PARALLEL DEBLOCKING FILTER FOR H.264 AVC/SVC

S Vijay, C Chakrabarti, L J Karam

School of Electrical, Computer and Energy Engineering, Arizona State University
{vijay.sundaram, chaitali, karam}@asu.edu

ABSTRACT

This paper presents a parallel and scalable solution for adaptive deblocking filtering in H.264/AVC. While traditionally in deblocking filtering, the edges in a macroblock are processed in a sequential order, this paper demonstrates how algorithm modifications can be used to enable processing multiple consecutive edges at the same time. The proposed method increases the throughput in proportion to the number of edges that are being processed simultaneously without affecting the PSNR and bit-rate. Details of the method to process 2 consecutive edges in parallel as well as extensions to process 4 and 8 consecutive edges, are provided. A dedicated hardware architecture to process 2 edges is presented along with synthesis results. The architecture achieves a 2x increase in throughput at the expense of a 2.2x increase in area and a 1.23x increase in power.

Index Terms— deblocking filter, parallel architecture

1. INTRODUCTION

The H.264/AVC is the newest video coding standard of Joint Video Team (JVT) [1] that is widely used in video communication servers in network and wireless environment [2] - [3]. It achieves significant rate distortion efficiency by use of several video encoding and decoding tools. Deblocking filter, placed in prediction loop, is one important tool to increase coding efficiency and significantly improve the decoded video quality of the H.264 standard.

Deblocking filter requires significant CPU time and has been reported in [4] to be the most computationally intensive part of the H.264 decoder, taking almost one-third of the computational resources. In the adaptive deblocking filter algorithm in [5], the filter is applied to all the edges of 4-by-4 pixel blocks in each macroblock (MB) (16x16 block). In each pixel block, vertical edges are filtered from left to right, and then horizontal edges are filtered from top to bottom. This ordering introduces a dependency which has to be addressed for fast processing in high throughput applications.

Several architectures have been proposed for the deblocking filter [6]-[13]. All of them retain the quality of the original filter. While methods [6] and [7] focussed on data reuse, [8], [9] and [10] were directed at efficient on-chip memory management. For instance, a vertical processing order was proposed in [9] which reduced the number of memory references and an architecture with an effective data exchange scheme with memory was proposed in [10]. To reduce the number of memory accesses, vector registers were proposed to load and store 4x4 blocks in [11]. In addition, most of these existing methods employed efficient scheduling and hardware pipelining techniques to increase the throughput of the filter. All these architectures process one edge at a time with the exception of [11] which processes a horizontal and vertical edge at the same time. Such a processing rate is not sufficient for applications requiring full High-Definition (HD) resolutions and 3D video. In fact, applications have to manipulate data streams for concurrent processing and parallelize sequential operations as much as possible to cater to the faster processing requirements.

In this paper, we propose a method to increase the throughput of the adaptive deblocking filter by processing multiple consecutive edges at the same time. We consider two adjacent block edges at a time and implement the filtering of both edges simultaneously by directly computing the values of the pixels involved in the filtering of both the edges (overlapped pixels or conflict pixels). The design is scalable and can be used to process four and eight edges provided that hardware is not a constraint. We developed dedicated hardware for this method and synthesized it in 45-nm CMOS technology. Comparison with the original deblocking filter implementation showed that this architecture achieves double the throughput with no degradation in PSNR or bit-rate.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the deblocking filter. Section 3 analyzes the parallelism and describes the algorithm proposed to process two edges simultaneously. Section 4 discusses the architecture, provides results and comments on the scalability of the design. Section 5 gives an overview of related work followed by the conclusion in Section 6.

2. BACKGROUND

In H.264, the deblocking filter is a module which occurs in both the encoding and the decoding paths. It is adaptive at several levels - slice, block-edge and the sample level. The filter varies from edge to edge and is dependent on the boundary

Table 1. Boundary Strength values for various filtering modes

Conditions for Bs value	Bs
Block A or Block B is Intra and the edge between them is a macroblock edge	4
Block A or Block B is Intra predicted	3
Block A or Block B had coded residuals	2
Motion Compensation is from different frames or the difference in motion is greater than one luma sample	1
Else	0

strength (Bs), a parameter which is assigned a value between 0 and 4 to indicate the strength of filtering. Table 1 shows how Bs depends on the modes and coding conditions of the two adjacent blocks. Conditions are evaluated from top to bottom, until one of the conditions holds true, and the corresponding value is assigned to Bs. In the actual filtering algorithm, Bs determines the strength of the filtering performed on the edge. A value of 4 implies a strong filter (also referred to as 'S'in this paper), 1,2,3 means a weak filter (also 'W'in this paper) whereas a value of 0 means no filtering is applied on this specific edge. Based on Table 1, we find that there are two types of edges that exist in every macroblock - the MB edges (edges of the 4x4 blocks which form the macroblock edges) and the Non-MB edges (edges of the 4x4 blocks inside a macroblock which don't form MB edges).

Fig.1 shows an example where two block edges need to be filtered. p_i, q_i and s_i are the pixel values on either side of the edges. Up to three sample values for luminance and one for chrominance on each side of the edge may be modified by the filtering process. For the block edge 1 shown in Fig.1,

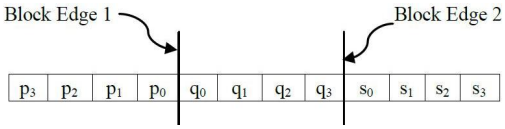

Fig. 1. Two edges with unfiltered pixel values on either side

filtering takes place if all the three conditions (1)-(3) are satisfied. The condition constraints are a function of both the table-derived thresholds α and β which are dependent on the average quantization parameter (QP) of the two blocks and the index values that are calculated using encoder selected offset values.

$$(1)\ p_0 - q_0 < \alpha(Index_A) \qquad (2)\ p_1 - p_0 < \beta(Index_B)$$

$$(3)\ q_1 - q_0 < \beta(Index_B) \qquad (4)\ p_2 - p_0 < \beta(Index_B)$$

$$(5)\ q_2 - q_0 < \beta(Index_B) \qquad (6)\ p_0 - q_0 < (\alpha >> 2 + 2)$$

For both filtering modes, in addition to conditions (1)-(3), the threshold value is used to evaluate two additional spatial activity conditions (4) and (5) that are used to determine the extent of the filtering in the case of luminance samples. For Bs = 4 in particular, (6) is also checked to decide on the number of pixels to be modified on each side of the edge along the line of pixels. Depending on which set of conditions are true from (1)-(6), the strength of filtering is decided.

In the traditional adaptive deblocking filter, filtering is conducted 'in-place', so that the modified sample values are used as input values to subsequent filtering operations. For Bs = 1, 2 and 3, a basic filtering operation followed by a clipping operation is performed. For Bs = 4, the filtering operation decision is made based on the image content between a very strong 4- and 5-tap filter that modifies the edge sample and two interior samples on each side, or a weaker 3-tap filter that modifies only the edge samples. For ease of explanation of the different cases, we refer to them as W.x and S.x. The different filtering cases have been specified briefly below without the formulae. Please refer to [5] for more details.

For Bs = 1,2,3, the filtering conditions are given as :

W.1 : Only if (1), (2), and (3) are satisfied, p_0 and q_0 get modified with the delta function [5].

W.2 : If (1)-(5) are satisfied, p_1, p_0, q_0, and q_1 get modified.

W.3 : If (1)-(4), are all satisfied, p_0, q_0, and q_1 get modified.

W.4 : If (1)-(3), and (5), are satisfied, p_0, q_0, and p_1 get modified.

For Bs = 4, the filtering conditions are :

S.1 : If (1)-(6) are all satisfied, p_2, p_1, p_0, q_0, q_1, and q_2 get modified by the 5-tap filter.

S.2 : If (1), (2), and (3) hold good, and combinations of either (4) and (6), or (5) and (6) are not satisfied, only p_0 and q_0 get modified by the 3-tap filter.

S.3 : If (1)-(3), (5), and (6) are satisfied, p_0, q_0, q_1 and q_2 get modified by the 4-tap filter.

S.4 : If (1)-(4), and (6) are satisfied, p_2, p_1, p_0 and q_0 get modified by the 4-tap filter.

W.1 through W.4 and S.1 through S.4 expresses the different cases for the Bs = 1,2,3 and the Bs = 4 filtering respectively.

3. PROPOSED PARALLEL ALGORITHM

3.1. Analysis of Parallelism

In this section, we analyze the parallelism that could possibly be exploited to facilitate a completely scalable deblocking filter design. Let us consider the two adjacent block edges shown in Fig.1. There could be a maximum of 32 different cases for filtering in two adjacent block edges. Depending on the strength and the filtering cases, the number of pixels modified on either side of the block edge varies. We introduce two terms: the region of modification which is the region on either side of the block edge where the pixel values get modi-

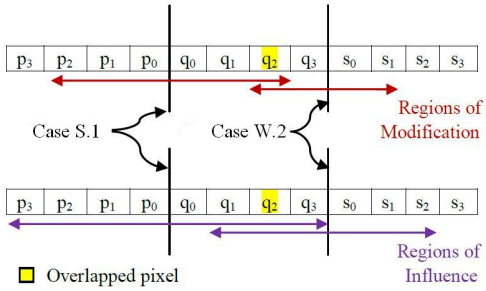

Fig. 2. Regions of Modification and Influence

fied due to filtering and the region of influence which includes all the pixels which are involved in the filtering of the block edges. Fig.2 shows the regions of influence and modification for a pair of block edges. For example, in Fig.2, block edge 1 modifies three pixels to its right and block edge 2 modifies two pixels to its left, and hence pixel q_2 is modified twice. This is the reason why consecutive edges could not be processed simultaneously in prior designs.

By examining the different possible combinations of filtering cases, we identify the conflict cases i.e., the cases in which filtering of both edges cannot be performed in parallel. These are shown in Fig.3. In this figure, the directed edge (i, j) between node i and node j stands for a conflict if node j is executed after node i. It is seen that out of the possible 32 combinations, only 12 combinations have a conflict and are responsible for the non-parallelizable nature of the deblocking filter. All the other cases are inherently parallel, i.e., the

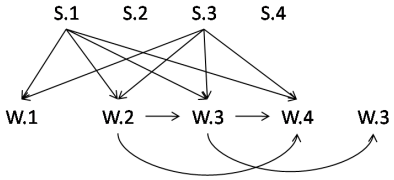

Fig. 3. Conflict cases for two edges processed in parallel

pixel values on either side of the block edges can be modified simultaneously.

In order to analyze the frequency of occurrence of these 8 non-parallelizable cases, we used a series of QCIF test video sequences (300 frames) with varying motion like Foreman, News, Akiyo, Hall, Highway, Coastguard, Carphone, and Mother-Daughter. The percentage of conflicts have been tabulated in Table 2. From the table, we see that on an average, these twelve conflict cases feature in 1.126 % of the total filtering cases when two edges are processed in parallel. Similarly, we find that the average percentage of conflicts for processing 4 edges (includes three edges) is 0.738 %, and the average percentage for processing 8 consecutive edges simultaneously is about 1.5 %. From this analysis, we conclude that

Table 2. Percentages of conflicts for two, four and eight edges

Video	Conflict % for 2 edges	Conflict % for 4 edges	Conflict % for 8 edges
Akiyo	0.174	0.123	0.127
Carphone	1.650	1.131	1.252
Coastguard	1.894	0.364	0.849
Foreman	1.960	1.885	1.511
Hall	0.476	0.486	0.304
Highway	1.535	1.229	1.333
Mother-Daughter	0.544	0.239	0.416
News	0.775	0.449	0.645
Average	**1.126**	**0.738**	**0.805**

there is significant parallelism inherent in the algorithm. This inherent parallelism is exploited in the proposed architecture in this paper.

3.2. Parallelizing the Conflict Cases

The conflict cases shown in Fig.3 need to be addressed in parallel so as to make filtering of two edges in the deblocking filter data independent. Analyzing the 12 conflict cases, we find that they fall into two different categories: overlap of the region of modification of the first edge with the region of influence of the second edge, and overlap of the regions of modification of both edges. Examples of category 1 are filtering block edges S.1 and W.1, S.1 and W.3, S.3 and W.1, S.3 and W.3, W.2 and W.2, W.2 and W.4, W.3 and W.2, and W.3 and W.4. Block edges S.1 and W.2, S.1 and W.4, S .3 and W.2, and S.3 and W.4 fall into category 2. For each category, we consider pair of cases and work out the modifications to be carried out for parallelizing operations.

From category 1, let us consider the conflict case where the two block edges to be filtered are S.1 and W.1. Using the same notations as in Fig.1, we see that when filtering block edge 1, the pixels p_2, p_1, p_0, q_0, q_1 and q_2 are modified as before. However, when filtering block edge 2, since q_2 gets modified to q_2', a new delta function Δ_1 has to be calculated using q_2', q_3, s_0 and s_1 and then used to calculate q_3' and s_0'.

$$\Delta_1 = Clip((((s_0 - q_3) << 2) + (q_2' - s_1) + 4) >> 3) \quad (7)$$

$$q_3' = Clip(q_3 + \Delta_1) \qquad s_0' = Clip(s_0 - \Delta_1) \quad (8)$$

where the $Clip$ function prevents an overflow by constraining the values between 0 and 255.

Hence, if we have the hardware to compute the 'new delta' as in (7) directly, then we could parallelize computations and filter both the edges simultaneously and get all the filtered pixels (p_2' to s_1') out at the same time.

Similarly from category 2, let us consider the case where the two block edges to be filtered are S.1 and W.2. While filtering the first block edge, p_2, p_1, p_0, q_0 and q_1 get modified.

q_2 gets modified twice and is given by

$$q_2'' = q_2' + Clip(q_1' + ((q_3 + s_0 + 1) >> 1 - (q_2' << 1)) >> 1) \quad (9)$$

where q_2' and q_1' are modification that occur due to the first edge filtering. q_3' and s_0' are calculated as in (8). Similar to the conflict category 1, if we have the hardware to compute the final q_2 (or q_2'') directly as in (9), then we could filter both the block edges in parallel. In the proposed architecture, we have accounted for these modifications to address all 2-edge conflicts. Specifically, we introduce extra hardware blocks in the filtering units PE1 and PE2 (shown in Fig.5 and the shaded blocks of the Fig.6) to resolve these conflicts.

To prove the correctness of the proposed algorithmic modifications to address the two edge conflicts, we modified the deblocking (loop filter) module of the JM H.264/AVC software to process two edges at a time (as opposed to a single edge in the original implementation) and filter the whole video sequence. We encoded eight videos with both the original implementation and our proposed implementation and compared the SNR quality of the Y component and bit rate of the encodings. We observe that the algorithmic modifications result in no loss in quality and encode the videos with the same bit-rate, as expected.

3.3. Design Scalability

The two edge parallelization algorithm can be extended to filtering four and eight edges simultaneously. When 4 consecutive edges are processed, there are 256 possible filtering cases. Of these, only 16 combinations have a conflict and these 16 combinations occur on an average 0.738% of the time (as shown in Table 2). For instance, one of the possible 4-edge conflicts is: S.1 for block edge1, W.2 for edge2, W.2 for edge3 and W.4 for edge4. In a similar note, when 8 consecutive edges are processed, the conflicting combinations occur on an average 0.805% of the time (excluding 2-edge and 4-edge conflicts).

4. PROPOSED ARCHITECTURE

4.1. Architecture Overview

The block level architecture of the deblocking filter application specific processor is shown in Fig.4. It consists of three PEs which work on filtering two edges in parallel, a vector register file to hold several 4x4 blocks locally, a Bs unit to decide on the filtering strength, a Clip-LUT unit for clipping the filtered values, an alpha-beta module to compute the offset dependent threshold parameters, a branch-comp module which computes the branch conditions and the intermediate thresholds (conditions (1) - (6)) and finally the control unit which decides on whether the edge has to be loaded into the PE for filtering or not.

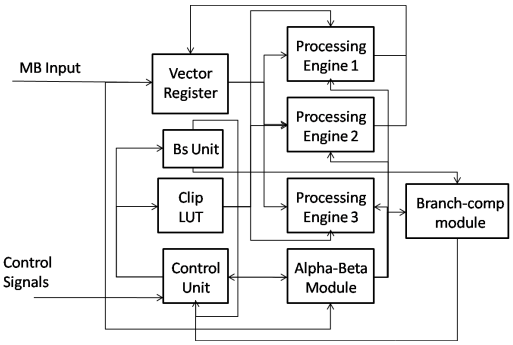

Fig. 4. Block level architecture of the Deblocking filter

One of the important considerations in deciding the datapath of the proposed architecture is the number of modules needed to process two edges in parallel. On analyzing the ratio of occurrences of Bs = 4 and Bs = 1,2,3 filtering strengths, we find that the weak filter occurs way more frequently than the strong filter (approximately in the ratio 10:1). Hence, we propose to have two modules to filter edges with boundary strength Bs = 1,2,3, (PE2 and PE3) and one module to filter edges with boundary strength Bs = 4 (PE1).

The block diagram of the proposed architecture is shown in Fig.5. At any time, only two of the three PEs are used for filtering 2 edges in parallel. For instance, if the two adjacent edges are edges with Bs = 4 and a Bs = 1,2,3, PE1 and PE3 are used for filtering, and the additional hardware in PE1 computes the conflict pixels (if any). Similarly, if the two adjacent edges are both Bs = 1,2,3, PE2 and PE3 are used and the additional hardware in PE2 computes the conflict pixels (if any). In each PE, the architecture has been designed to use minimal number of adds and shifts for the filtering operations. This has been done by computing many intermediate values and reusing them at various stages.

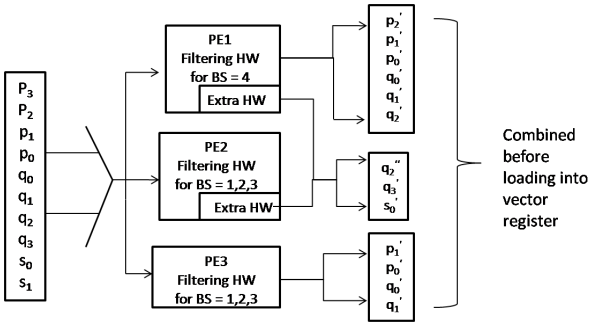

Fig. 5. Proposed architecture addressing 2-edge conflicts

The schematic of PE2 is shown in Fig.6. The schematic has the base implementation of the Bs = 1,2,3 filter (shown solid in Fig.6) along with additional hardware to address the 2-edge conflicts (shown shaded in Fig.6). The schematic of the PE1 is similar, and consists of base implementation of the

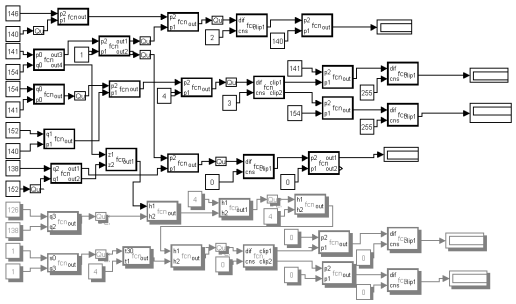

Fig. 6. PE2 with Bs = 1,2,3 filter implementation (solid) and additional hardware (shaded)

Bs = 4 filter and additional hardware to resolve the conflicts.

4.2. Synthesis Results

The proposed architecture was implemented in Verilog. The design was synthesized using 45-nm CMOS technology and analyzed in terms of area, power, and throughput. H-SPICE was used to characterize the gates and HDL synthesis was used for estimating the switching energy of the deblocking filter system. The data paths of the original deblocking filter which processes one edge at a time [5] and the proposed architecture which processes two edges were implemented. The original deblocking filter architecture has two modules - one strong filter and one weak filter unit, while the proposed architecture has three modules - one strong filter and two weak filter units. Table 3 shows the number of NAND gates and inverter gates in the datapath of each of the architectures. These were used to generate the area estimates which also includes 20 % overhead for routing. From Table 3, we see that the area of datapath of the proposed architecture is 2.2 times the area of the original deblocking filter implementation. Table 4 shows the results of the switching energy spent to deblock each video for both the architectures. On an average it is observed that the power numbers for the proposed parallel architecture is about 1.23 times the power of the original sequential implementation. Hence, for a 2.2x area and 1.23x increase in dynamic power, the proposed architecture obtains twice the throughput.

Table 3. Deblocking Filter: Datapath Area Comparison

Architecture	# of nand gates	# of inverter gates	Area (in um sq.)
Original	2779	5866	120.79
Proposed	6223	12918	268.64

Table 4. Switching Energy comparison

Input video	Original Architecture Energy (nJ)	Proposed Architecture Energy (nJ)	Ratio wrt original architecture
Akiyo	59.04	71.63	1.21
Carphone	430.65	529.66	1.23
Coastguard	639.27	840.91	1.31
Foreman	398.79	487.23	1.22
Hall	112.30	137.16	1.22
Highway	298.78	356.11	1.19
Mother-Daughter	165.46	204.38	1.24
News	209.63	261.64	1.25

4.3. Extension for Processing 4 and 8 Edges

In Section 3.3, we have seen that 4-edge conflicts occur less than 1% of the time. We could design a specialized architecture which handles these 4-edge conflicts and processes four edges every time. Instead, in this work, we have fixed the architecture to cater to processing two edges in parallel and increase the number of PEs for the 4-edge and 8-edge cases. For example, for the 4-edge case we use 5 PEs (we add two PE2 modules to the proposed architecture). This helps us maintain a throughput of 4 edges 99.262% of the time, and during a conflict (for the rest 0.738 % of the time), process just 2 edges. We use a similar technique to filter eight edges.

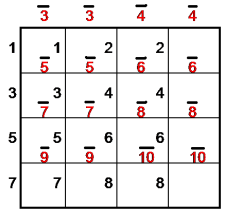

Fig. 7. Scheduling of vertical and horizontal (bar on top) edge filtering for processing 4 edges simultaneously

We can increase the throughput even further by processing horizontal and vertical edges in parallel. By duplicating the proposed 2-edge hardware, we can process 2 horizontal and 2 vertical edges simultaneously. Fig.7 shows one such schedule for processing 4 edges.

5. RELATED WORK

Several architectures have been proposed to deal with the sequential order of filtering in the deblocking filter. Cheng et al. [6], [7] proposed an in-place computing architecture with data reuse to improve performance. In [8], Sheng et al. introduced a memory mapping method for efficient deblocking computation. Huang et al. [12] implemented an array of shift registers with reconfigurable data paths to support both vertical and horizontal filtering. Recently, Dang proposed an

application specific hardware architecture [11] which incorporated several techniques to reduce complexity and increase the throughput. Dang's method processes a horizontal edge and a vertical edge in parallel. In comparison, our architecture can process two or more horizontal and vertical edges in parallel. More recently, Wang [13] proposed a method which exploits data level parallelism with a technique called 'limited propagation effect' where a frame is partitioned into multiple independent rectangles and these are processed in parallel. The method shows a speedup in throughput, but is not scalable. In terms of gate count, the proposed method has a gate count of 19.14K which is less than 21.2K for [9], 20.66K for [12] and 24K for [8]. Moreover, these methods process just one filter edge at a time as opposed to at least two edges by the proposed method. Dang's method has a higher gate count of 24.4K since it includes hardware costs for memory and address decoder.

In summary, all the above methods [6]-[13] retain the sequential order in which the edges are processed and compensate for this dependency by intelligently scheduling and pipelining the filter operations. On the contrary, we approach the problem by identifying the cause of the dependency and proposing an efficient solution to eliminate it.

6. CONCLUSION

This paper presents a scalable solution to parallelizing the adaptive deblocking filter algorithm in H.264. At the algorithm level, a new scheme was introduced which process two edges at a time, thereby doubling the throughput. We mapped the 2-edge method onto an architecture and characterized its performance. Our method is scalable and can process 4- and 8- consecutive edges simultaneously, if hardware is not a constraint. We can increase the throughput even further by using multiple copies of the proposed architecture to process multiple horizontal and vertical edges simultaneously.

Acknowledgements

This work was supported in part by NSF grant CSR-0615135 and by ASU-IETSM Collaborative Information Technology grant program

7. REFERENCES

[1] T. Wiegand, G. Sullivan, and A. Luthra, "Draft ITU-T recommendation and final draft international standard of joint video specification," 2003.

[2] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[3] T. Stockhammer, M. Hannuksela, and T. Wiegand, "H.264/AVC in wireless environments," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 657–673, July 2003.

[4] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704–716, July 2003.

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

[6] C.-C. Cheng and T.-S. Chang, "An hardware efficient deblocking filter for H.264/AVC," in *Proceedings of International Conference on Consumer Electronics, ICCE*, Jan 2005, pp. 235–236.

[7] C.-C. Cheng, T.-S. Chang, and K.-B. Lee, "An in-place architecture for the deblocking filter in H.264/AVC," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 7, pp. 530–534, July 2006.

[8] B. Sheng, W. Gao, and D. Wu, "An implemented architecture of deblocking filter for H.264/AVC," in *Proceedings of International Conference on Image Processing ICIP*, vol. 1, 2004, pp. 665–668.

[9] C.-M. Chen and C.-H. Chen, "A memory efficient architecture for deblocking filter in H.264 using vertical processing order," in *Proceedings of the Intelligent Sensors, Sensor Networks and Information Processing Conference*, vol. 2005, 2005, pp. 361 – 366.

[10] G.-Q. Zheng and L. Yu, "An efficient architecture design for deblocking loop filter," in *Picture Coding Symposium 2004*, 2004, pp. 343–347.

[11] P. Dang, "High performance architecture of an application specific processor for the H.264 deblocking filter," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 10, pp. 1321–1334, Oct. 2008.

[12] Y.-W. Huang, T.-W. Chen, B.-Y. Hsieh, T.-C. Wang, T.-H. Chang, and L.-G. Chen, "Architecture design for deblocking filter in H.264/JVT/AVC," in *Proceedings of International Conference on Multimedia and Expo, ICME*, vol. 1, July 2003, pp. 693–696.

[13] S.-W. Wang, S.-S. Yang, H.-M. Chen, C.-L. Yang, and J.-L. Wu, "A multi-core architecture based parallel framework for H.264/AVC deblocking filters," *J. Signal Process. Syst.*, vol. 57, no. 2, pp. 195–211, 2009.