast H.264 intra prediction algorithm using macroblock properties," in *Proceedings of the IEEE International Conference on Image Processing (ICIP '04)*, vol. 1, pp. 461–464, Singapore, October 2004.

[23] R. Yang, H. Lu, X. Xue, and Y.-P. Tan, "An efficient early termination algorithm of intra prediction for H.264/AVC," in *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision (ICARCV '06)*, pp. 1–4, Singapore, December 2006.

[24] U. Mithun and P. S. S. B. K. Gupta, "An early intra mode skipping technique for inter frame coding in H.264 BP," in *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE '07)*, pp. 1–2, Las Vegas, Nev, USA, January 2007.

[25] Z. Wei and K. N. Ngan, "A fast macroblock mode decision algorithm for H.264," in *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS '06)*, pp. 772–775, Singapore, December 2006.

[26] E. Arsura, L. Del Vecchio, R. Lancini, and L. Nisti, "Fast macroblock intra and inter modes selection for H.264/AVC," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '05)*, pp. 378–381, Amsterdam, The Netherlands, July 2005.

[27] Y.-C. Kao, H.-C. Kuo, Y.-T. Lin, et al., "A high-performance VLSI architecture for intra prediction and mode decision in H.264/AVC video encoding," in *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS '06)*, pp. 562–565, Singapore, December 2006.

[28] E. Sahin and I. Hamzaoglu, "An efficient hardware architecture for H.264 intra prediction algorithm," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE '07)*, pp. 1–6, Nice, France, April 2007.

[29] D.-W. Li, C.-W. Ku, C.-C. Cheng, Y.-K. Lin, and T.-S. Chang, "A 61 MHz 72 K gates 1280 × 720 30FPS H.264 intra encoder," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '07)*, vol. 2, pp. 801–804, Honolulu, Hawaii, USA, April 2007.

[30] JVT Reference Software Version 7.3, http://iphome.hhi.de/suehring/tml/download/old_jm.

[31] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, et al., "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 6, pp. 673–688, 2006.

[32] Z. Liu, L. Li, Y. Song, T. Ikenaga, and S. Goto, "VLSI oriented fast multiple reference frame motion estimation algorithm for H.264/AVC," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '07)*, pp. 1902–1905, Beijing, China, July 2007.

[33] Y.-H. Chen, T.-C. Chen, and L.-G. Chen, "Power-scalable algorithm and reconfigurable macro-block pipelining architecture of H.264 encoder for mobile application," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 281–284, Toronto, Canada, July 2006.

[34] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," in *Proceedings of the 13th Video Coding Experts Group Meeting (VCEG-M33 '01)*, Austin, Tex, USA, April 2001.

[35] I. Choi, J. Lee, and B. Jeon, "Fast coding mode selection with rate-distortion optimization for MPEG-4 Part-10 AVC/H.264," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 12, pp. 1557–1561, 2006.

[36] C.-W. Ku, C.-C. Cheng, G.-S. Yu, M.-C. Tsai, and T.-S. Chang, "A high-definition H.264/AVC intra-frame codec IP for digital video and still camera applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 8, pp. 917–928, 2006.