A MEMORY-EFFICIENT DEBLOCKING FILTER FOR H.264/AVC VIDEO CODING

Tsu-Ming Liu, Wen-Ping Lee, Ting-An Lin and Chen-Yi Lee

Institute of Electronics Engineering, National Chiao Tung University
1001, Ta Hsueh Road, Hsinchu, 300, Taiwan, R.O.C.
mingle@si2lab.org, cylee@si2lab.org

ABSTRACT

A memory-efficient architecture design for de-blocking filter in H.264/AVC is presented. We use the novel data arrangement of Column-of-Pixel to facilitate the memory access and reuse the pixel value. Further, we propose a hybrid filter scheduling to improve the system throughput. As compared with some existing approaches of realizing de-blocking filter [1] [2], the proposed design saves about one-half of processing cycles. With novel data arrangement and hybrid filter scheduling, an efficient architecture design is implemented. Further, it is evaluated on H.264 system and easily achieved real-time decoding with 1080 HD (1920x1088@30fps) when working frequency is 100MHz.

1. INTRODUCTION

H.264/AVC is the newest video coding standard of Joint Video Team (JVT) [3]. H.264/AVC has achieved significant rate-distortion efficiency by many useful tools. De-blocking filter (a.k.a. loop filter) placed in the prediction loop is one important tool to increase the coding efficiency and remove the blocking artifacts. Generally, the de-blocking filter contributes about one-third of the computational complexity of the decoder [4], and it's the system bottleneck in terms of processing cycles. Compared to the loop filters in H.263 or MPEG-4/H.263 post filters [5], the de-blocking filter in H.264 operates each filter process on 4x4 sub-block structure instead of 8x8 block structure. Therefore, large amount of computation and memory access are its penalty for the real-time decoding demand.

Through the analysis of system performance, we make a decision of our memory organization between prediction unit and de-blocking filter. Instead of LOP in [2], we exploit the block ordering which is standard-defined to decide the Column-of-Pixel (CoP) memory organization. By using this method, we can reuse the neighboring data in intra or inter prediction unit and reduce the number of memory access.

We modify the processing order of filtered block boundaries without affecting the data dependency. In the de-blocking filter of H.264/AVC, vertical edges are filtered first, and then horizontal edges are filtered. Unfiltered data should be fetched in each filtered direction. Therefore, double of memory accesses are required for one 4x4 sub-block. To reduce the memory access and the processing cycles, we propose a hybrid filtering method to re-schedule the filter ordering and reuse the pixel value on the different directions.

*Work supported by the National Science Council of Taiwan, R.O.C., under Grant NSC 93-2220-E-009 -010-

Figure 1. The different data arrangement in the de-blocking filter (a) and prediction unit (b).

The rest of this paper is organized as follows. Section 2 describes our memory organization between the prediction unit and the de-blocking filter. Section 3 shows our hardware architecture with hybrid scheduling method for the high-throughput design. Section 4 presents our simulation results. Finally, section 5 summarizes our work and draws the conclusions.

2. MEMORY ORGANIZATION BETWEEN PREDICTION AND LOOP-FILTER

Different memory organization leads to the different memory access and processing latency. The input data of de-blocking filter is just the output data of prediction unit, and plus the residual data. To improve the overall processing throughput, we make the hardware cycle profiling to decide the memory organization among them. Further, two dedicated single port SRAMs are designed to not only store the current and neighboring data but also achieve the efficient data access in each 4x4 edge.

2.1 Memory Organization

We use one Column-of-Pixel (CoP) as the data word size in each memory address. In Figure 1(a), there are two policies for data arrangement. The Row of Pixel (RoP) is labeled with the case of L1 and 2 blocks, and the Column of Pixel (CoP) is in the case of U1 and 1 blocks. Each row or column of pixel contains four pixels with a total of 32-bit wide. For the de-blocking filter, RoP (i.e. LOP in [2]) is a straightforward method to arrange the pixel value in the vertical edge filtering. However, it will induce extra memory access when applying to the horizontal edge filtering.

By the same way, this situation is also occurred in the CoP arrangement. Different arrangements of CoP or RoP also affect the number of memory access in the intra or inter prediction unit. In Figure 1(b), the standard-defined 4x4 sub-block ordering is labeled in each block. There are strong dependencies in the horizontal block order. Therefore, we choose CoP data arrangement to reuse the pixel value in the block-boundary with white-circle region. Further, we list the hardware profiling in terms of memory access in Table 1. The evaluated cycles with CoP or RoP data arrangement are almost the same in the de-blocking filter unit. The reason is that the filtering process will be performed on not only horizontal edge but also vertical edge. However, there are improvements in the prediction unit when applying the CoP arrangement. Therefore, compared to the RoP data arrangement, we choose the CoP data arrangement in each MB to reduce the number of memory access.

Table 1. The analysis of average memory access per luma MB.

Memory Arrangement	# of memory access		
	Intra Prediction	Inter Prediction	De-blocking Filter
CoP	40	313	156
RoP	48	432	156
Improvement (RoP-CoP)/RoP	17%	28%	0%

2.2 Slice and Content Memory

To facilitate the data access with each block pixel or neighboring pixel, we use two single-port SRAMs named as slice memory and content memory to keep the neighboring pixel and block-content pixel value. The fetching and restoring pixel value is very frequently since de-blocking filter in H.264/AVC is performed on each 4x4 sub-block level. To reduce the pin counts and speed up the filtering process, the internal SRAM module is essential to meet the real-time decoding demand.

The slice memory is used to store the neighboring pixel. It is required to keep them until they have been filtered completely. Further, the address depth is decided by the frame width. In Figure 2(a), considering the frame size with $M \times N$, each square represents the 16x16 MB. Each MB contains the 16 points, and 4x4 pixels within each point. When the filtering process is performed from the MB index of B to $B+1$, the pixel data within upper and left neighbor will be updated as the arrows show. The shaded region should be kept when the filtering index is $B+1$. Therefore, the slice memory is used to keep the pixel value of upper and left neighbor and contains the size of about $2N \times 32$ for the 4:2:0 format.

The content memory is used to store the unfiltered pixel value in luma or chroma block. The data word-length of memory is based on the 32-bit of CoP, and the address depth of content memory is decided by the YUV format (4:4:4, 4:2:2 or 4:2:0). For 4:2:0 format, there are 16 blocks of luma and 8 blocks of chroma should be stored. Therefore, the size of content memory is $(16+8) \times 4 \times 32$ in total. Further, the data address is increased as the standard-defined block ordering of Figure 2(b). The grid region is stored in the slice memory and the dotted region is stored in the content memory.

Figure 2. The slice memory with grid or shaded region and the content memory with black-dotted region.

Figure 3. The proposed hybrid scheduling method.

3. HIGH-THROUGHPUT LOOP FILTER

3.1 Proposed Hybrid Scheduling

To reduce the overhead with the reloaded data when switching the filtering edge from horizontal to vertical, we propose a hybrid filter scheduling to re-schedule the standard-defined edge. The de-blocking filter in H.264/AVC is performed in the vertical edge first, and then the horizontal edge. Based on the standard-defined filter ordering, we can deduce the filter order on each 4x4 sub-block as Figure 3(a). In the filter ordering of 4x4 sub-block, left edge is filtered first and lower edge is the last one. We propose a novel filter ordering to schedule our filter process on each edge as Figure 3(b). Each filter order of one block obeys the rules of the left edge first and the lower edge last. Compared to the traditional scheduling [1][2], our proposed method prevents the re-access for different direction and combine the vertical and horizontal filter at the rule of standard-compliance.

We use four 4x4 pixel buffer to keep the temporary data in our hybrid scheduling process. In Figure 4(a), each MB has been partitioned into two main parts (i.e. *Loop Filter-MB-Upper* or *Lower*) to reduce the kept buffer size. Each part is composed of eight time-instances to process the filtering procedure in Figure 4(b). The grid region represents the neighboring block and the shaded region is the position of kept data buffer with the size of four 4x4 sub-blocks. There is no need to keep the neighboring block as the data buffer in each time instance (except for the initial state $t1$ since we use the CoP data arrangement) because the neighboring block and current MB are located at different memory module. Both data of them can be accessed at the same time instance and sent to the input of edge filter.

Figure 4. The partitioned MB and each time instance when applying the hybrid scheduling method.

Figure 5. The block diagram and data flow of the proposed design.

We derived the filter ordering of the proposed hybrid scheduling method in Figure 4(b). Each bold line represents the edge to be filtered in each time instance. The filtered ordering complied with the hybrid scheduling in Figure 3(a) at each time instance (t1~t8). By the same way, the proposed scheduling is also performed in the 4x4 sub-block of chroma representation.

The main problem of the de-blocking filter in H.264/AVC is the considerable amount of memory access and processing cycles. To apply the proposed hybrid scheduling into the overall system and enhance the system throughput, we propose a high-throughput architecture design of de-blocking filter.

3.2 Proposed Architecture of De-blocking Filter

Figure 5 shows the proposed design with block diagram and data flow representation. The size and organization of content and slice memory have been presented in section 2. We choose CoP memory arrangement to improve the pixel data utilization and reduce the memory access in the prediction unit. The external frame buffer is an off-chip memory, and the size is decided by the frame size and the frame number for the long-term prediction. The shaded-arrows denote the data flow inside the de-blocking filter unit, and the black-arrows denote the data flow outside. The pixel buffer is used to store the intermediate pixel value when applying the proposed hybrid scheduling. It contains the four 4x4 pixel values. Moreover, in each time instance, it locates at the position as the shaded regions of Figure 4(b) shows. The edge filter is a simple parallel in and parallel out process. It exploits the 3, 4 or 5-tap filter to attenuate the blocking artifacts due to the motion compensation or prediction error coding in each block boundary. More detailed algorithms are described in [4].

Figure 6. The detailed architecture for the de-blocking filter.

The detailed architecture for the de-blocking filter unit of Figure 5 has been shown in Figure 6. All the data signals are 32-bit wide and contain the LoP of memory organization discussed in section 2. There are four input signals $\{wt\_B\_0, wt\_B\_1, wt\_B\_2, wt\_B\_3\}$ to write the buffers with 4 blocks. Further, there are three output signals $\{rd\_B\_0, rd\_B\_1, rd\_B\_2\}$ to read three of them to perform the edge filter, and then write to the frame buffer, pixel buffer or slice memory. In addition, the write result of the 4 blocks is shown in Figure 4(b) to achieve the hybrid filtering and avoids the extra access from the filtering of different direction. By the same naming rule, each data flow represents the writing/reading to/from the storage module including slice memory, content memory or frame buffer.

After the behavioral illustration of pixel buffer, we use one MB with 48 edges in Figure 3(b) as an example to illustrate the other behavior of Figure 6. The behavior of Figure 6 can be partitioned into two main parts.

- **Write Process** is a writing mechanism through the signal $\{wt\_S\_0 \sim 2, wt\_F\_0 \sim 1, wt\_B\_0 \sim 3\}$.
- **Read Process** is a reading mechanism through the signal $\{rd\_S\_0 \sim 1, rd\_C\_0, rd\_B\_0 \sim 2\}$.

For writing to slice memory, wt_S_0 is used to write the filtered data into the slice memory, and it will be activated only on the edge 6, 10, 14 and 16 (see Figure 3(b)). For the edge 6, the lower block will become the next neighboring block of LF-MB_L in Figure 4(b). The same condition is also applied on the edge 10, 14 and 16. Further, the wt_S_1 will be activated on the edge 31, 32, 40 and 48. The wt_S_2 is performed to write the dotted block data of Figure 3(b) into the slice memory. For the writing signal of frame buffer, wt_F_0 is used to write filtered data into the external frame buffer. It will be activated on each filtering of horizontal edge except for the edge of activated signal wt_S_1 and wt_B_0 , since wt_F_0 , wt_S_1 and wt_B_0 have the same root-signal of P_Pixel . For the edge 6 as an example, the upper block of edge 6 is the P_Pixel of edge filter's output. This block will write to the external frame buffer since it has been filtered completely for all the edges of $\{1, 3, 5, 6\}$. The wt_F_1 is performed in the same way except that the input signal comes from the output of pixel buffer.

For the reading process of slice memory, rd_S_0 is only activated on the edge of $\{1, 2, 17, 18, 31, 33, 34, 39, 41, 42, 47\}$. For the edge 1, the rd_S_0 is the input of pixel buffer. We need to keep the pixel value since we apply the CoP arrangement of each data. That's why we keep the left neighboring as the pixel buffer in the $t1$ of

Figure 4(b). However, for the vertical filtering of edge {5,9,13,15,21,25,29,37,45}, it can directly feed through the edge filter by rd_S_1 . Finally, compared to the existing approach, the content memory of our proposed design is only used for read. There is no need to store the filtered result into the content memory in one direction, and read them in another direction. By our proposed hybrid scheduling, we combine the horizontal and vertical filtering process in one filtering flow. Therefore, we need 4 blocks at most to perform the hybrid filtering.

Table 2. The cycle analysis in the de-blocking filter unit.

Cycle Counts		[1]'s basic	[2]	Proposed
Vertical / Horizontal		Seperated	Seperated	Hybrid
Luma	Horizontal	128	104	159
	Vertical	200	110	
Chroma	Horizontal	64	N/A	90
	Vertical	112	N/A	
Total		504	214 + N/A	250

4. SIMULATION RESULTS

In Table 2, based on the proposed architecture for high throughput de-blocking filter, the evaluated cycle counts are 159 cycles for Luma block and 90 cycles for chroma block. Specifically, we need 8 cycles ($LF-MB-U + LF-MB-L$) in the initial states. Further, there are 4×32 cycles to filter each horizontal and vertical edge in one luma MB. Finally, we need 20 cycles to write the filter result for the edge {16,22,26,30,32} and incur 3 cycles due to the data hazard in our filtering process. In sum, we need 159 (i.e. $8+4 \times 32+20+3$) cycles to filter horizontal and vertical edge of luma MB. By the same analysis, we need 90 (i.e. $4+4 \times 8+8+1=45$ for each chroma) cycles to filter the horizontal and vertical edges in chroma block. Therefore, there are total 250 cycles with extra 1 cycle for data hazard.

The simulation results are summarized in Table 3. The target technology is $0.18 \mu m$, and the synthesized gate count is 19.64K excluding the slice and content memory. Two single port SRAM is organized to store the content of YUV data and the neighboring data. They contain the size of 96×32 and $2N \times 32$ where N means the width of the coded frame.

We use “foreman” as our test sequence. The evaluated cycle counts per MB are 250 cycles. Further, compared with the existing approach [1][2], our proposed architecture can save about one-half of processing cycles per MB. We use the hybrid-scheduling scheme to combine the horizontal and vertical filtering process, and slightly increase gate counts to keep the intermediate pixel value. With a pipeline methodology, Figure 7 shows the average processing cycles per one 16×16 MB with the decoding of first frame. Originally, the de-blocking filter is a system bottleneck in terms of processing cycles. Based on the proposed architecture, we can greatly reduce the processing cycles and improve the system throughput (i.e. $350 \text{ cycle/MB} = 9523 \text{ MB/frame}$ with $30 \text{ fps}@100 \text{ MHz}$). Therefore, this processing capability can real-time decode 1080HD (1920×1088 , i.e. 8160 MB/frame) or higher with 4:2:0 format when the working frequency is 100MHz.

Table 3. The comparison of architecture and processing cycles for the de-blocking filter in H.264/AVC.

Items	[1]	[2]	Proposed
Design Methodology	Shift-register based design	Line-buffer based design	Line-buffer based design
Kept Data Size	2 blocks	4 blocks	4 blocks
Gate Count	18.91K (0.25um)	N/A	19.64K (0.18um)
Working frequency	100 MHz	N/A	100 MHz
Processing cycles per MB	504 cycles/MB	214 cycles/luma-MB + N cycles/chroma-MB	250 cycles/MB = 159 cycles/luma-MB + 90 cycles/chroma-MB + 1 data hazard
Memory Requirement	2 single-port SRAM (basic architecture)	N/A	2 single-port SRAM

Figure 7. The average processing cycles in pipeline stage i.

5. CONCLUSION

In this paper, we present a Column-of-Pixel (CoP) memory arrangement to reuse the pixel between the de-blocking filter and the prediction unit. Further, we propose a hybrid scheduling to reduce the processing cycles and improve the system throughput. The main idea is that we use four pixel buffers to keep the intermediate pixel value and perform the horizontal and vertical filtering process in one hybrid scheduling flow. Moreover, the proposed design is implemented on the hardware architecture. Based on the working frequency of 100MHz, the synthesized gate counts are very small (19.64K, only the 10% of our intra decoder design). Finally, the proposed architecture can easily achieve the real-time decoding with 1080HD@30fps in 4:2:0 format.

6. REFERENCES

- [1] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang and Liang-Gee Chen, “Architecture Design for Deblocking Filter in H.264/JVT/AVC” *International Conference on Multimedia and Expo(ICME'03)*, Vol. 1, pp. 1-693-6, July 2003.
- [2] Miao Sima, Yuanhua Zhou and Wei Zhang, “an Efficient Architecture for Adaptive Deblocking Filter of H.264/AVC Video Coding” *IEEE Transactions on Consumer Electronics*, Vol. 50, Issue 1, pp. 292-296, Feb. 2004.
- [3] Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC), Mar. 2003.
- [4] P. List, A. Joch, J. Lainema, G. Bjøntegaard and M. Karczewicz, “Adaptive Deblocking Filter”, *IEEE Trans. Circuit Sysr. Video Technol.*, Vol. 13, pp. 614-619, July 2003.
- [5] ITU-T Recommendation H.263: “Video Coding for Low Bitrate Communication”, March 1996.