國立交通大學

資訊科學與工程研究所

碩士論文

利用運算重排發揮 H.264 中去方塊濾波器的平行度

Exploiting Parallelism in the H.264 Deblocking Filter by Operation Reordering

研究生：王薏婷

指導教授：鍾崇斌 教授

中華民國一百年九月

利用運算重排發揮 H.264 中去方塊濾波器的平行度

Exploiting Parallelism in the H.264 Deblocking Filter by Operation Reordering

研 究 生：王薏婷　　Student：Yi-Ting Wang

指導教授：鍾崇斌　　Advisor：Chung-Ping Chung

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
In

Computer Science

Sep. 2011

Hsinchu, Taiwan, Republic of China

中華民國 一百 年 九 月

利用運算重排發揮 H.264 中去方塊濾波器的平行度

學生：王薏婷　　指導教授：鍾崇斌　博士

國立交通大學資訊科學與工程研究所碩士班

摘要

在 H.264 影像壓縮標準中，去方塊濾波器的計算量大約占整體解碼器的三分之一。隨著多工處理器將成為未來系統設計的趨勢，若可把去方塊濾波器內的運算良好的分配到各個處理單元，則可以省下計算時間。在這篇論文中，我們提出了兩個粒度來做平行處理的單位，第一是 H.264 去方塊濾波器所使用的基本單位 16 像素長的邊，第二是根據分析去方塊濾波器所能達到的最高平行度所使用的最小基本單位 4 像素長的邊。此外，在此篇論文中我們提出一個方法使得在硬體資源受限的情況下，盡可能地充分利用所擁有的硬體資源，以及符合我們設計所需要的去方塊濾波器硬體需求。在硬體無限制下，對 1920×1080 與 1080×1920 這兩種形狀的圖片做去方塊濾波，16 像素長的邊這個粒度的處理順序與二維波前方式所提出來的處理順序相比所得到的加速分別為 1.57 與 2.15 倍，4 像素長的邊這個粒度的處理順序與二維波前方式所提出來的處理順序相比所得到的加速分別為 1.92 與 2.44 倍。另外，我們的方法可使處理時間成正比於圖片大小的平方根(在一樣的圖片長寬比下)。

Exploiting Parallelism in the H.264 Deblocking Filter by Operation Reordering

Student：Yi-Ting Wang　　Advisor：Chung-Ping Chung

Institute of Computer Science and Engineering
National Chiao-Tung University

Abstract

In the H.264 video compression standard, the deblocking filter contributes about one-third of all computation in the decoder. With multi-processor architectures becoming the future trend of system design, computation time reduction can be achieved if the deblocking filter well apportions its operations to multiple processing elements. In this paper, we apply a 16 pixel long boundary, the basic unit for deblocking in the H.264 standard and a 4 pixel long boundary as the basis for analyzing and exploiting possible parallelism in deblocking filtering. Moreover, a possible compromise to fully utilize limited hardware resources and hardware architectural requirements for deblocking are also proposed in this paper. Compared with the 2D wave-front method order for deblocking both 1920*1080 and 1080*1920 pixel sized frames, the 16 pixel long boundary method gains speedups of 1.57 and 2.15 times given an un-limited number of processing elements respectively, and the 4 pixel long boundary method gains speedups of 1.92 and 2.44 times given an un-limited number of processing elements respectively. Using this approach, the execution time of the deblocking filter is proportional to the square root of the growth of the frame size (keeping the same width/height ratio), pushing the boundary of practical real-time deblocking of increasingly larger video sizes.

致謝

本論文的完成，首先要感謝我的指導教授 鍾崇斌老師，因為老師這兩年來的耐心教導與鼓勵，我才能順利完成本論文。另外，口試時也承蒙 陳添福老師及謝萬雲老師提出寶貴的建議與意見，使得本論文能夠更完整，在此一併致上感謝之意。

除此之外，還要感謝博士班學長翁綜禧，總是不厭其煩地跟我討論研究上遇到的問題，並且會適時的給予建議。還有要感謝實驗室的學長姊、同學以及學弟，在兩年多的日子裡，實驗室生活的點點滴滴，共同討論研究，休息時間的閒聊，還有一起出遊的回憶，都是我的美好回憶，感謝你們的陪伴讓我的研究所生涯多采多姿。另外，對於所有幫助過我的人，致上最真誠的感謝。

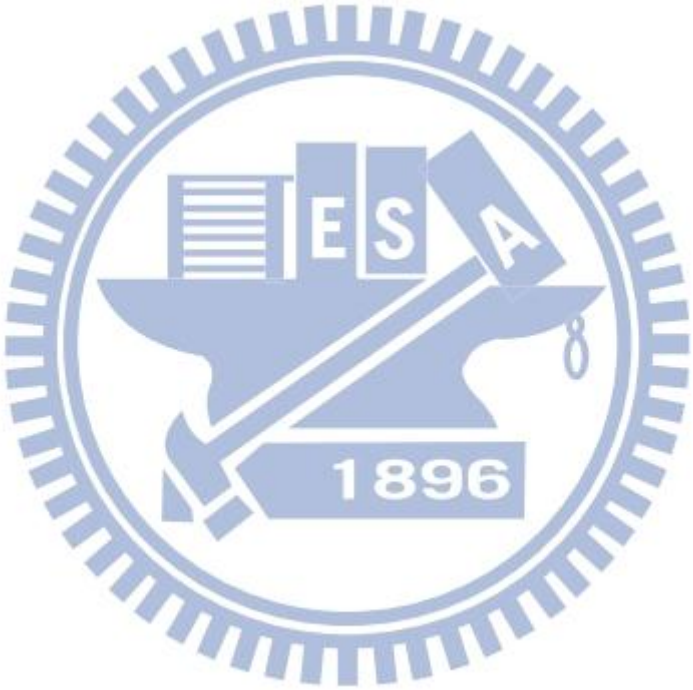
最後，要感謝我的家人，謝謝你們一路上的支持與勉勵。

王蕙婷 謹誌於

國立交通大學資訊科學與工程研究所碩士班

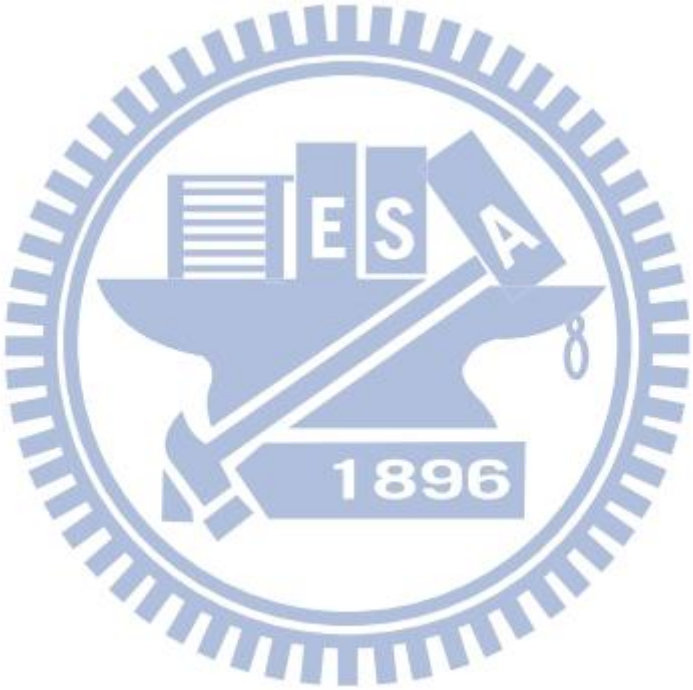
中華民國一百年十月

Content



List of figure

List of table



Chapter 1 Introduction

The H.264 standard provides acceptable image quality combined with a reduction in bit-rate compared with existing video compression standards. Besides this, it can also provide higher adaptability and better error resilience for a wider range of applications. With regards to the compression rate, the bit rate of H.264 is almost 50% lower than that of the MPEG-2, H.263v2 and MPEG-4 Advanced Simple Profile video compression standards for the same picture quality [7].

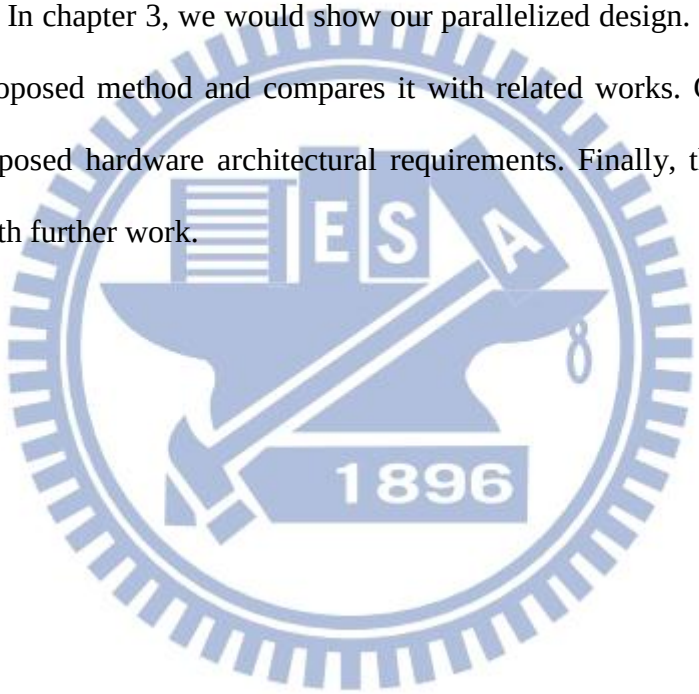
Deblocking is intended to smooth block-edge artifacts caused by the decoding process and enhance picture quality. In the encoding process, the H.264 encoder uses the macroblock (MB, 16x16 pixel square) as the basic coding unit. Quantization of the macroblocks causes visual discontinuities between the edges of decoded macroblocks. Pixels located on macroblock boundaries with a similar value may for the above reason be decoded with a larger difference in values, resulting in a decline in picture quality. Therefore, the purpose of deblocking is to smooth block artifacts caused by the decoding process to enhance picture quality. Another advantage of deblocking is to increase coding efficiency. Decoded and deblocked images will be referenced later, and because the picture is of higher quality, there will be a reduction in the encoded bit rate.

Deblocking filtering accounts for one-third of all computation in the decoder [1]. With multi-core becoming the trend, if deblocking can be processed using a multi-core parallel processing architecture, the processing can be distributed to different computing processing elements (PEs) to address and reduce execution time. Currently parallel processing of deblocking focuses on parallelization at the MB-level.

We find that parallelizing deblocking at a finer granularity can be developed according to our presented design.

We analyze the deblocking order to obtain the dependency between the various boundaries, and then propose an execution order, with execution of deblocking in this order giving higher parallelism.

The rest of this paper is organized as follows. In chapter 2, we would introduce the background of the deblocking filter and related work for deblocking filter parallelization. In chapter 3, we would show our parallelized design. Chapter 4 would analyze the proposed method and compares it with related works. Chapter 5 would shows our proposed hardware architectural requirements. Finally, the conclusion is given along with further work.



Chapter 2 Background and Related work

2.1 Background

The deblocking filter is used in order to smooth block-edge artifacts. Figure 2-1(b) shows a block-edge artifact caused by a large difference in pixel values. The pixels P0~P3 and Q0~Q3 in Figure 2-1(b) can be located either vertically or horizon-tally as shown in Figure 2-1(a)[6]. A deblocking filter is applied on the P0~P3 and Q0~Q3 pixel values to make these eight values visually smooth. The pixel value distribution after applying the deblocking filter is shown in Figure 2-1(c).

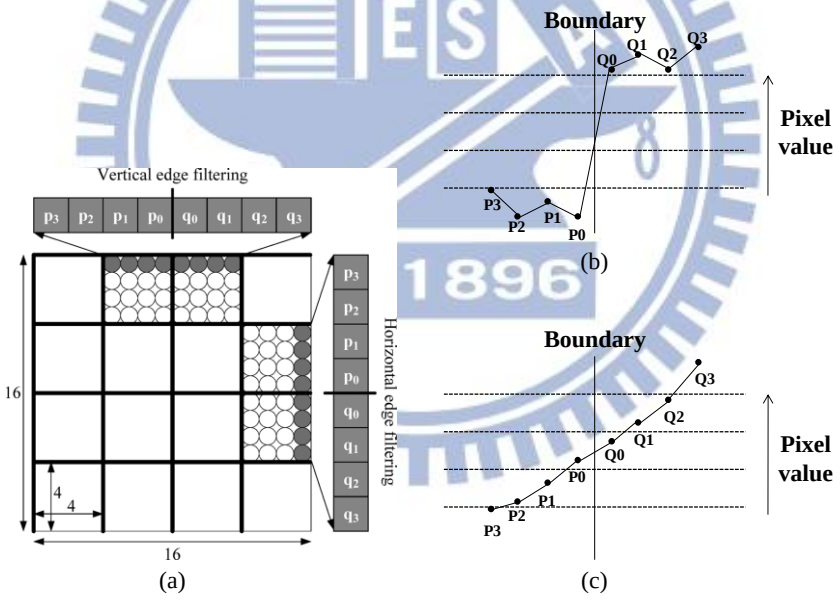


Figure 2-1 (a) Affected pixels in deblocking [6] (b) The pixel values before deblocking filtering; the P0~P3 and Q0~Q3 pixel value gap causes a visual discontinuity. (c) After deblocking filtering; the pixel values are now smooth.

Deblocking is needed for both MB boundaries and 4*4 block boundaries. As the MB is the basic coding unit in H.264, block-edge artifacts occur easily at MB boundaries. In addition, there are some coding modes using 4*4 blocks for inter

prediction and intra prediction. For these cases deblocking is needed to smooth the block-edge artifacts.

The MB deblocking internal (intra MB) execution order as defined by the H.264 standard is shown in Figure 2-2(a). Execution starts by deblocking a column of pixels moving horizontally left to right, and then a row of pixels moving vertically top to bottom. The inter MB execution order is shown in Figure 2-2(b), and moves from left to right, top to bottom.

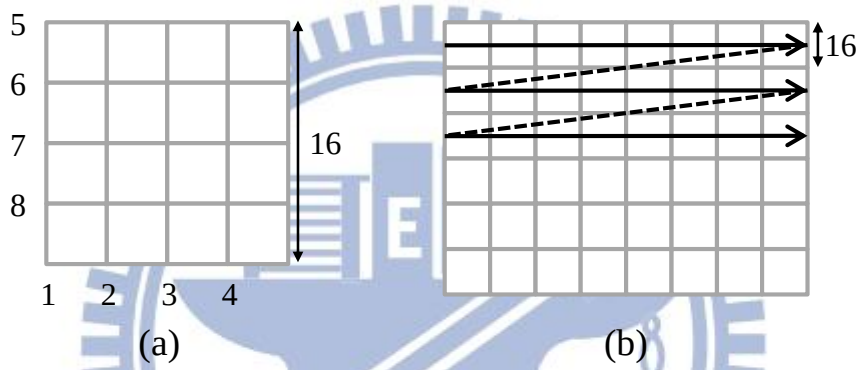


Figure 2-2 (a) Intra MB order. (b) Inter MB order.

Although the H.264 standard defines the deblocking order as shown above, as long as the final decoding results in the correct output, the above order can be changed. Changing the order in which the calculation is performed is an opportunity for parallelizing deblocking filtering. We propose a conceptual design to improve the parallelizability of the deblocking filter.

2.2 Related Work

The 2D wave-front method is based on using the MB as a unit for parallelization [2]. In Figure 2-3(a), according to the deblocking order, we find the current MB has a data dependency on the Upper, Upper-Right and Left MBs. So when using a MB as

the parallelization unit, the Upper, Upper-Right and Left MB must be deblocked before the Current MB. In Figure 2-3(b), MBs that can be processed simultaneously are numbered together.

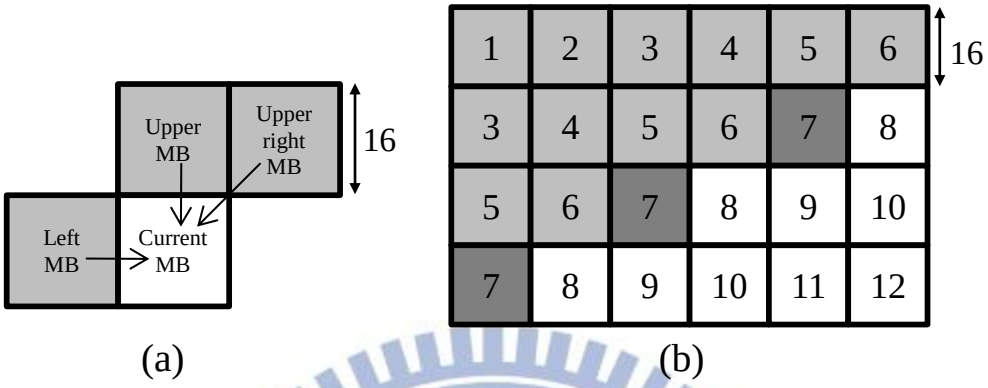


Figure 2-3 (a) Data dependencies in inter MB deblocking. (b) MBs that can be processed simultaneously.

According to this observation, this method does not have a fixed degree of parallelism. The degree of parallelism initially steadily increases. Some wind up time is needed before reaching maximum parallelism. After maintaining maximum parallelism for some time, the degree of parallelism will begin to steadily decrease. In Figure 2-4, the units of time are in terms of the time to deblock one MB, and the frame size is 1920*1080.

The 2D wave-front method's maximum parallelism and required wind up time and wind down time can be expressed by the equations:

$$Maximum\ parallelism\ (P) = Min\left(\left\lceil \frac{1}{2} M_W \right\rceil, M_H\right) \quad (1)$$

$$Build\ up\ time\ and\ Degrading\ time = 2 * (P - 1) \quad (2)$$

M_W: # of columns of MBs in frame.

M_H: # of rows of MBs in frame.

Where the wind up and wind down time are in units of time required for deblocking a MB.

In theory, for this method the maximum parallelism should be equal to the total number of MBs in a column which is 68 in the above example, but from Figure 2-4 the maximum parallelism is 60, which is less. The reason for this is the frame aspect ratio. The first row of MBs has finished being processed, yet the last row of MBs has not yet begun to be processed, resulting in the degree of parallelization unable to reach the theoretical maximum.

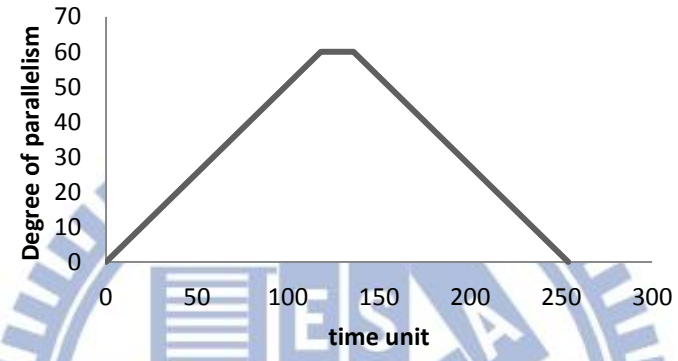


Figure 2-4 The idealize computation execution time and parallelism relationship. The vertical axis is the number of MBs processed in parallel, the horizontal axis is time. The time unit here is time required for deblocking a MB.

The 3D wave-front method [3] is based on the 2D wave-front method, but also uses inter frame parallelism, meaning more MBs can be processed in parallel. This method can significantly enhance the parallelism. In Figure 2-5 [4], the dark gray MBs can be processed in parallel.

The 3D wave-front method is used with the 2D wave-front method. The 2D wave-front method is used for intra frame parallelization, while the 3D wave-front method is used for inter frame parallelization. The 2D wave-front method parallelizes at the MB-level, which is larger than the basic deblocking unit in the H.264 standard. Can deblocking at a finer granularity increase the amount of parallelism, and still be

combined with the 3D wave-front method to further increase the parallelism? We will explain this in the following sections.

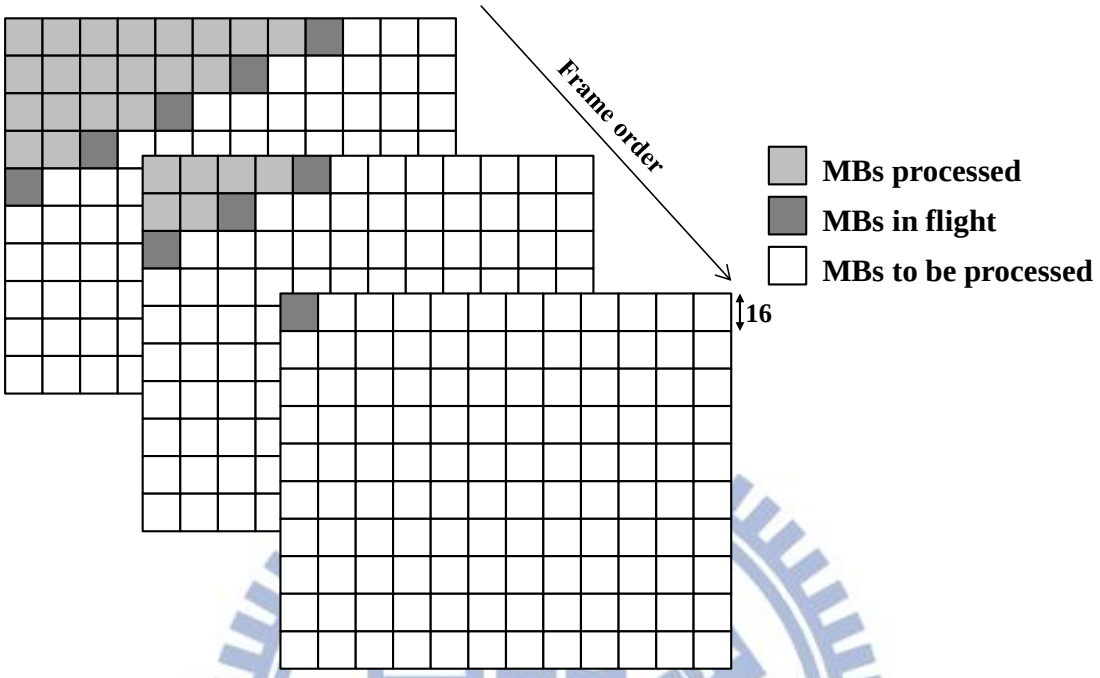


Figure 2-5 The dark gray MBs can be processed in parallel[3].

Chapter 3 Algorithm

3.1 16 Pixel Long Boundary Method

Analyzing applications at a finer granularity usually opens extra opportunities for parallelization. In H.264, the standard defines the order for deblocking using a 16 pixel long boundary as its basic unit. As a result, in this section we analyze the data dependencies within the deblocking filter, and then propose our deblocking order and design. When deblocking a 16 pixel long boundary, in total it will affect eight 4x4 blocks adjacent to the boundary. Figure 3-1(a) shows the affected blocks when we deblock a vertical 16 pixel long boundary, and Figure 3-1(b) shows the case for a horizontal boundary.

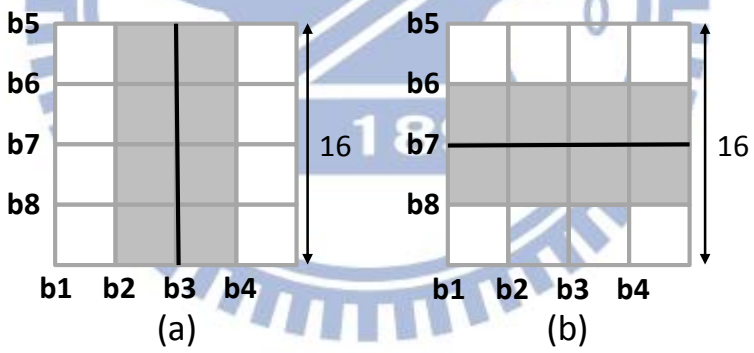


Figure 3-1 Gray blocks are the affected 4x4 blocks when deblocking a (a) vertical and (b) horizontal 16 pixel long boundary

We separate the data dependencies when using a 16 pixel long boundary for deblocking into 3 cases:

Case 1: Intra MB 16 pixel long boundary data dependencies.

In Figure 3-2(a), the result after deblocking MB_{b1} (boundary b1) is input into the deblocking filter for MB_{b2}, with that result then becoming the input into the

deblocking filter for MB_{b3} and so on. Through this analysis the data dependency chain is MB_{b1}⇨MB_{b2}⇨MB_{b3}⇨MB_{b4} and MB_{b5}⇨MB_{b6}⇨MB_{b7}⇨MB_{b8}. Moreover, the deblocking result of MB_{b4} is input to MB_{b5}, so the data dependency chain for intra MB deblocking is MB_{b1}⇨MB_{b2}⇨MB_{b3}⇨MB_{b4}⇨MB_{b5}⇨MB_{b6}⇨MB_{b7}⇨MB_{b8} as shown in Figure 3-2(b).

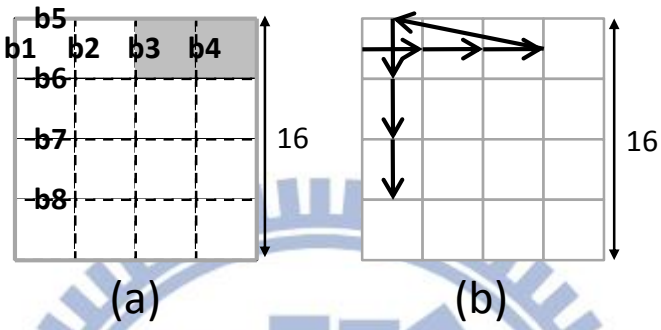


Figure 3-2 (a) Intra MB deblocking execution order, the gray blocks are data dependencies from boundary 4 to boundary 5. (b) The data dependency chain for intra MB deblocking.

Case 2: Same row inter-MB 16 pixel long boundary data dependencies.

In Figure 3-3, part of the deblocking result of Current MB_{b8} (the gray blocks) is the deblocking input to Right MB_{b1}, so Right MB_{b1} depends on Current MB_{b8}. In other words, Right MB_{b1} can begin execution after the Current MB_{b8} has completed execution. This shows that using 16 pixel long boundaries, MBs within the same row cannot be deblocked at the same time.

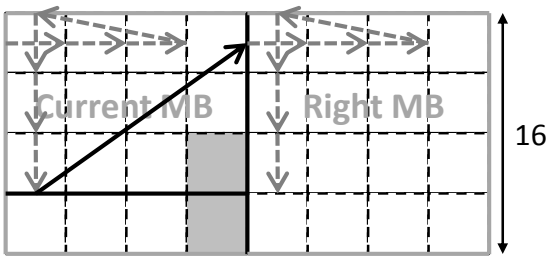


Figure 3-3 The deblocking data dependency chain for MBs in the same row.

Case 3: Adjacent row inter-MB 16 pixel long boundary data dependencies.

In Figure 3-4, the deblocking input of Current MB_{b5} needs 4 4x4 blocks from Upper MB (gray blocks). According to Case 2, we find that the dark gray block is the last to be modified. The dark gray block is modified by deblocking Upper-right MB_{b1} after which it is able to become the deblocking input to Current MB_{b5}. Therefore, Current MB_{b5} depends on Upper-right MB_{b1}, with the data dependency chain shown as a black arrow in Figure 3-4.

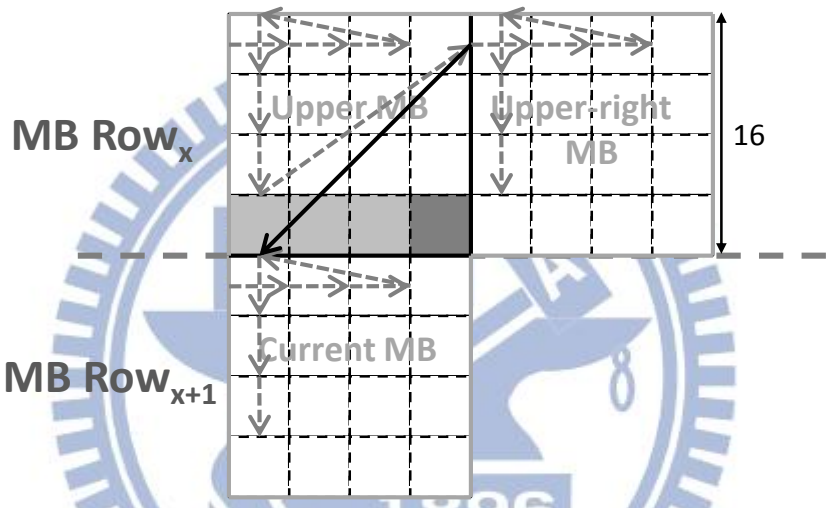


Figure 3-4 The data dependency chain between adjacent rows of MBs.

According to the above 3 cases, we propose a new execution order. This order fulfills the required data dependencies whilst providing an extra degree of deblocking parallelism. The time that deblocking is performed on each 16 pixel long boundary is shown in Figure 3-5. If the time of execution for deblocking Current MB_{b1} is t, by the above Case 2 the execution time of Right MB_{b2} is t+9, by Case 3 the execution time of Lower MB_{b5} is also t+9, and by Case 1 the execution time of Lower MB_{b1} is t+5.

If the time of execution of Current MB_{b1} is t, it shows in Figure 3-6 the execution time of Lower MB_{b1} is t+5 in 16 pixel long boundary method order, and the execution time of Lower MB_{b1} is t+16 in the 2D wave-front method.

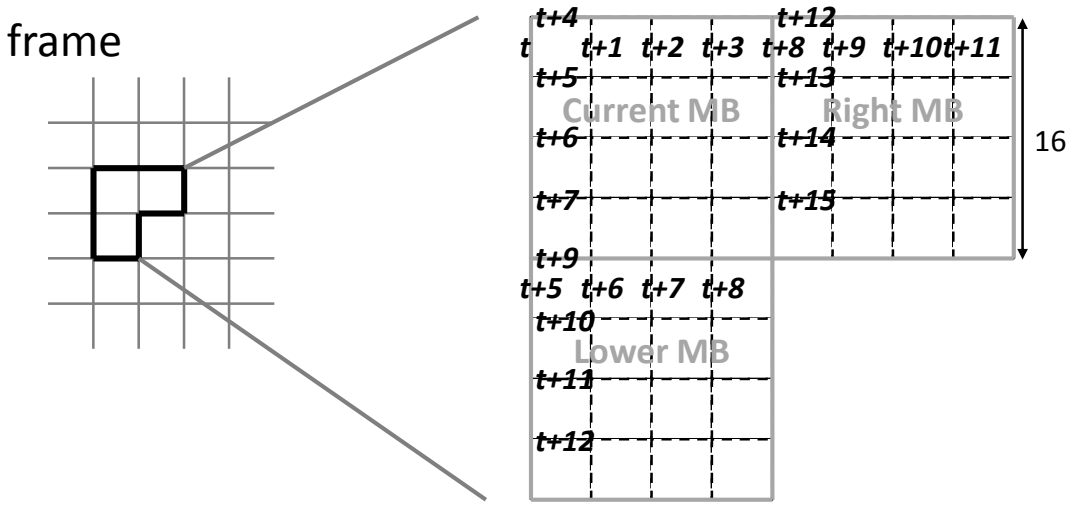


Figure 3-5 Proposed execution order.

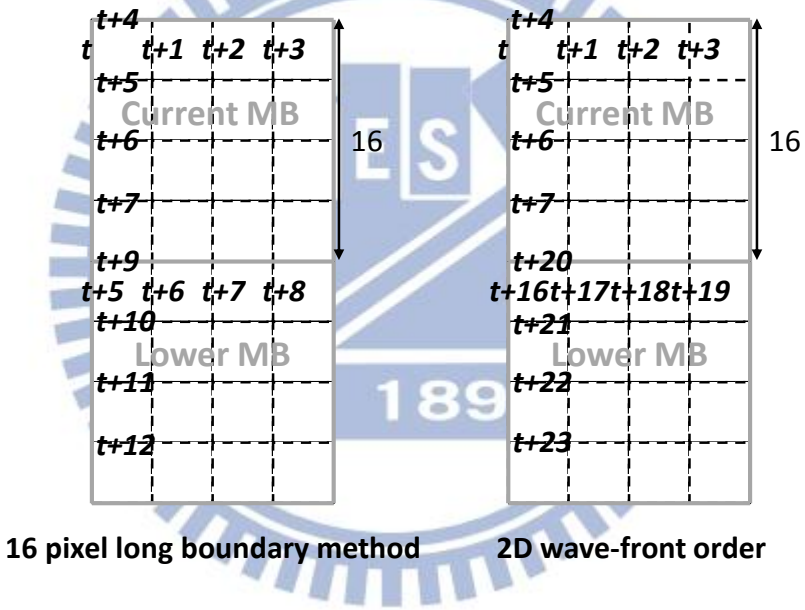


Figure 3-6 Timing difference of the 16 pixel long boundary method order and the 2D wave-front method order.

According to Case 2 mentioned above, when deblocking on 16 pixel long boundaries within the same MB row, MBs cannot be deblocked at the same time. As a result, we can assign one PE to each row of MBs. Due to the relationship between the number of PEs and the aspect of the frame to be deblocked, there are two cases that can occur:

Case I: Degree of parallelism depends on frame aspect

Assuming there are more PEs than needed, the degree of parallelism will be limited only by the frame aspect. While processing 16 pixels horizontally (the width of one MB) takes 8 stages, processing 16 pixels vertically takes only 5 stages in the proposed order. As a result, deblocking of the first row of MBs will finish before starting the last row of MBs, if the number of rows of MBs is less than (8/5) × the number of columns of MBs in a frame. We categorize the effects of frame aspect ratio into the following two situations:

i. *# rows of MBs in frame ≤ 8/5 * # columns of MBs in frame (Degree of parallelism limited by # rows of MBs in frame)*

In this situation, the maximum parallelism is equal to the number of rows of MBs in the frame. Our method has a wind up and wind down time similar to the 2D wave-front method. A diagram is shown in Figure 3-7 to help explain. The upper-left gray region is the starting up of the deblocking filter, and the lower-right gray region is the finishing of the deblocking filter. In these regions, the deblocking filter is not able to reach maximum parallelism. The white region is where the deblocking filter is able to reach maximum parallelism. The degree of parallelism and timing relationship diagram is shown in Figure 3-8.

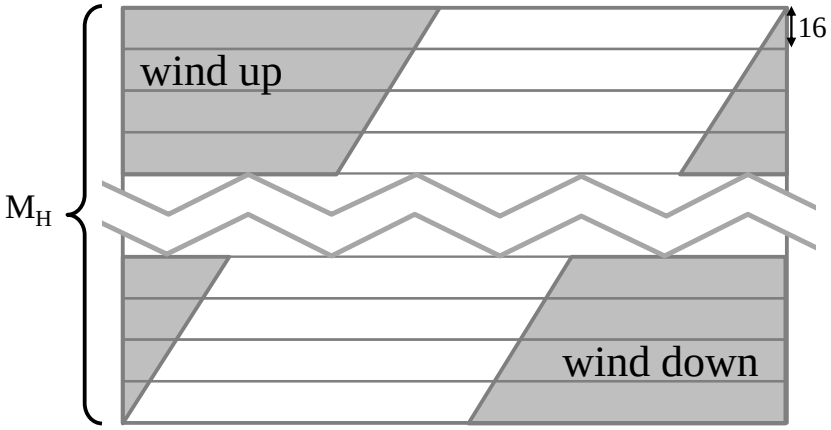


Figure 3-7 Zones of wind up and wind down of deblocking order.

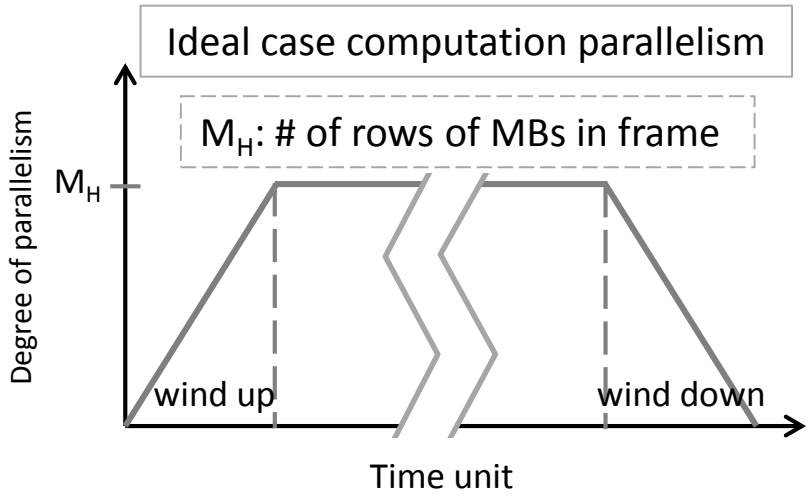


Figure 3-8 The idealize computation execution time and degree of parallelism relationship diagram. The time unit is the time required for deblocking a 16 pixel long boundary.

*ii. # rows of MBs in frame > 8/5 * # columns of MBs in frame (Degree of parallelism limited by # columns of MBs in frame)*

In this situation shown in Figure 3-9(a), the degree of parallelism is equal to the number of rows of MBs that can start their deblocking before the deblocking has completed for the first row of MBs. As explained in the beginning of case I, if the ratio of the height to width is larger than 8/5, the degree of parallelism will be limited by the frame width. The degree of parallelism and timing relationship diagram is shown in Figure 3-9(b).

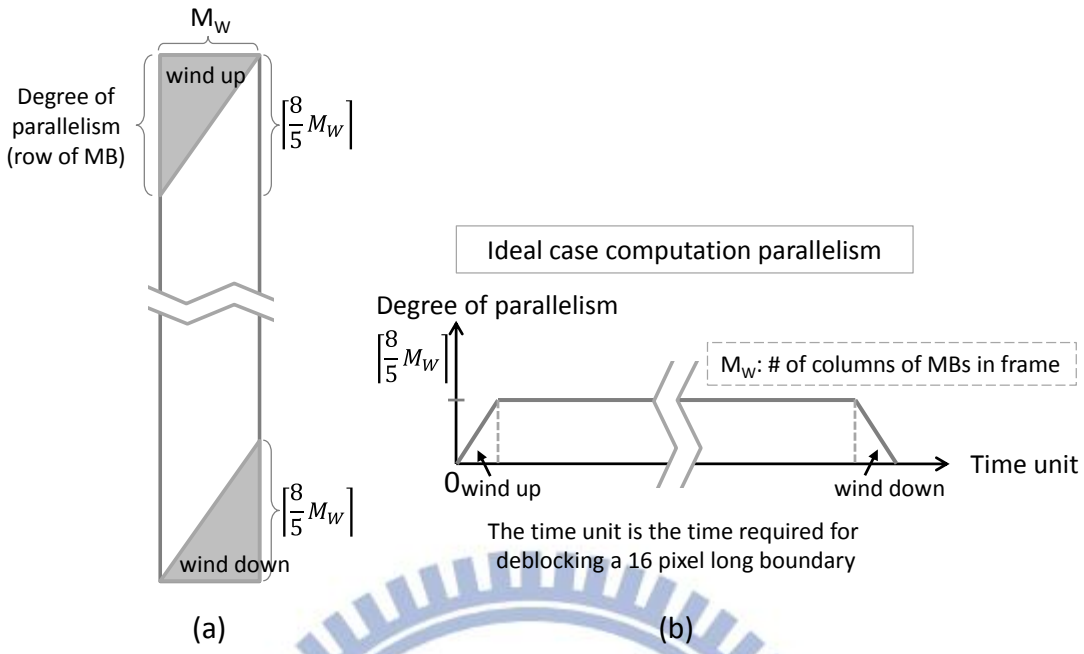


Figure 3-9 (a) The degree of parallelism is limited when the frame height is larger than 8/5 times of the frame width. (b) The idealize computation execution time and degree of parallelism relationship diagram.

Case II: *# of PEs not enough for maximum parallelism.*

In this case, the frame has to be split into multiple stripes for deblocking. Here we first show a naive approach, and then propose an improved one.

Naive approach: In Figure 3-10, assume the number of PEs is K, and then divide the frame into stripes where each stripe contains K rows of MBs. The execution order of the stripes is from top to bottom. We find that each stripe has a wind up and wind down time, meaning PEs remaining idle often occurs. The degree of parallelism and timing diagram is shown in Figure 3-11.

Improved approach: In the naive approach, PEs are frequently idle between the deblocking of stripes as shown in Figure 3-12(a). But after analyzing the details, we find that the execution of the wind down of stripe x and the wind up of stripe x+1 can

be overlapped to fully utilize the PEs. They are able to be overlapped because there are no direct data dependencies between the wind down of stripe x and the wind up of stripe x+1. Therefore the wind up of stripe x+1 can begin execution earlier as shown in Figure 3-12(b). The first row of MBs of stripe x will finish deblocking first after which a PE will become idle, so that PE is then assigned to the first row of MBs of stripe x+1. Continuing this method we find the degree of parallelism and timing as shown in Figure 3-13, showing a reduction in the idle time of PEs.

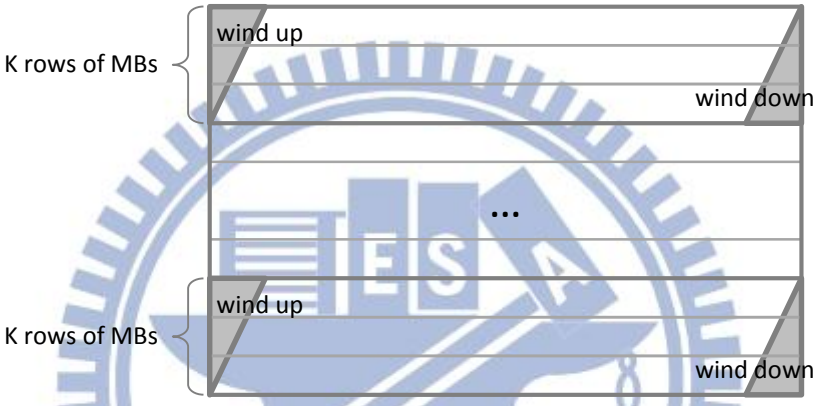


Figure 3-10 # of PEs is not enough for maximum parallelism, frame split into multiple stripes for deblocking.

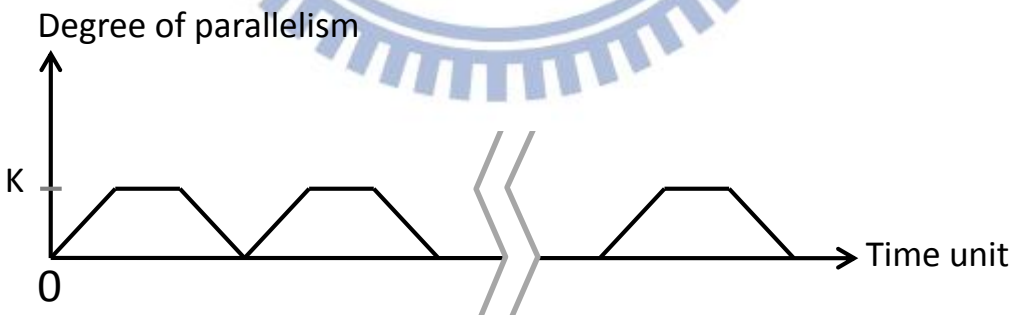


Figure 3-11 The degree of parallelism and timing diagram when the # of PEs is not enough for maximum parallelism.

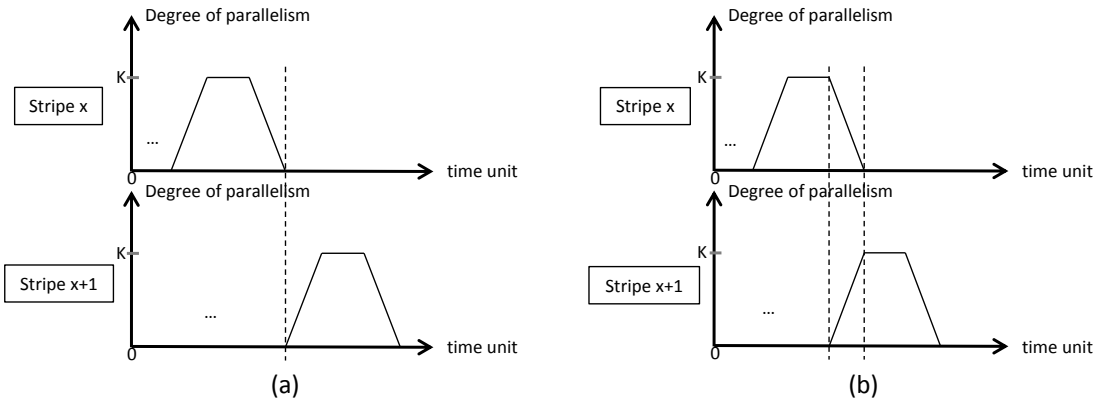


Figure 3-12 (a) The degree of parallelism and timing relationship between stripe x and stripe x+1 before overlapping. (b) The degree of parallelism and timing relationship between stripe x and stripe x+1 after overlapping.

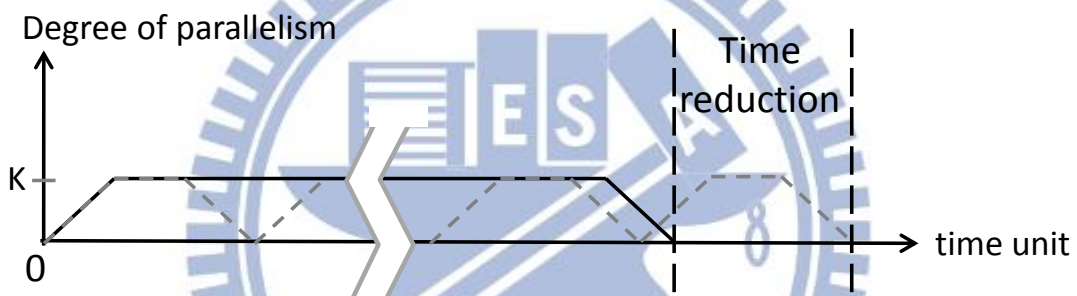


Figure 3-13 The idealize computation execution time and degree of parallelism relationship after overlapping.

In addition, when the number of rows per stripe K does not divide evenly into the total number of MB rows, the final stripe will have a number of idle PEs as shown in Fig. 19.

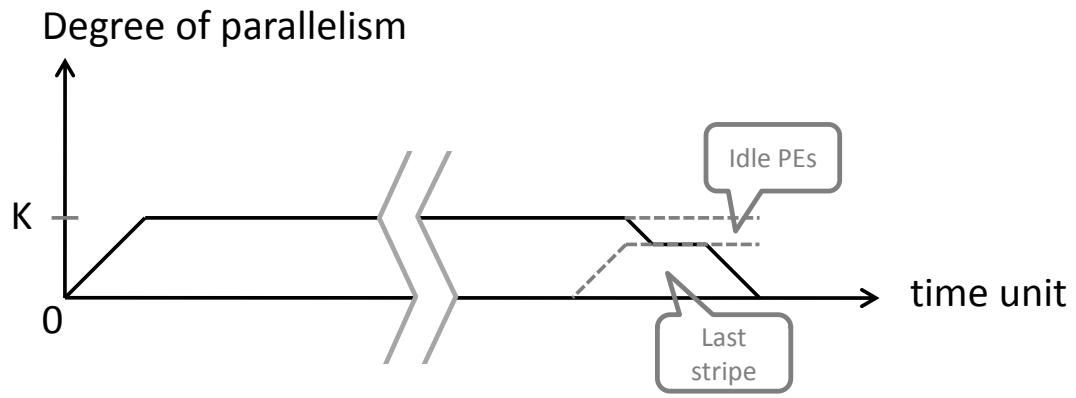


Figure 3-14 The idealize computation execution time and degree of parallelism relationship when the number of rows per stripe K does not divide evenly into the total number of MB rows.

3.2 4 Pixel Long Boundary Method

Analyzing applications at a finer granularity usually opens extra opportunities for parallelization. In H.264, the standard defined orders intersect with each other on a 4*4 grid and split boundaries into 4 pixel long boundaries as its basic units. As a result, we have to analyze the data dependencies and generate the corresponding data dependency chain first. In Figure 3-15(a), we assign IDs (b_1~b_{32}) to 4 pixel long boundaries in a MB. The result after deblocking b_1 is an input into the deblocking filter for b_5, with that result then becoming the inputs into both b_{17} and b_9, and so on. According to the data dependencies caused by the standard order, the data dependency chain for intra MB deblocking is as shown in Figure 3-15 (b)

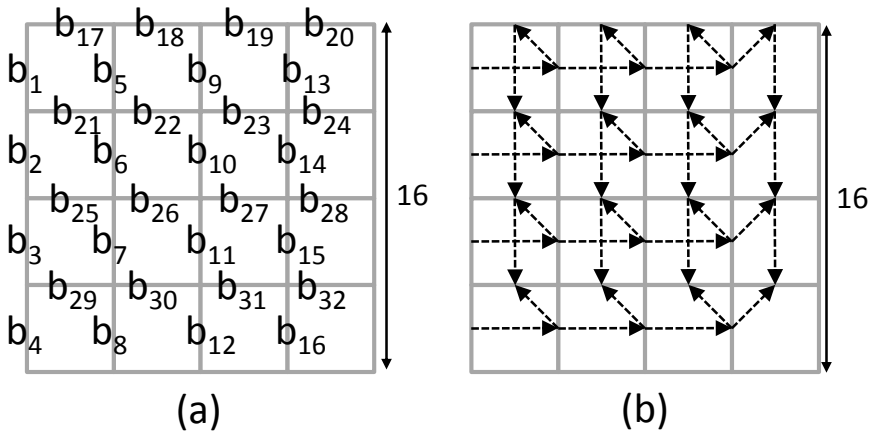


Figure 3-15 (a) ID assignment and (b) data dependency of a MB.

Moreover, we can derive the data dependency tree for intra MB deblocking as shown in Figure 3-16.

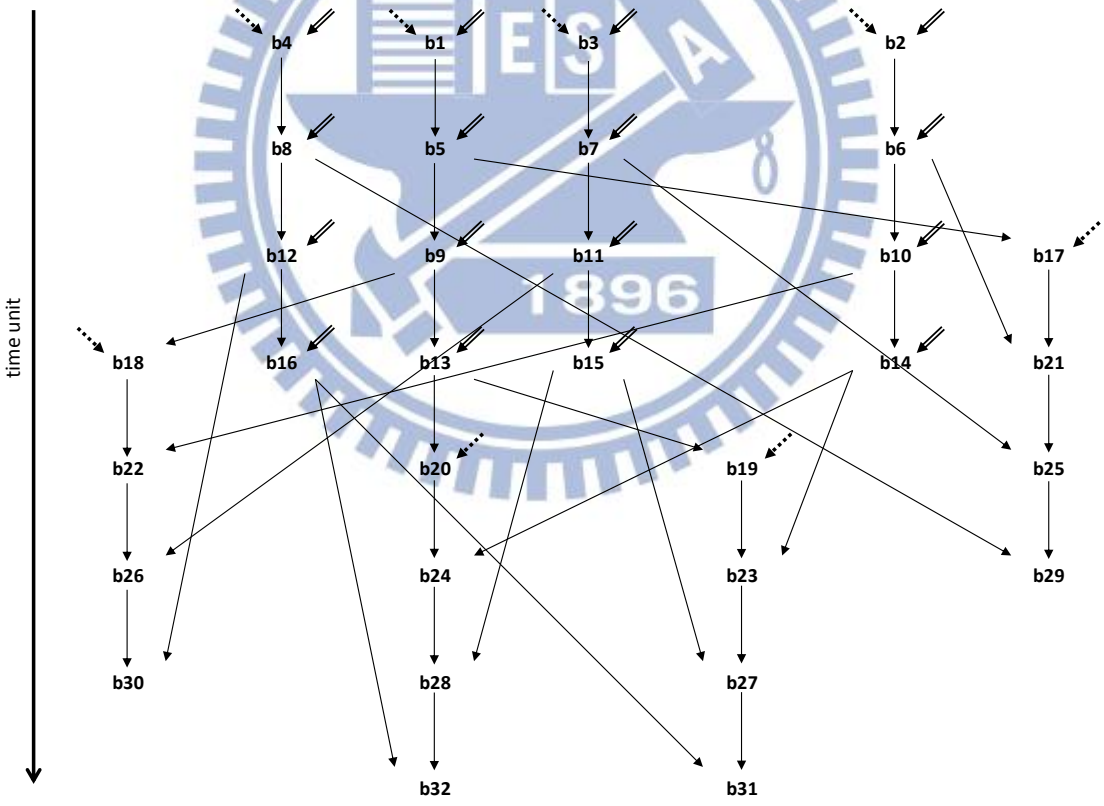


Figure 3-16 Data dependency tree of a MB.

In Figure 3-16, the data dependency tree is represented in 8 timing phases. The timing of each boundary means the earliest timing the deblocking filter can operate.

The black solid arrows mean these input sources are the results from other 4 pixel long boundaries that are in the same MB, the black dotted arrows mean the input sources are the results from other 4 pixel long boundaries that are in different MBs, and the double arrows mean the input sources are used for this first time and come from memory. The complete data dependency tree for a frame can be composed from multiple copies of Figure 3-16 connected with black dotted arrows.

Next, we have to figure out the critical paths of the data dependency tree. When the deblocking filter is on the critical paths, execution should be as soon as possible for best performance. Here, we use following three steps to illustrate the critical paths of a frame.

Step1: Intra MB 4 pixel long boundary critical paths.

Firstly assume the frame contains only one MB. Figure 3-17, which is a trivial derivation from Figure 3-16, shows the critical paths of this frame with arrows representing the data dependency directions. By sorting the counts of arrows from b_5 to each 4 pixel long boundary, we can generate the execution order of the critical paths.

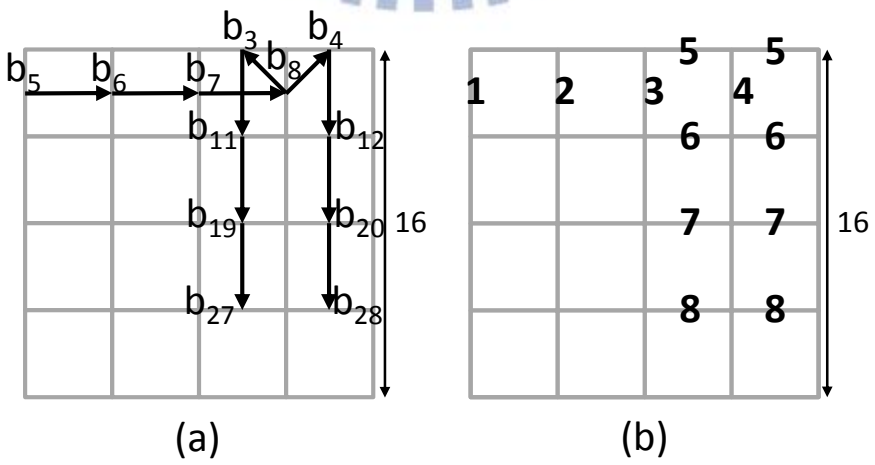


Figure 3-17 (a) Critical paths of frame with only one MB and (b) Deblocking order

on critical paths.

Step 2: Same row of MBs 4 pixel long boundary critical paths.

After step 1, we extend the analyzed frame size to one row of *m* MBs (*m* > 1). Figure 3-18 shows the critical paths of this frame. All arrows compose the critical paths of this frame. The gray arrows (from step 1) are caused by intra MB dependencies. The double arrows are caused by the inter-MB dependencies in a row of MBs. The black dotted arrows are also caused by intra MB dependencies, but added due to the effects of inter-MB dependencies.

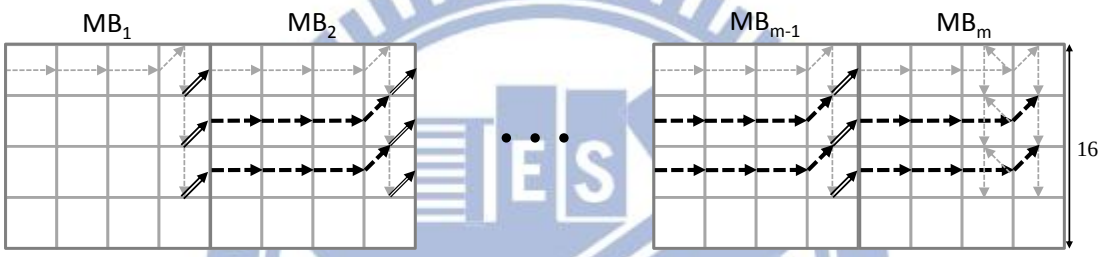


Figure 3-18 Critical paths of frame with only one row of MBs.

To meet the order demanded by the critical paths, we modify the deblocking order of Step 1 as shown in Figure 3-19. The only modification made in Figure 3-19 are the numbers in **bold**, which are on the extra critical paths caused by the inter-MB dependencies in a row of MBs. Though not every critical paths that resides in different MBs is identical, this order fulfills the requirements while keeping regularity. Figure 3-20 shows the order of two adjacent MBs in the same row of MBs. Note that the deblocking of the adjacent right MB starts at time 7, which is before the last operation of the left MB. We will further analyze performance improvements in the Chapter 4.

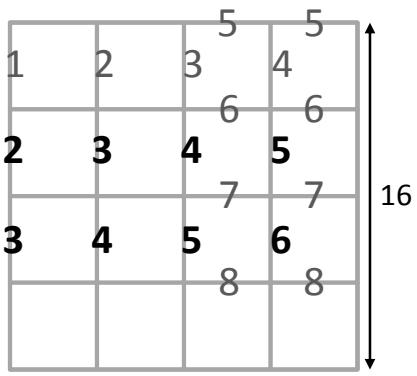


Figure 3-19 Deblocking order fulfills the critical path of a single row of MBs.

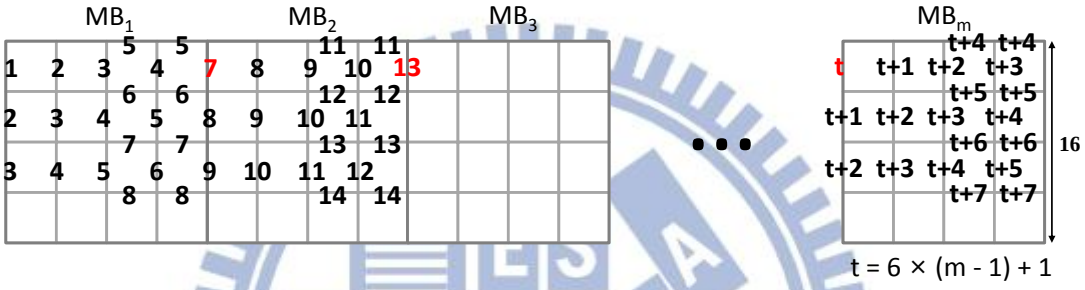


Figure 3-20 Deblocking order of only one row of MBs.

Step 3: Adjacent row inter-MB 4 pixel long boundary critical paths.

In this step, we further extend the size of a frame to *n* rows of *m* MBs (*m* > 1, *n* > 1). Figure 3-21 shows the corresponding critical paths. The gray arrows represent the critical paths caused by both intra MB and inter same row MB data dependencies. The black arrows represent the critical paths caused by inter adjacent rows of MBs. In order to meet the requirements of this critical paths, we provide the deblocking order extended from Step 2 in Figure 3-22.

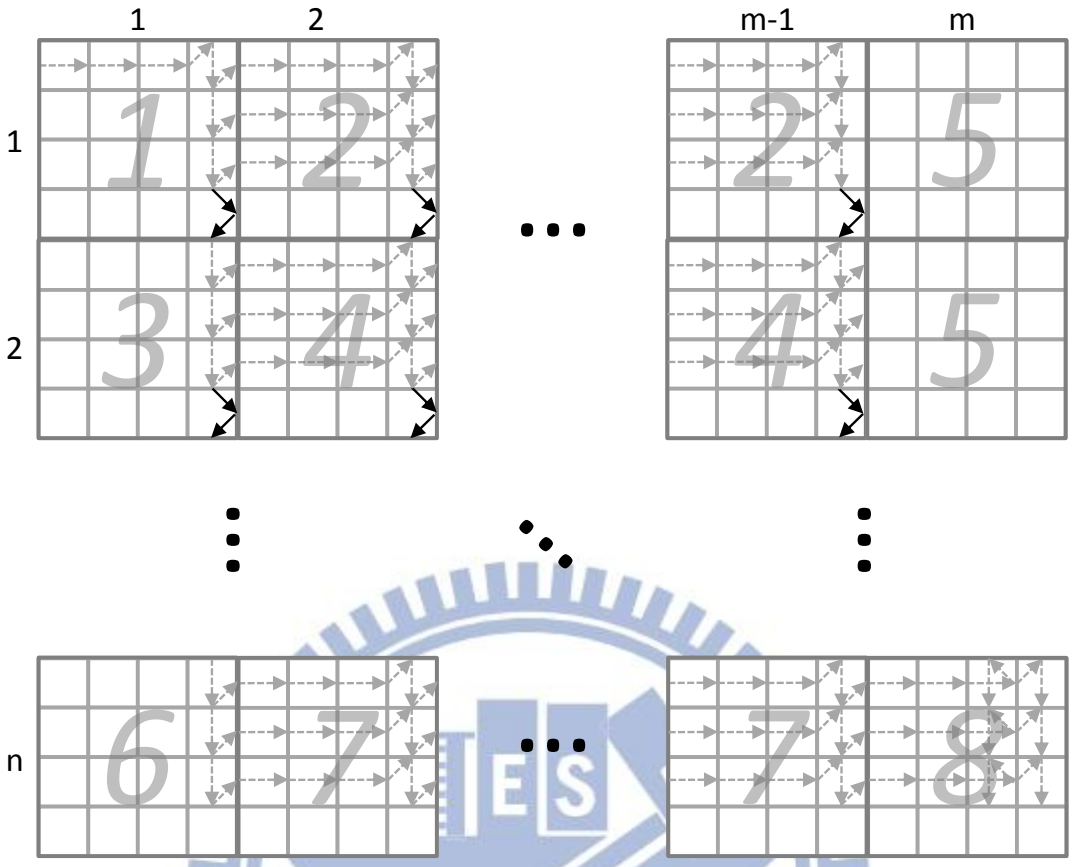


Figure 3-21 Critical paths of frame with m×n MBs.

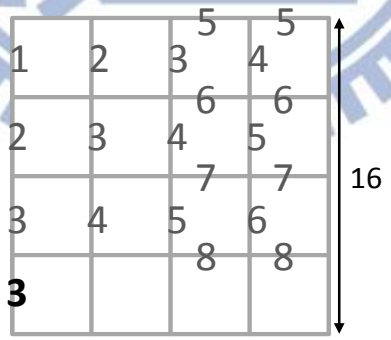


Figure 3-22 Deblocking order that fulfills the critical paths of a frame with m×n MBs.

The only modification made in Figure 3-22 is the number "3" in **bold**, which is on the extra critical paths caused by adjacent rows of MBs. In Figure 3-21, the distribution of critical paths form 8 types of MBs. Figure 3-23 illustrates the

deblocking order of a 3MB×3MB square area, which is the smallest example that contains all 8 types. In Figure 3-23, the gray numbers are the type of MBs. Assume the order start from stage 1, which is the start of critical paths. On the one hand, the 4 pixel long boundaries that labeling the numbers in **bold** are on the critical paths with 3 ×3 MBs frame. On the other hand, the 4 pixel long boundaries with non-bold numbers were labeled by the order of MB proposed in Figure 3-22. We find that the second row of MBs starts deblocking at the 6th stage. Analysis of performance improvements will be discussed in the Chapter 4.

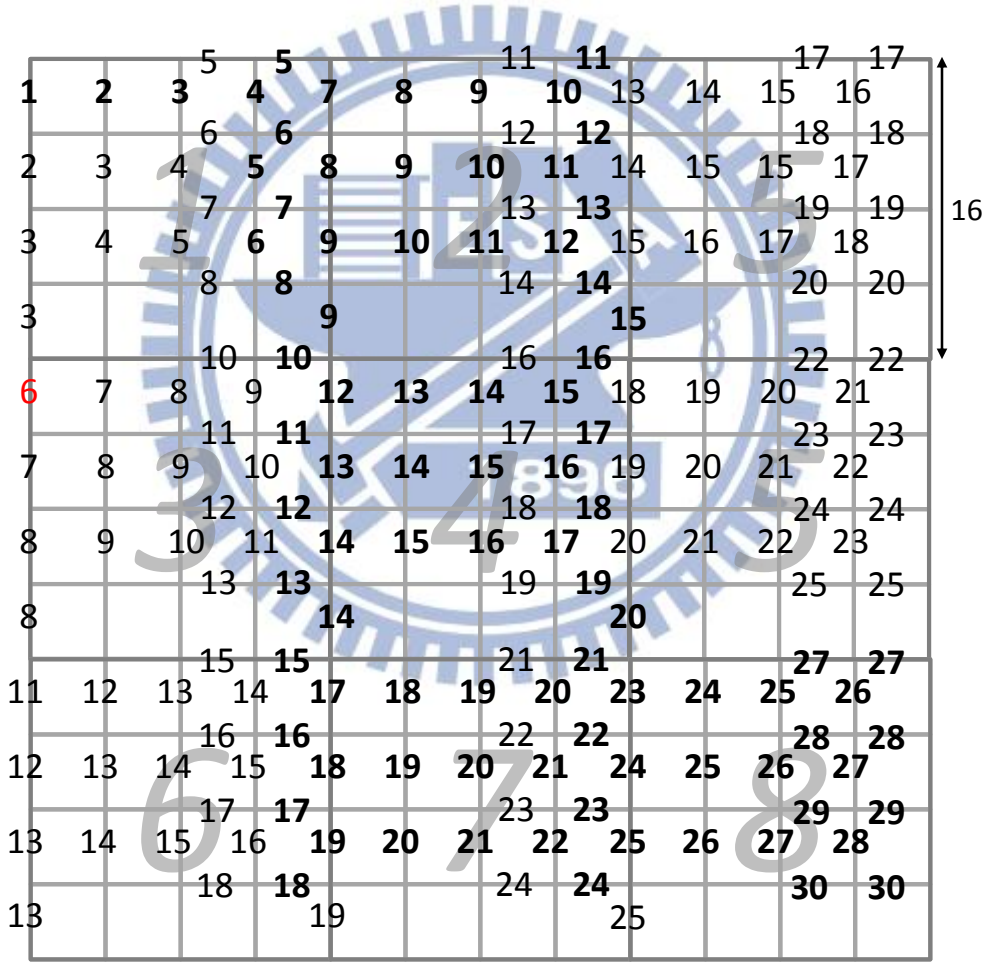


Figure 3-23 Deblocking order of 8 types MB.

The deblocking order in Figure 3-22 fulfills the requirements for correct deblocking of all 4 pixel long boundaries on the critical paths. Next, we decide the

execution order of boundaries that are not on the critical paths. Because these boundaries are not on the critical paths, there is some flexibility in reordering while not increasing the time for deblocking. Figure 3-24 shows all possible orders for 4 pixel long boundaries not on critical paths while not increasing the length of any critical path. These boundaries are categorized into 3 groups. By following arrows in each group, all possible orders can be generated. Taking the group containing b_8, b_{12}, and b_{16} for example, {4,5,6}, {4,5,7}, {4,6,7}, and {5,6,7} are all possible order assignment for $\{b_8,b_{12},b_{16}\}$ in this group.

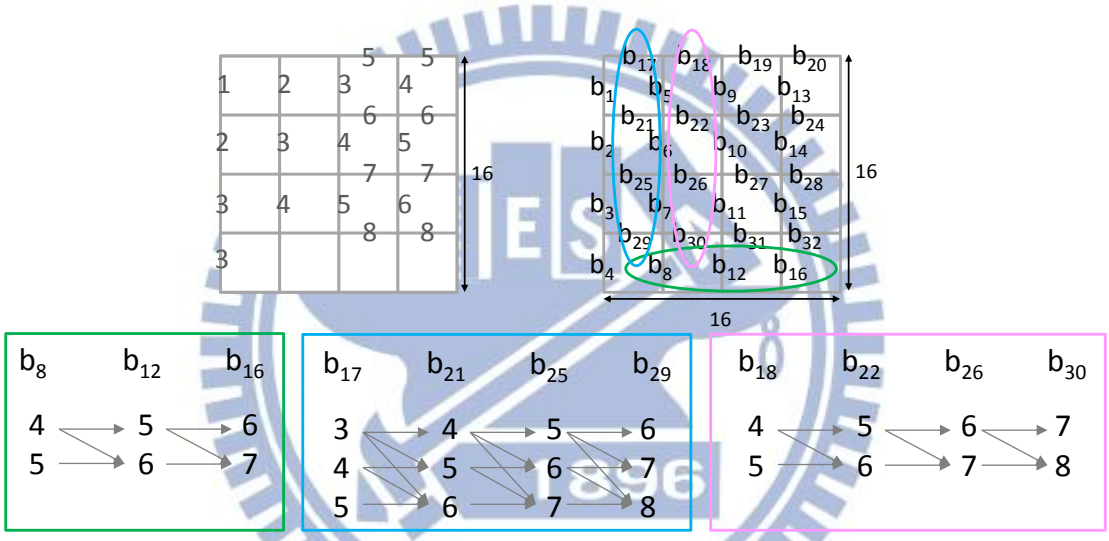


Figure 3-24 Flexible orders on non-critical paths boundaries.

When we consider the minimum required amount of PEs for processing one additional MB row without prolonging the required processing time, the effects of orders for 4 pixel long boundaries not on critical paths should be noticed.

In order to minimize the amount of required PEs, we derive the minimum amount of required PEs for one additional MB row. Deblock one MB need at least 8 time units due to the length of critical path in a single MB. According to the order for 4 pixel long boundaries as shown in Figure 3-22, we find the serial numbers of boundaries that can be deblocked at same time unit for each time unit of deblock a

MB is 12434322 (Figure 3-25(a) black numbers). We can find from Figure 3-24, the possible orders of boundaries that on non-critical paths are at 3rd to 8th time unit for deblock a MB (Figure 3-25(a) red numbers). One MB have 32 boundaries need to be deblocked, so we find the sum of numbers in green block as shown in Figure 3-25(a) is 32. As mention in Figure 3-20, the MB2 deblocked start at 7th time unit of MB1 as shown in Figure 3-25(a). The sum of numbers in blue block as shown in Figure 3-25(a) is 32, we can find that deblock one MB row must process 32 boundaries in 6 time units. So deblock one MB row at least required ceiling(16/3) PEs, and deblock *n* MB row at least required ceiling(16*n*/3) PEs. Deblock one MB row at least required 6 PEs, required 5 PEs for 2nd additional MB row, and required 5 PEs for 3rd additional MB row.

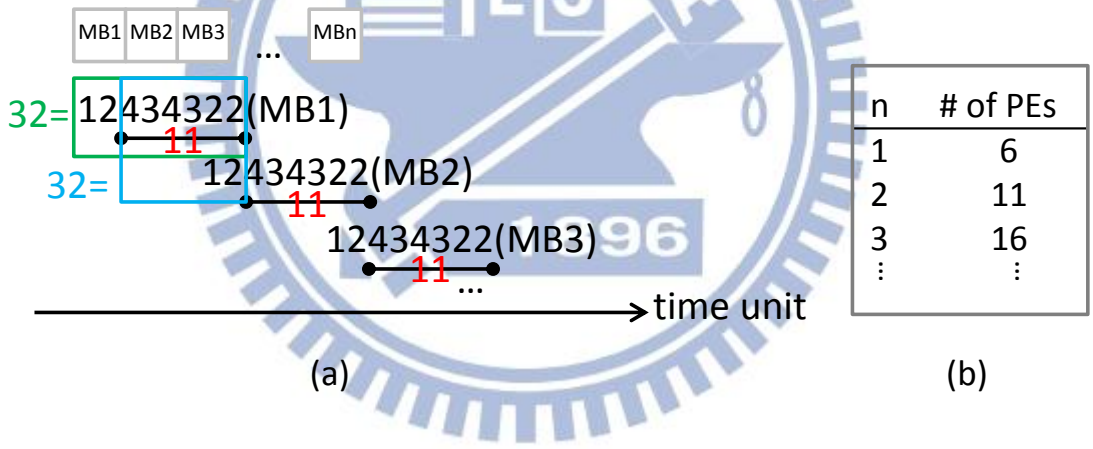


Figure 3-25 (a) The minimum amount of required PEs for one MB row. (b) The amount of required PEs for one additional MB row.

As the result of minimum amount of required PEs for one MB row, we proposed a order as shown in Figure 3-26. It is satisfy the minimum amount of required PEs for one MB row.

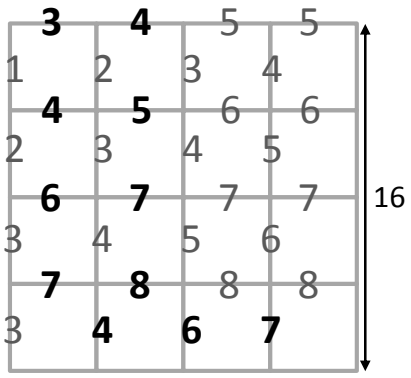


Figure 3-26 Proposed deblocking order

In Figure 3-27 shows the regularity in the number sequence that the number of boundaries in one MB row able to be deblocked in parallel: 12(565565)*565553. As mention in Figure 3-23, deblocking of one MB row can start with a delay of 5 time units to its upper adjacent MB row. Figure 3-27 shows the number of PEs required with number of MBs raising up. It is clear that the proposed order meets the rule of minimum amount of required PEs mention in Figure 3-25. Moreover, every 3 rows of MBs regularly provide 16 4 pixel long boundaries that can be deblocked in parallel in every time unit after a build-up time.

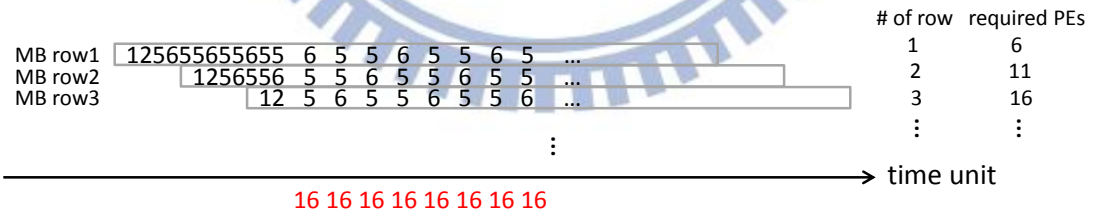


Figure 3-27 The number of PEs required with number of MBs raising up

Due to the relationship between the number of PEs and the aspect of the frame to be deblocked there are two cases that can occur:

Case I: Degree of parallelism depends on frame aspect

Assuming there are more PEs than needed, the degree of parallelism will be

limited only by the frame aspect. While processing 16 pixels horizontally (the width of one MB) takes 6 stages, processing 16 pixels vertically takes only 5 stages in the proposed order. As a result, deblocking of the first row of MBs will finish before starting the last row of MBs, if the number of rows of MBs is less than (6/5) × (the number of columns of MBs in a frame). We categorize the effects of frame aspect ratio into the following two situations:

i. *# rows of MBs in frame ≤ 6/5 * # columns of MBs in frame (Degree of parallelism limited by # rows of MBs in frame)*

In this situation, the maximum parallelism is proportional to the number of rows of MBs in the frame. According to the deblocking order in Figure 3-29, the maximum parallelism of 3 rows of MBs is 16. We can find the maximum parallelism of one row of MBs is ceiling(16/3). For example, the maximum parallelism of one row of MBs is 6, the maximum parallelism of two rows of MBs is 11, and the maximum parallelism of three rows of MBs is 16. Our method has a wind up and wind down time similar to the 2D wave-front method. A diagram is shown in Figure 3-28 to help explain. The upper-left gray region is the starting up of the deblocking filter, and the lower-right gray region is the finishing of the deblocking filter. In these regions, the deblocking filter is not able to reach maximum parallelism. The white region is where the deblocking filter is able to reach maximum parallelism. The degree of parallelism and timing relationship diagram is shown in Figure 3-29.

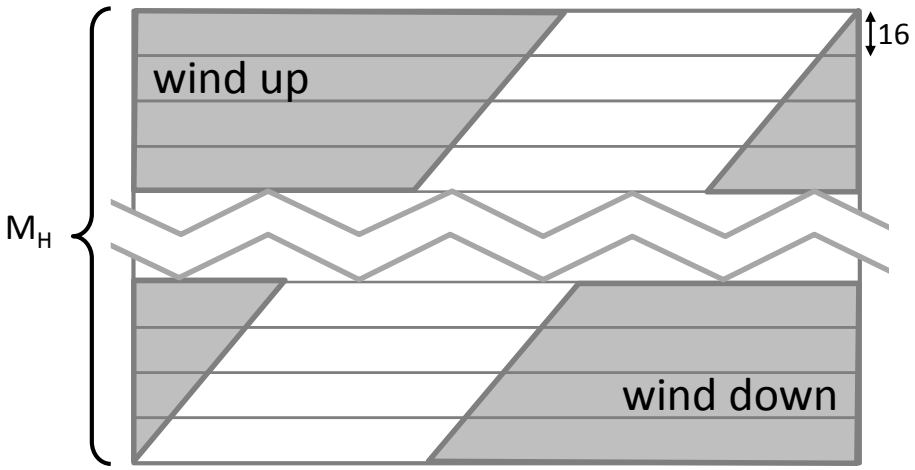


Figure 3-28 Zones of wind up and wind down of deblocking order.

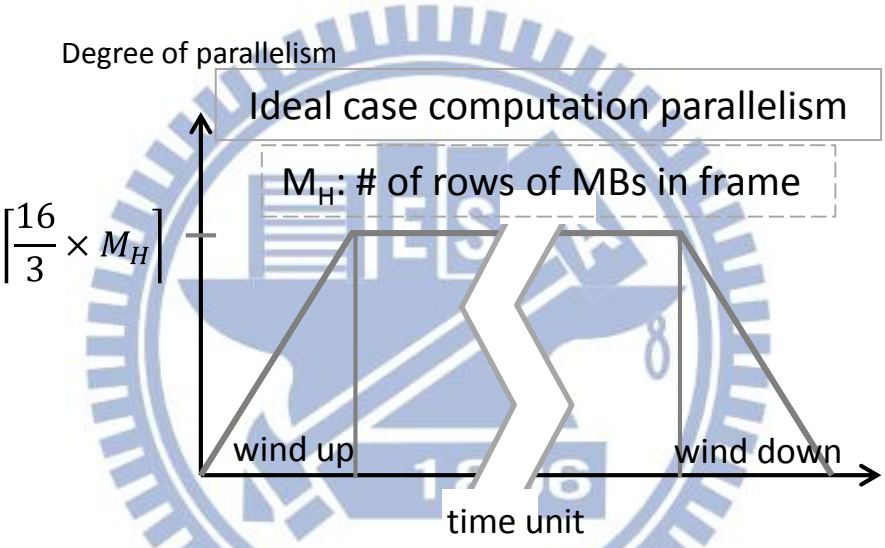


Figure 3-29 The idealize computation execution time and degree of parallelism relationship diagram. The time unit is the time required for deblocking a 4 pixel long boundary.

ii. # rows of MBs in frame > 6/5 * # columns of MBs in frame (Degree of parallelism limited by # columns of MBs in frame)

In this situation, shown in Figure 3-30(a), the degree of parallelism is equal to ceiling((16/3) multiplied by the number of rows of MBs that can start their deblocking before the deblocking has completed for the first row of MBs). As explained in the beginning of case I, if the ratio of the height to width is larger than 6/5, the degree of

parallelism will be limited by the frame width. The degree of parallelism and timing relationship diagram is shown in Figure 3-30(b).

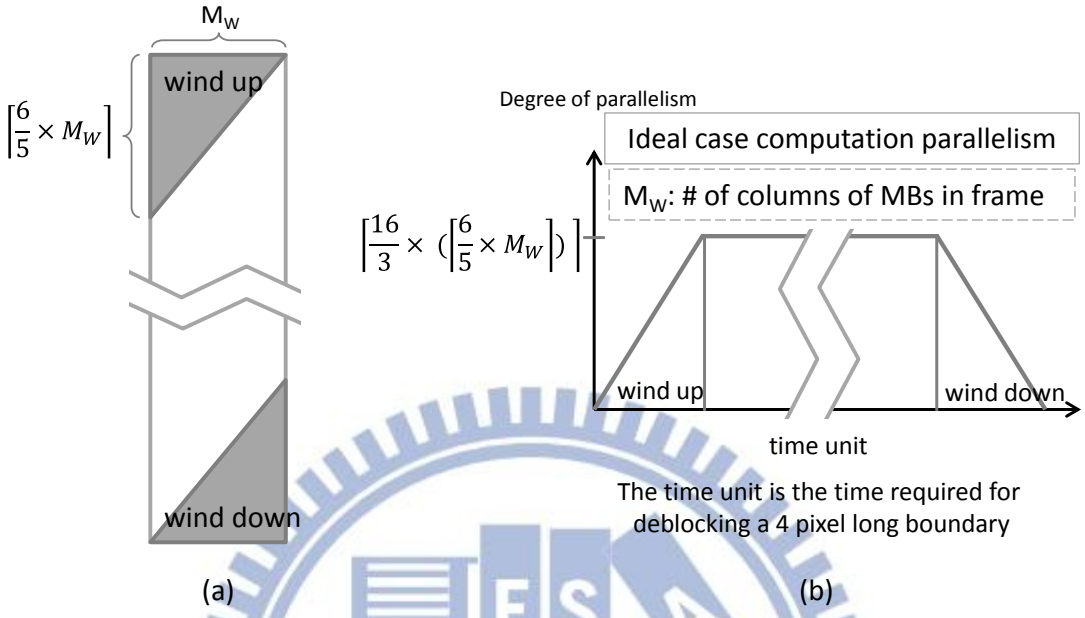


Figure 3-30 (a) The degree of parallelism is limited when the frame height it larger than 6/5 times of the frame width. (b) The idealize computation execution time and degree of parallelism relationship diagram.

Case II: # of PEs not enough for maximum parallelism.

In this case, the frame has to be split into multiple stripes for deblocking. Here we first show a naive approach, and then propose an improved one.

Naive approach: In Figure 3-31, assume the number of PEs is able to deblock only K rows of MBs at the same time, and K is less than the maximum number of parallelizable MB rows. Then divide the frame into stripes where each stripe contains K rows of MBs. The execution order of the stripes is from top to bottom. We find that each stripe has a wind up and wind down time, meaning PEs remaining idle often occurs. The degree of parallelism and timing diagram is shown in Figure 3-32.

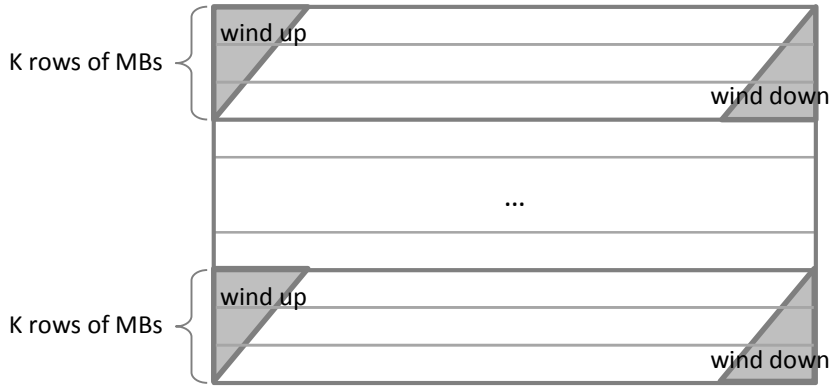


Figure 3-31 # of PEs is not enough for maximum parallelism, frame split into multiple stripes for deblocking.

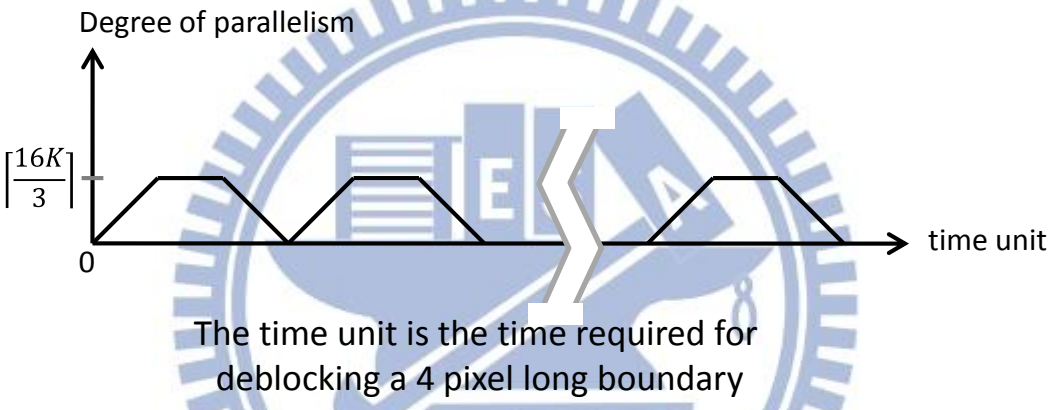


Figure 3-32 The degree of parallelism and timing diagram when the # of PEs is not enough for maximum parallelism.

Improved approach: In the naive approach, PEs are frequently idle between the deblocking of stripes as shown in Figure 3-34(a). But after analyzing the details, we find that the execution of the wind down of stripe x and the wind up of stripe x+1 can be overlapped to fully utilize the PEs. The only reason that the first MB row of stripe x+1 has not started deblocking is because there are not enough PEs. Once the PEs are idle, the first MB row of stripe x+1 can start, and shortly after the other MB rows of stripe x+1. Figure 3-33 shows how the first MB row of stripe x+1 can start deblocking at the last two stages of deblocking the first MB row of stripe x as an example. Using this approach, the wind up of stripe x+1 can begin execution earlier as shown in

Figure 3-34(b). Moreover, we keep the regularity of the amount of required PEs for both stripes.

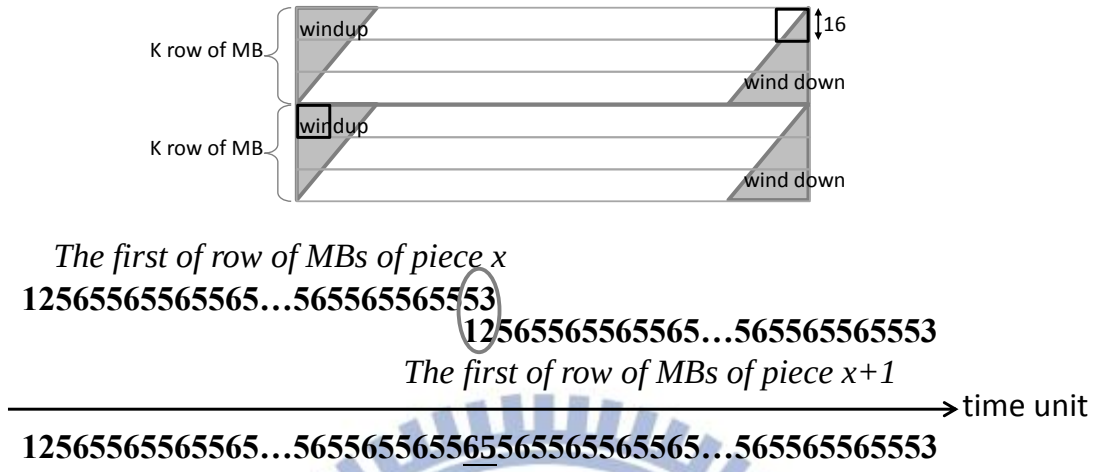


Figure 3-33 PE assignment for the first MB row of both stripe x and x+1

Applying this method we find the degree of parallelism and timing as shown in Figure 3-35, showing a reduction in the idle time of PEs.

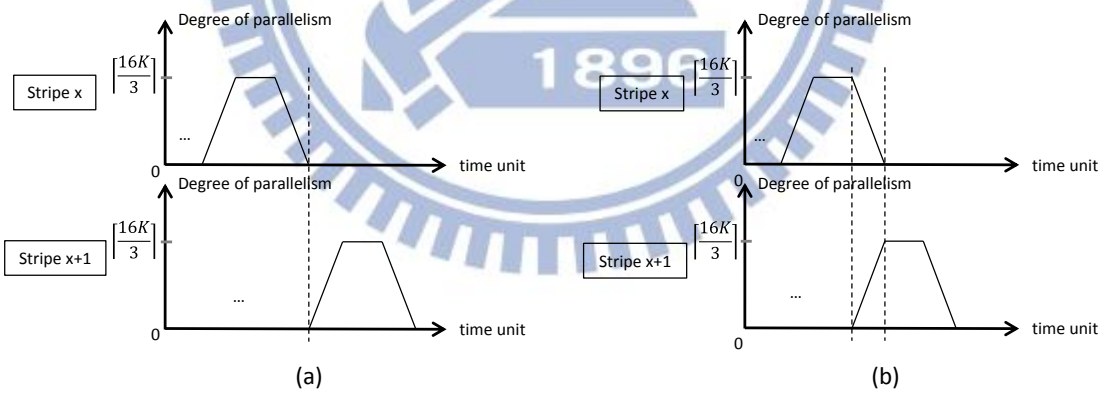


Figure 3-34 (a) The degree of parallelism and timing relationship between stripe x and stripe x+1 before overlapping. (b) The degree of parallelism and timing relationship between stripe x and stripe x+1 after overlapping.

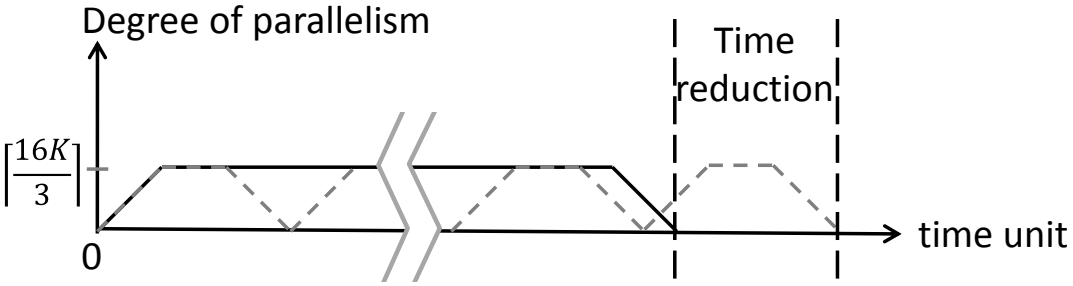


Figure 3-35 The idealize computation execution time and degree of parallelism relationship after overlapping.

Chapter 4 Comparison

The proposed order has been shown in the previous chapters, so the focus of this section is on determining the degree to which parallelism and execution time can be improved from this design. In this chapter, we first model the parallelism and time of deblocking a frame for both the proposed order and 2D wave-front method order. Then we show the benefit of overlapping, and then we construct two figures to show the effects of the number of PEs and the benefits from overlapping the deblocking of adjacent rows of MBs. After that, the time required will be compared using three representative examples. In the end, we explain that our design is also complementary to the 3D wave-front method.

4.1 Equations

The proposed execution order's maximum parallelism and required wind up time, wind down time and execution time can be expressed by the equations:

4.1.1 4 pixel long boundary method

Maximum # of rows of MBs that can be deblocked in parallel (K)

$= Min\left(M_H, \left\lceil \frac{6}{5} M_W \right\rceil, \left\lfloor \frac{3 \times \#\ of\ PEs}{16} \right\rfloor\right)$ (3)

$Maximum\ parallelism(P)$

$= \left\lceil K \times \frac{16}{3} \right\rceil$ (4)

$Wind\ up\ time$

= *Time to reach the row of maximum parallelism + Time to reach the maximum parallelism in that row*

= (*delay between processing rows*) × (*Maximum # of rows of MBs that can be deblocked in parallel* $- 1$) $+$ *(Time to reach the maximum parallelism in the* K*th row)*

$= 5 \times (K - 1) + 2$

$= 5 \times K - 3$ (5)

$$Wind\ down\ time$$

$$= \begin{cases} Time\ after\ finishing\ the\ last\ \mathrm{K}th\ row\ of\ MBs. & ,if\ enough\ PEs \\ Time\ after\ finishing\ 1st\ row\ of\ MBs\ in\ last\ stripe. & ,if\ limited\ PEs \end{cases}$$

$$= \begin{cases} (delay\ between\ processing\ rows) \times (Maximum\ \#\ of\ rows\ of\ MBs\ that\ can\ be\ deblocked\ in\ parallel\ -1) \\ +\ (Time\ after\ maximum\ parallelism\ in\ the\ last\ Kth\ row\ of\ MBs) & ,if\ \#of\ PEs \geq \left\lceil \frac{16}{3} \times Min\left(\left\lceil \frac{6}{5} M_W \right\rceil, M_H\right) \right\rceil \\ (delay\ between\ processing\ rows) \times (\#\ of\ rows\ in\ last\ stripe - 1) \\ +\ (Time\ after\ maximum\ parallelism\ in\ the\ 1th\ row\ of\ MBs\ in\ last\ stripe) & ,if\ \#of\ PEs < \left\lceil \frac{16}{3} \times Min\left(\left\lceil \frac{6}{5} M_W \right\rceil, M_H\right) \right\rceil \end{cases}$$

$$= \begin{cases} 5 \times (K-1) + 2 & ,if\ \#of\ PEs \geq \left\lceil \frac{16}{3} \times Min\left(\left\lceil \frac{6}{5} M_W \right\rceil, M_H\right) \right\rceil\ or\ \#of\ rows\ in\ last\ stripe = P \\ 5 \times \left((M_H\ mod\ K) - 1\right) + 2 & ,otherwise \end{cases} \quad (6)$$

$$Total\ Execution\ time$$

$$= \begin{cases} Time\ before\ last\ row\ of\ MBs\ starts + Time\ to\ finish\ last\ row\ of\ MBs & ,if\ enough\ PEs \\ (Time\ to\ finish\ one\ row\ of\ MBs) \times (\#\ of\ pieces) + Wind\ down\ time\ of\ last\ stripe & ,if\ limited\ PEs \end{cases}$$

$$= \begin{cases} 2 + 5 \times (M_H - 1) + 6 \times M_W & ,if\ \#of\ PEs \geq \left\lceil \frac{16}{3} \times Min\left(\left\lceil \frac{6}{5} M_W \right\rceil, M_H\right) \right\rceil \\ 6 \times M_W \times \left\lceil \frac{M_H}{K} \right\rceil + 5 \times (K-1) + 2 & ,if\ \#of\ PEs < \left\lceil \frac{16}{3} \times Min\left(\left\lceil \frac{6}{5} M_W \right\rceil, M_H\right) \right\rceil\ and\ \#of\ rows\ in\ last\ stripe = K \\ 6 \times M_W \times \left\lceil \frac{M_H}{K} \right\rceil + 5 \times \left((M_H\ mod\ K) - 1\right) + 2 & ,if\ \#of\ PEs < \left\lceil \frac{16}{3} \times Min\left(\left\lceil \frac{6}{5} M_W \right\rceil, M_H\right) \right\rceil\ and\ \#of\ rows\ in\ last\ stripe \neq K \end{cases} \quad (7)$$

The time unit is the time required for deblocking a 4 pixel long boundary. The parallelism unit is a 4 pixel long boundary.

4.1.2 16 pixel long boundary method

In order to compare the 4 pixel long boundary method with the 16 pixel long boundary method, we modify the original equations. First, taking the number of PEs into consideration; and second, adjusting the parallelism and the time unit. Each 16 pixel long boundary contains four 4 pixel long boundaries that can be deblocked in parallel. So assuming the computation power of all PEs are the same, we multiply the time by 1 and the parallelism by 4. The following equations show the modified equations for the 16 pixel long boundary method:

$$Maximum\ parallelism(P)$$

= Maximum # of rows of MBs that can be deblocked in parallel

= *Min (# of rows of MBs, # of parallel deblocked rows of MBs when limited by the width of frame, # of available PEs)*

$$= 4 \times Min\left(M_H, \left\lceil \frac{8}{5} M_W \right\rceil, \#\ of\ PEs\right) \quad (8)$$

Wind up time

= *Time to reach the row of maximum parallelism*

= (*delay between processing rows*) × (*maximum parallelism* - 1)

$$= 5 \times (P - 1) \quad (9)$$

Wind down time

$$= \begin{cases} Time\ after\ finishing\ the\ last\ \mathrm{P}th\ row\ of\ MBs. & ,if\ enough\ PEs \\ Time\ after\ finishing\ 1st\ row\ of\ MBs\ in\ last\ stripe. & ,if\ limited\ PEs \end{cases}$$

$$= \begin{cases} (delay\ between\ processing\ rows) \times (maximum\ parallelism - 1) & ,if\ \#of\ PEs \geq Min(M_H, \left\lceil \frac{8}{5} M_W \right\rceil) \\ (delay\ between\ processing\ rows) \times (\#\ of\ rows\ in\ last\ stripe - 1) & ,if\ \#of\ PEs < Min(M_H, \left\lceil \frac{8}{5} M_W \right\rceil) \end{cases}$$

$$= \begin{cases} 5 \times (P - 1) & ,if\ \#of\ PEs \geq Min(M_H, \left\lceil \frac{8}{5} M_W \right\rceil)\ or\ \#of\ rows\ in\ last\ stripe = P \\ 5 \times \left((M_H\ mod\ P) - 1\right) & ,otherwise \end{cases} \quad (10)$$

$$Total\ Execution\ time$$

$$= \begin{cases} Time\ before\ last\ row\ of\ MBs\ starts + Time\ to\ finish\ last\ row\ of\ MBs & ,if\ enough\ PEs \\ (Time\ to\ finish\ one\ row\ of\ MBs) \times (\#\ of\ pieces) + Wind\ down\ time\ of\ last\ stripe & ,if\ limited\ PEs \end{cases}$$

$$= \begin{cases} 5 \times (M_H - 1) + 8 \times M_W & ,if\ \#of\ PEs \geq Min\left(M_H, \left\lceil \frac{8}{5} M_W \right\rceil\right) \\ 8 \times M_W \times \left\lceil \frac{M_H}{P} \right\rceil + 5 \times (P - 1) & ,if\ \#of\ PEs < Min\left(M_H, \left\lceil \frac{8}{5} M_W \right\rceil\right)\ and\ \#of\ rows\ in\ last\ stripe = P \\ 8 \times M_W \times \left\lceil \frac{M_H}{P} \right\rceil + 5 \times \left((M_H\ mod\ P) - 1\right) & ,if\ \#of\ PEs < Min\left(M_H, \left\lceil \frac{8}{5} M_W \right\rceil\right)\ and\ \#of\ rows\ in\ last\ stripe \neq P \end{cases} \quad (11)$$

4.1.3 2D wave-front method

In order to compare the 4 pixel long boundary method with the 2D wave-front method, we modify the original equations. First, taking the number of PEs into consideration; and second, adjusting the parallelism and the time unit. Deblocking one MB with the 2D wave-front method is done by deblocking eight 16 pixel long boundaries consecutively. Moreover, each 16 pixel long boundary contains four 4 pixel long boundaries that can be deblocked in parallel. So assuming the computation

power of all PEs are the same, we multiply the time by 8 and the parallelism by 4. The following equations show the modified equations for the 2D wave-front method:

$$Maximum\ parallelism(P) = 4 \times Min\left(M_H, \left\lceil\frac{1}{2}M_W\right\rceil, \#\ of\ PEs\right) \quad (12)$$

$$Wind\ up\ time = 8 \times 2 \times (P-1) \quad (13)$$

$$Wind\ down\ time$$

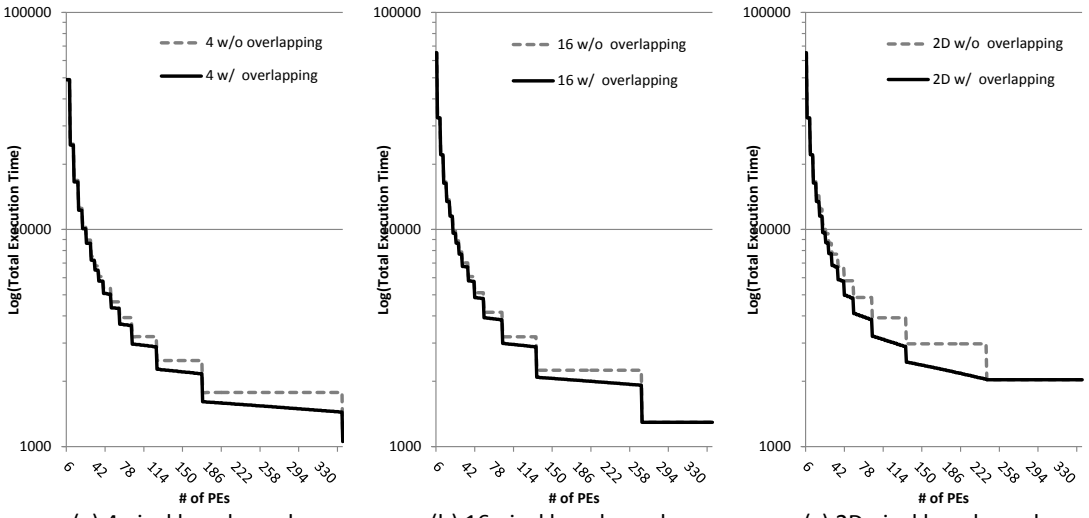
$$= \begin{cases} 8 \times 2 \times (P-1) & ,if\ \#of\ PEs \geq Min(M_H, \left\lceil\frac{1}{2}M_W\right\rceil)\ or\ \#of\ rows\ in\ last\ stripe = P \\ 8 \times 2 \times \left((M_H\ mod\ P) - 1\right) & ,otherwise \end{cases} \quad (14)$$

$$Total\ Execution\ time$$

$$= \begin{cases} 8 \times 2 \times (M_H - 1) + 8 \times M_W & ,if\ \#of\ PEs \geq Min\left(M_H, \left\lceil\frac{1}{2}M_W\right\rceil\right) \\ 8 \times M_W \times \left\lceil\frac{M_H}{P}\right\rceil + 8 \times 2 \times (P-1) & ,if\ \#of\ PEs < Min\left(M_H, \left\lceil\frac{1}{2}M_W\right\rceil\right) and\ \#of\ rows\ in\ last\ stripe = P \\ 8 \times M_W \times \left\lceil\frac{M_H}{P}\right\rceil + 8 \times 2 \times \left((M_H\ mod\ P) - 1\right) & ,if\ \#of\ PEs < Min\left(M_H, \left\lceil\frac{1}{2}M_W\right\rceil\right) and\ \#of\ rows\ in\ last\ stripe \neq P \end{cases} \quad (15)$$

4.2 Overlapping benefit

In Figure 4-1, we can find overlapping is beneficial for all methods. Overlapping is actually more useful for the 2D wave-front method because it has a longer wind up and wind down time. In following, we will compare these three methods using the results with overlapping.



(a) 4 pixel long boundary (b) 16 pixel long boundary (c) 2D pixel long boundary

Figure 4-1 Overlapping benefit for proposed methods and 2D wave-front method.

4.3 Comparing different granularities

Figure 4-2 shows that for all the 2D wave-front method and two proposed methods, the greater the number of PEs, the greater the benefit to the time to deblock a frame. However, our proposed method gains more benefit than the 2D wave-front method which comes from the shorter wind up and wind down time requirement, especially for the vertically shaped frame.

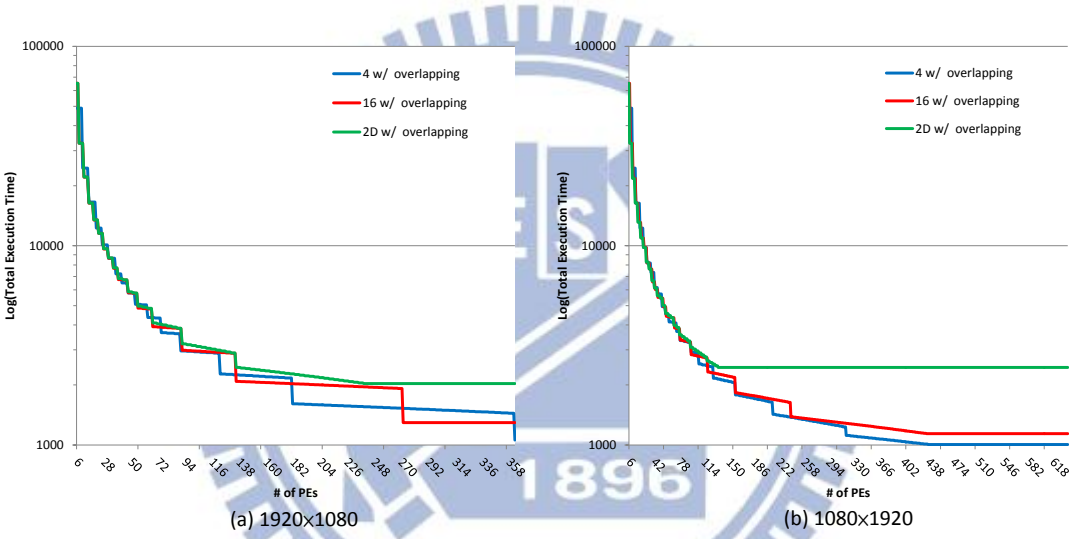


Figure 4-2 Proposed methods compared with the 2D wave-front method in time for deblocking and number of PEs when frame size is (a)1920×1080 (b)1080×1920.

Moreover, we find the total execution time curves have a step-like pattern. This characteristic comes from the splitting of frames. When the number of PEs passes a threshold in which the number of PEs can divide evenly into the total number of MB rows, the total execution time is greatly reduced, thus forming the curves.

While the speedup of the 2D wave-front method stops at 240 PEs for the horizontal shaped frame, and 136 PEs for the vertically one, the speedup of our proposed methods keeps improving until 272 PEs in 16 pixel long boundary method

and 363 PEs in 4 pixel long boundary method for the vertically shaped frame; 436 PEs in 16 pixel long boundary method and 438 PEs in 4 pixel long boundary method for the horizontal case.

Considering limited amount of PEs, we find the 16 pixel long boundary method is better than the 2D wave-front method for any number of PEs. The 4 pixel long boundary method is better than the 16 pixel long boundary even the parallelism is similar, because of short wind up/wind down time in the 4 pixel long boundary method. However in some cases the 16 pixel long boundary method is better than the 4 pixel long boundary, most cases in the 4 pixel long boundary method is better than both the 16 pixel long boundary method and 2D wave-front method.

As mentioned above, in some cases the 16 pixel long boundary method and 2D wave-front method sometimes get better time reduction. We use an example to explain. Figure 4-3 compares the 2D wave-front method and the 4 pixel long boundary method with deblocking overlapping for a 1920*1080 sized frame using 70 PEs. This example explains when the 4 pixel long boundary method will get higher maximum degree of parallelism but fail in getting better time reduction. When a frame is divided into stripes, the last stripe of the frame might be small, but still takes long delay to deal with. This delay could be covered when more than 74 PEs available.

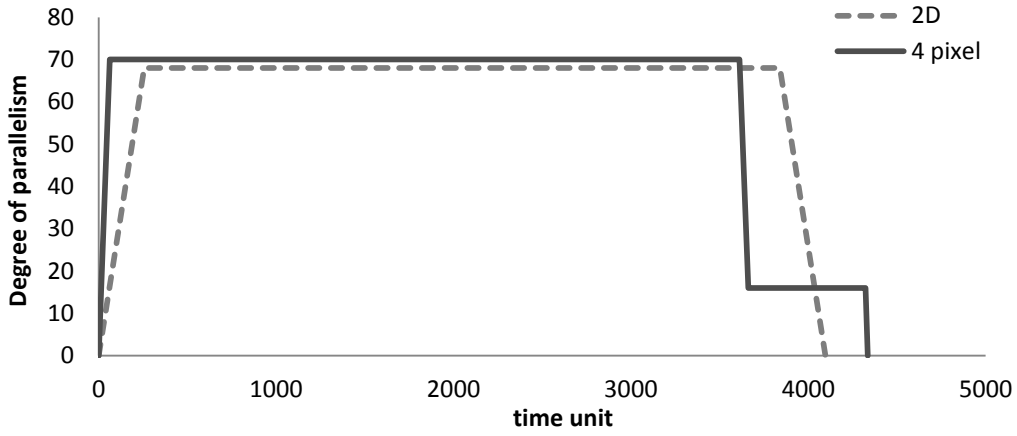


Figure 4-3 4 pixel long boundary method compared with the 2D wave-front method in degree of parallelism and time for deblocking when frame sizes are 1920×1080 and using 70 PEs.

With enough PEs, Figure 4-4 shows the comparison of the proposed designs and the 2D wave-front method for both (a) horizontally shaped and (b) vertically shaped frames.

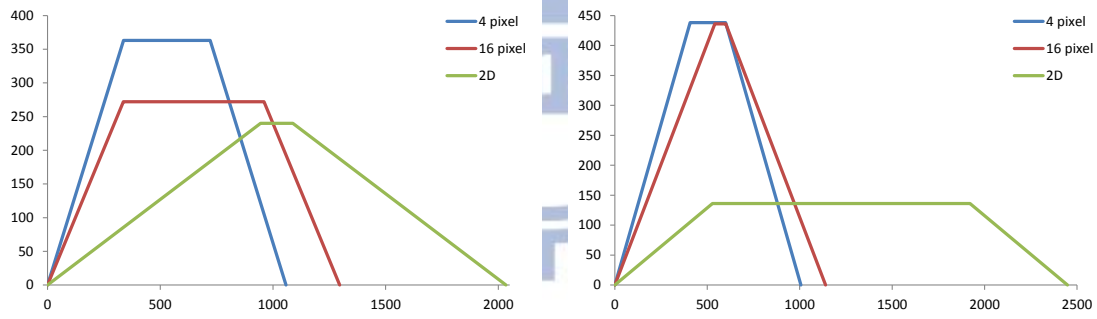


Figure 4-4 Proposed methods compared with the 2D wave-front method in idealize degree of parallelism and time for deblocking when frame sizes are (a) 1920×1080 and (b) 1080×1920.

Table 4-1 shows the speedup for total execution time and Table 4-2 shows the slope of idealize parallelism in wind up time for each design.

Table 4-1 Speedup for total execution time.

speedup	1920×1080	1080×1920
16 : 2D	1.57	2.15
4 : 2D	1.92	2.44
4 : 16	1.22	1.13

Table 4-2 Increasing slope of parallelism in wind up.

Method	Slope of parallelism in wind up time
4 pixel long boundary	1.08
16 pixel long boundary	0.81
2D wave-front	0.25

Last but not least, when considering if our design is complementary with the 3D wave-front method as the 2D wave-front is, the answer is yes. Due to the deblocking filter having no inter-frame data dependencies, our approach is definitely complementary with the 3D wave-front method.

Chapter 5 Hardware Architectural Requirements

In order to deblock a video frame in the proposed order, some hardware support may be necessary. In this Chapter, we list some major hardware requirements such as dedicated buses between PEs, data loop-backs, and internal buffers. Any hardware that fulfills these requirements should be capable of gaining the benefits from proposed order.

5.1 16 pixel long boundary

Based on the PEs assignment mentioned above, followings are the requirements of the hardware design:

- As mentioned in case 1 and 2 of Chapter 3.1, 16 pixel long boundaries a row of MBs are required to be deblocked in sequential order, so we can assign one PE for each row of MBs. While deblocking a row of MBs, some intermediate pixel values should be looped back to the PE itself or be kept in internal buffers for further use later.

- As mentioned in case 3 of Chapter 3, we know that the deblocking of every MB_{b5} requires the pixel values that come from its Upper and Upper-right MB. Since we assign a PE to the deblocking of a row of MBs, we need a dedicated bus for data bypassing between PEs dealing with adjacent rows of MBs. This bus can be unidirectional from the upper PE to its adjacent lower PE.

A schematic of the hardware architectural requirements is shown in Figure 5-1. The Input Buffer stores pixels that are not yet deblocked, the Output Buffer stores

pixels that have been deblocked, and the Internal Buffer stores pixels whose value is needed later. Moreover, PEs will use a shared bus to access memory. If the share bus bandwidth satisfied, the Input Buffer size and Output Buffer size are 1 MB size, and the Internal Buffer size is 1/4 MB size for one PE.

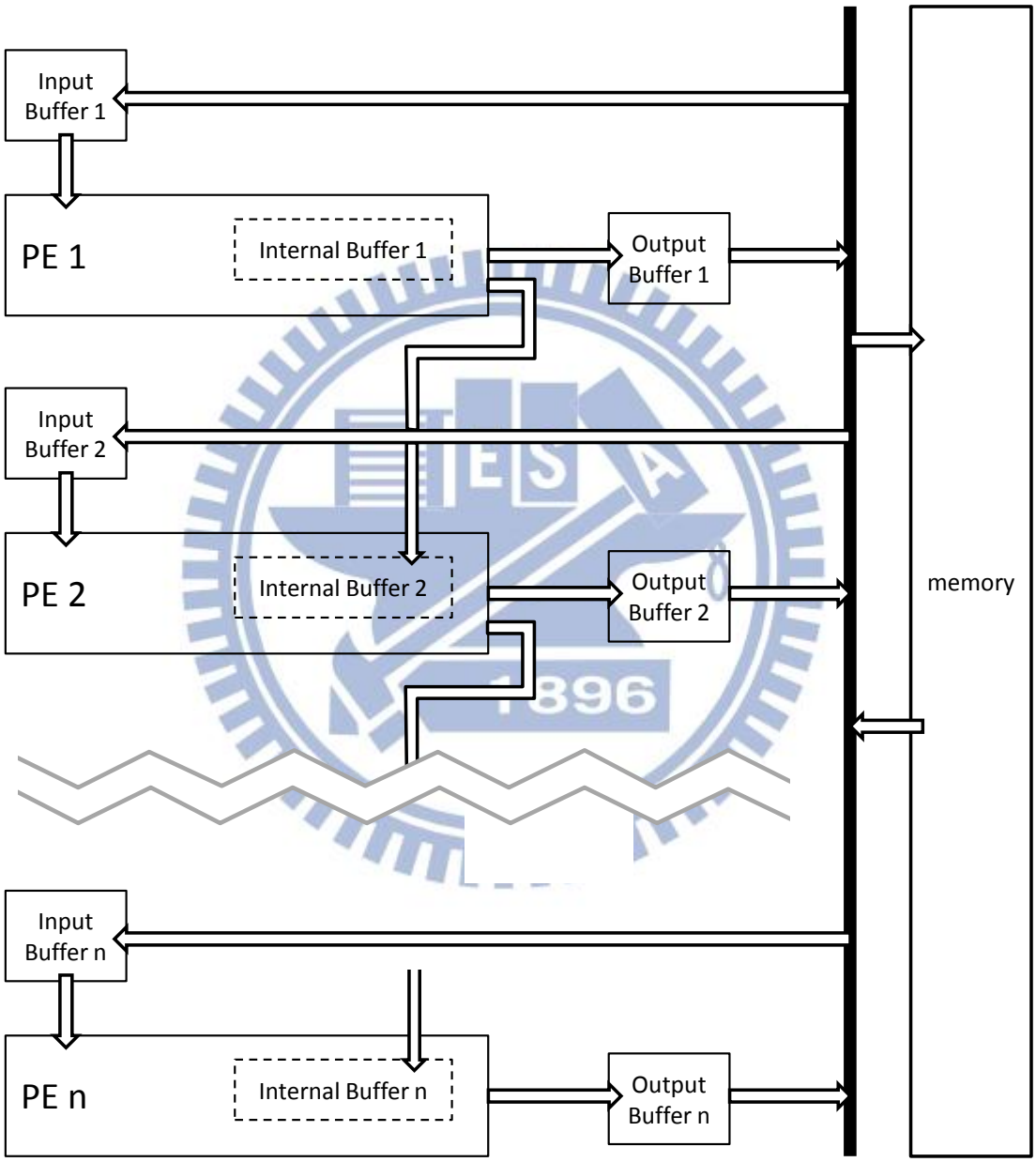
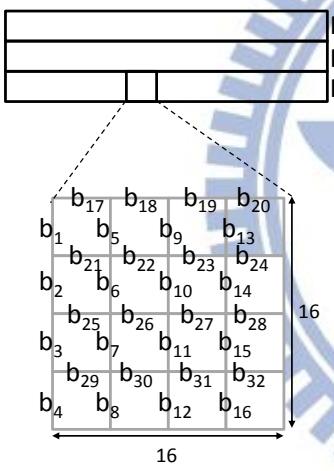


Figure 5-1 Schematic of hardware architectural requirements.

5.2 4 pixel long boundary

As mentioned in Figure 3-30 of Chapter 3.2, it provides totally 16 4 pixel long

boundaries that can be deblocked in parallel every 3 rows of MBs. As a result, we can group 3 rows of MBs and assign 16 PEs for each group. We proposed a PEs assignment as shown in Figure 5-2(b), it shows all 4 pixel long boundaries in group be processed by 16 PEs. For examples, PE1 process each MB's b_1, b_5, b_{17}, b_{18}, b_{20}, b_{24} in MB R1; PE6 process each MB's b_8, b_{29} in MB R1 and each MB's b_1, b_{12}, b_{17}, b_{18} in MB R2. In other words, the same PE processes the same boundaries in each MB of same MB row.



(a)

PE	Assignment (not Exec. Order)
PE1:	R1-b_1 , R1-b_5 , R1-b_{17}, R1-b_{18}, R1-b_{20}, R1-b_{24}
PE2:	R1-b_2 , R1-b_9 , R1-b_{13}, R1-b_{19}, R1-b_{23}, R1-b_{28}
PE3:	R1-b_6 , R1-b_{16}, R1-b_{21}, R1-b_{22}, R1-b_{25}, R1-b_{32}
PE4:	R1-b_3 , R1-b_{10}, R1-b_{14}, R1-b_{15}, R1-b_{27}, R1-b_{31}
PE5:	R1-b_4 , R1-b_7 , R1-b_{11}, R1-b_{12}, R1-b_{26}, R1-b_{30}
PE6:	R1-b_8 , R1-b_{29}, R2-b_1 , R2-b_{12}, R2-b_{17}, R2-b_{18}
PE7:	R2-b_5 , R2-b_9 , R2-b_{13}, R2-b_{20}, R2-b_{24}, R2-b_{28}
PE8:	R2-b_2 , R2-b_6 , R2-b_{16}, R2-b_{21}, R2-b_{22}, R2-b_{25}
PE9:	R2-b_3 , R2-b_{10}, R2-b_{19}, R2-b_{23}, R2-b_{26}, R2-b_{32}
PE10:	R2-b_4 , R2-b_7 , R2-b_{14}, R2-b_{15}, R2-b_{29}, R2-b_{31}
PE11:	R2-b_8 , R2-b_{11}, R2-b_{27}, R2-b_{30}, R3-b_{11}, R3-b_{18}
PE12:	R3-b_2 , R3-b_{17}, R3-b_{20}, R3-b_{21}, R3-b_{24}, R3-b_{28}
PE13:	R3-b_5 , R3-b_9 , R3-b_{13}, R3-b_{22}, R3-b_{23}, R3-b_{27}
PE14:	R3-b_6 , R3-b_{10}, R3-b_{14}, R3-b_{15}, R3-b_{26}, R3-b_{32}
PE15:	R3-b_3 , R3-b_7 , R3-b_{19}, R3-b_{25}, R3-b_{29}, R3-b_{30}
PE16:	R3-b_4 , R3-b_8 , R3-b_{11}, R3-b_{12}, R3-b_{16}, R3-b_{31}

(b)

Figure 5-2 (a) ID assignment in a MB. (b) PEs assignment for 3 MB rows.

Based on the PEs assignment mentioned above, followings are the requirements of the hardware design:

1. While deblocking a 4 pixel long boundary, some intermediate pixel values should be looped back to the PE itself or be kept in internal buffers for further use later.

2. We know one PE deblocking of a 4 pixel long boundary requires the pixel values that maybe come from five sources:

i. Its two results previous stage.

ii. Bypassing from others two PEs results previous stage.

iii. First deblocking pixel values that come from the memory.

3. While one PE deblocking of a 4 pixel long boundary, its outputs maybe transfer to other objectives:

i. Transfer to the output buffer.

ii. Transfer to other PEs.

A schematic of the hardware architectural requirements for one PE is shown in Figure 5-3. The Buffer stores pixels whose value is needed later, the Output Buffer stores pixels that have been deblocked. The Buffer size is 2 4×4 blocks, and the Output Buffer size is 2 4×4 blocks for one PE.

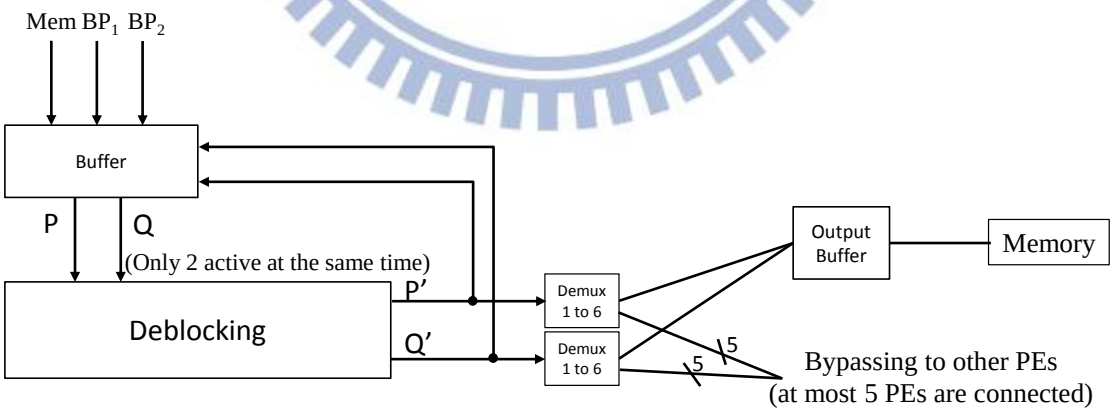


Figure 5-3 Schematic of hardware architectural requirements for one PE.

According to the PEs assignment in Figure 5-2(b), the connection of PEs as shown in Figure 5-4.

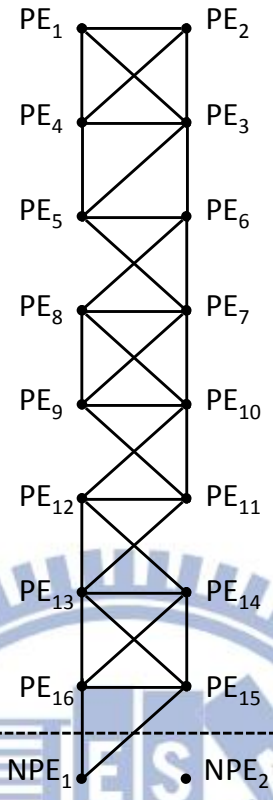


Figure 5-4 The connection of PEs.

5.3 Timing model

In order to find out the overhead of different granularity, we separate the time that process one deblocking operation to time for read, time for filer, and time for write.

According to the hardware as mention above, the sources of read and write are memory, others PEs, and self PE. Assuming an address can load 16 pixels, the time latency of 3 sources is memory: others PEs: self PE = x: y: z. Due to we don't find the time latency of 3 sources ratio, we assume y = x/1000 and the time latency of self PE is 0. We use CACTI[9] to estimate the time latency of memory is 1.63ns, and model the time latency of read data and write data in Table 5-1.

Table 5-1 Time for read data and write data.

Memory	Others PEs	Self PE
1.63(ns)	0.013(ns)	0(ns)

According to Ref[8], we can know time for filer is 10ns. We assume one stage compose of read data, filter, and write data. The time of one stage is 13.26ns that is sum up maximum time for read, maximum time for filter, and maximum time for write. The timing model can find the time for process one frame size is 1920×1080 as shown in Table 5-2. The P_t is the maximum parallelism at time t in Table 5-2.

Table 5-2 Time for deblocking a 1920×1080 frame.

Granularity	Time
2D-wavefront(MB)	$\sum_{t=0}^{2032}(P_t \times 13.26)$
Boundary_{16}	$\sum_{t=0}^{1295}(P_t \times 13.26)$
Boundary_{4}	$\sum_{t=0}^{1057}(P_t \times 13.26)$

After timing model, the speedup of different granularity is the same as Table 4-1. But the speedup of original sequential deblocking is different. We use Boundary_4 to process one QCIF(176×144) need 108 stages. In Ref[10] can know sequential deblocking one MB need 530 cycles, so average cycles for process one 4 pixel long boundary is 17. In Ref[10] sequential process one QCIF frame need 51930 cycles, the

Table 5-3 can find the ideal speedup and actual speedup is different.

Table 5-3 Speedup for idealize and actually.

	idealize	actually
speedup	$\frac{99 \times 32}{108} \cong 29.33$	$\frac{51930}{108 \times 17} \cong 28.28$

The difference is come from the Boundary strength (BS) value, BS value range from 0 to 4. Each boundary have a BS value, boundary unneeded deblocking if BS value is 0, boundary needed deblocking if BS value is 1~4. In sequential processing, process next boundary if current boundary's BS value equal to 0. But in our design, the time of stage is fixed, it must wait if current boundary's BS value equal to 0.

Chapter 6 Conclusion

As shown in our proposed order, examining the deblocking algorithm at a finer granularity did bring additional opportunities for exploiting parallelism, and thus speed up the execution time of the deblocking filter. 4 pixel long boundary method compared with the 2D wave-front method order in deblocking both 1920*1080 and 1080*1920 pixel sized frames, we gain a speedup of 1.92 and 2.44 times given an un-limited number of PEs respectively. For an environment with limited hardware resources, we also provide an algorithm able to fully utilize available resources for the deblocking filter.

Considering the trend of digital video codecs, larger frame sizes and reduced coded video size are both essential. In order to achieve this goal, the deblocking filter plays an important role because dealing with larger frames takes time proportional to the frame size. The proposed design can limit the growth in time spent deblocking by the maximum of the frame width and height, which are often proportional to the square root of the frame size. Thus it brings the opportunity for practical real-time deblocking of larger sized videos in the future.

The proposed approach in this paper is just the first step of parallelizing H.264 video decoding in a finer way. In order to exploit overall parallelism, decoding stages including intra decoding and motion compensation are all required to consider the parallel order of their operations. However, we are able to further analyze the algorithms of these stages to see if there are any opportunities for using a similar approach to that in this paper.

References

[1] List. P. Joch, A., Lainema., J., Bjontegaard. G., Karczewicz. M., "Adaptive deblocking filter," Circuits and Systems for Video Technology, IEEE Transactions on , vol.13, no.7, pp.614-619, July 2003

[2] E. Van der Tol, E. Jasper, R.H. Gelderblom, “Mapping of H.264 Decoding on a Multiprocessor Architecture” Proceeding of SPIE Conference on Image and Video Communications 2003, p.p.707-709

[3] Meenderinck, C., Azevedo, A., Alvarez, M., Juurlink, B., Ramirez, A.: Parallel Scalability of H.264. In: Proc. First Workshop on Programmability Issues for Multi-Core Computers (January 2008)

[4] Zhuo Zhao, Ping Liang, "Data partition for wavefront parallelization of H.264 video encoder," Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on , vol., no., pp.4 pp.-2672, 0-0 0

[5] Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC), Mar. 2003.

[6] Ke Xu, Chiu-Sing Choy, "A Five-Stage Pipeline, 204 Cycles/MB, Single-Port SRAM-Based Deblocking Filter for H.264/AVC," Circuits and Systems for Video Technology, IEEE Transactions on , vol.18, no.3, pp.363-374, March 2008

[7] Yun-Shuo Chang, "Improvements of H.264 De-blocking filter and DST Implementation of H.264 Decoder," A Thesis Submitted to Institute of Electrical Engineering National Yunlin University of Science & Technology in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering, July 2007.

[8] T.M. Liu, W. P. Lee, T.A. Lin, and C. Y. Lee, “A memory-efficient deblocking filter for H.264/AVC video coding,” in Proc. IEEE Int. Symp. Circuits Syst., May 2005, vol. 3, pp. 2140-2143.

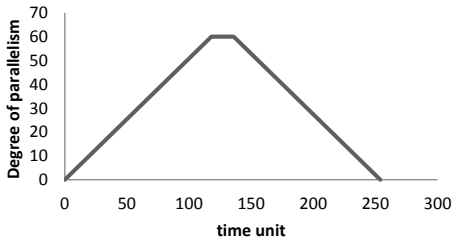
[9] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, “CACTI 5.3”, Technical Report. HPL-2008-20. 2008.

[10] Eric Gerard Ernst, “Architecture Design of a Scalable Adaptive Deblocking Filter for H.264/AVC,” A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering, July 2007.

Appendix

Q&A

1. 下圖橫軸為何是時間?

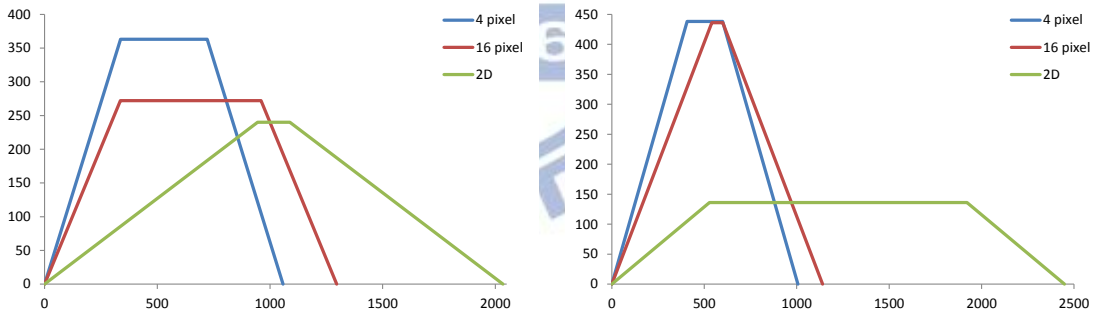


A1:

這個時間單位的時間長度為一個 stage 的時間長度，橫軸代表的是計算所需實際的 time unit 數，而不是直接執行所得到的時間。我們每一個 time unit 的長度為處理一個 MB 所需最長的時間。這裡使用時間是想表達隨著時間平行度的變化。

2. 為什麼粒度越小平行度越高，現象從何而來?粒度不一樣的差異在哪?

A2:



(a) 1920×1080　　(b) 1080×1920

根據上圖，粒度越小平行度越高，而改變粒度其實是改變判斷 dependency 的最小單位，原本被綁在一起的 boundaries 被打散了之後，部分實際上無 dependency 的 boundaries 就可能提早執行。所以粒度越小有機會讓部分 boundaries 提早的執行，也就可以讓一開始的平行度爬升較快。

3. HW requirement 為何 PE 間要有 connection，因為已經有 share bus 了？

A3:

同時需要傳的資料量很多，且 PEs 是緊鄰的，所以 PEs 間的 connection cost 是低的，故 PEs 之間用 connection 傳遞是適合的。

4. 因為 PE 有 buffer 所以應該也要加 buffer size 要假設上去?

A4:

16: 若 share bus 可滿足所需傳遞的資料量，則 input buffer 的 size 為 1 MB，output buffer 的 size 為 1 MB，internal buffer 的 size 為 1/4 MB。

4: input buffer 的 size 為 2 4x4 blocks，output buffer 的 size 為 2 4x4 blocks，internal buffer 的 size 為 4x4 block。

5. 如何證明我的 order 是最好的?

A5:

以本論文主要目標為提高計算平行度以降低執行時間而言，在 PE 數量可滿足最大平行度時，由於所提出的 order 已滿足 critical path 最短需求，而針對其他不在 critical path 上的 boundaries 的 order，若無硬體上的限制，則無所謂最好的 order。

而當 PE 數量無法滿足最大平行度時，依本論文提出的方法會將 frame 切割成多個 stripes 依序處理。在此種情形下，我們希望在所擁有的 PE 數量下能盡量處理越大的 stripe，而 stripe 是由多個 MB row 所組成的。為了充分利用到所擁有的 PEs，我們所提出的 order 就是根據每多處理一個 MB row 所需要的 PE 數量是理論上最小的。固我們的 order 在此兩個條件下是最好的 order。

6. 為什麼只考慮單張 frame 不去考慮 frame 之間的平行度?

A6:

Deblocking 在 frame 與 frame 之間完全沒有 data dependency，且[3]已完成 frame 間平行處理的設計，所以我們只需專注在單張 frame 內的平行就可以與之搭配。

符合[3]所需要的條件：

1: 對於 frame 內的平行方式為 wave-front。

2: frame 的 wind down 與下一張 frame 的 wind up 沒有 data dependency。

我們所提出的方法滿足上述條件，固可以與之搭配。

7. 時間單位是以每一個 boundary 所需處理時間都是一樣的，但在事實上不會是這樣的處理情形，這樣會不會產生新的問題?

A7:

Filter 一個 boundary 的時間會根據 boundary strength(BS)值而有所不同，若 BS 值為 0 則此 boundary 不需 filter，若 BS 值為 1~4 則需要 filter。

根據我們所提出來的 order，在同一個 stage 可以一起處理的 boundaries 的 BS 值都是 0 的機率很低的，所以在這邊我們時間單位是取處理一個 MB 所需最長的時間，以 fully synchronize 方式去處理。

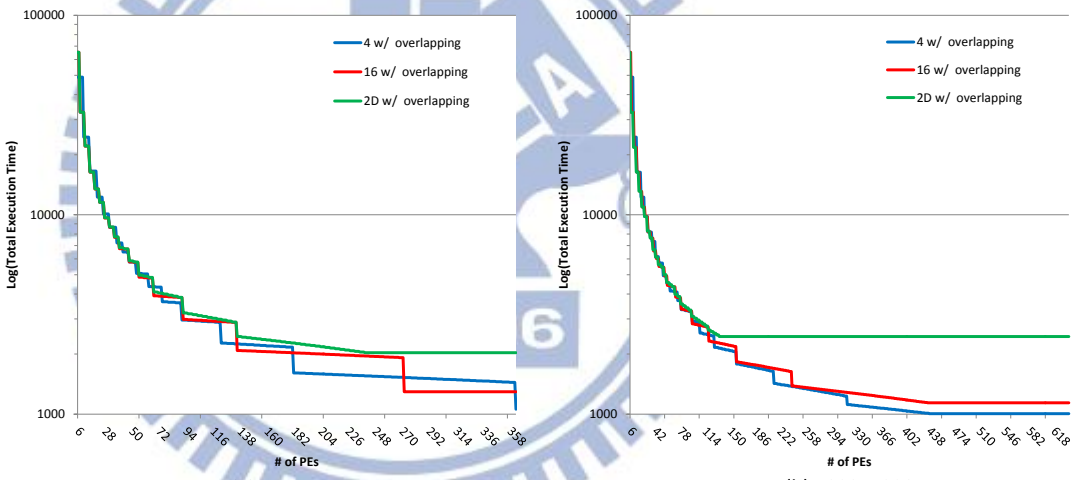
8. 為什麼 4 pixel long boundary order 可以直接對應到 1 pixel long boundary order，而這個現象不會直接出現在 16 跟 4 之間?

A8:

因為 16 pixel long boundaries 之間有互相交錯，所以以更細粒度去分析 data dependency 有機會讓部分可以先處理的 boundaries 提早處理，而 4 pixel long boundaries 之間沒有互相交錯，所以在以更細的粒度去分析 data dependency 不會有差別，以致於可以直接對應到 1 pixel long boundary。

9. 找出一個規則來決定適合的粒度?

A9:



本論文的目的是希望盡量降低執行時間，故在選擇適合的粒度可根據上圖，依目前 PE 數量，選擇執行時間最短的粒度。

10. 越細的粒度會衍生一些什麼 overhead?應該會在粒度以及 overhead 有 trade off?應該如何找到一個粒度所能達到的平行度是高的且可以 fully utilize 在 multi-core 上以及 overhead 是小的

A10:

根據上述問題，我們嘗試將 deblocking 的時間提出適合的數學 model，並去觀察不同粒度所會造成的 overhead。而 model 後的結果，在不同粒度間 speedup 的比較等同於 Chapter 4 的結果。

然而，根據我們的 model 仍然可以發現我們平行化方法與傳統循序

執行方法相比，在 idealize speedup 與 actual speedup 上會有所不同。(其詳細內容已加入到 5.3)