J Sign Process Syst
DOI 10.1007/s11265-008-0321-4

A Multi-core Architecture Based Parallel Framework for H.264/AVC Deblocking Filters

Sung-Wen Wang · Shu-Sian Yang · Hong-Ming Chen · Chia-Lin Yang · Ja-Ling Wu

Received: 15 September 2007 / Revised: 30 April 2008 / Accepted: 11 November 2008

Abstract Deblocking filter is one of the most time consuming modules in the H.264/AVC decoder as indicated in many studies. Therefore, accelerating deblocking filter is critical for improving the overall decoding performance. This paper proposes a novel parallel algorithm for H.264/AVC deblocking filter to speed the H.264/AVC decoder up. We exploit pixel-level data parallelism among filtering steps, and observe that results of each filtering step only affect a limited region of pixels. We call this "*the limited propagation effect*". Based on this observation, the proposed algorithm could partition a frame into multiple independent rectangles with arbitrary granularity. The proposed parallel deblocking filter algorithm requires very little synchronization overhead, and provides good scalability. Experimental results show that applying the proposed parallelization method to a SIMD optimized sequential deblocking filter achieves up to 95.31% and 224.07% speedup on a two-core and four-core processor, respectively. We have also observed a significant speedup for H.264/AVC decoding, 21% and 34% on a two-core and four-core processor, respectively.

Keywords Video coding · H.264/AVC · Deblocking filter · SIMD · Parallel algorithm

Ja-Ling Wu is a Fellow IEEE.

S.-W. Wang (✉) · S.-S. Yang · H.-M. Chen · C.-L. Yang · J.-L. Wu
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
e-mail: song@cmlab.csie.ntu.edu.tw

S.-S. Yang
e-mail: pigyoung@cmlab.csie.ntu.edu.tw

H.-M. Chen
e-mail: blacksmith@cmlab.csie.ntu.edu.tw

C.-L. Yang
e-mail: yangc@csie.ntu.edu.tw

J.-L. Wu
e-mail: wjl@cmlab.csie.ntu.edu.tw

1 Introduction

A content adaptive in-loop deblocking filter has been adopted by H.264/AVC. Unlike traditional post-loop filters, the in-loop filter is part of the frame reconstruction process. Any inappropriate modification of the in-loop filter causes serious error propagation to succeeding frames. The deblocking filter adopted by H.264/AVC, though offers 10–15% bit rate saving, suffers from heavy computational load due to the high data dependency among filtering stages [1]. Lappalainen et al. proposed an optimized implementation of H.264/AVC decoder [2], in which the profiling results show that image interpolation and deblocking filter contributes to a significant part of the decoding time. For HDTV applications, Chen et al. [3] also analyzed the computation profiles of H.264/AVC decoder with HDTV1024p video clips. Based on their analysis, the computational time contributed by the modules of interpretation and deblocking filter are 39% and 38%, respectively. Therefore, to improve the H.264/AVC decoding performance, accelerating deblocking filter is critical.

Many recent studies focused on designing dedicated hardware to improve the data transmission rate of deblocking filter. Huang et al. [4] proposed SRAM modules with carefully organized data structure for the efficient access of pixels in different blocks. To simplify data exchange between deblocking filter and outside data memory, Sima et al. [5] designed a compact data access unit and a build-in data buffer. To efficiently utilize the bus bandwidth, by classifying the filtering mode, Chang et al. [6] used an adaptive transmission scheme to avoid redundant data transfer. Cheng et al. [7] proposed an in-place computing flow to reuse the intermediate data. To support the deblocking operation for multi-channel video applications, Khurana et al. [8] proposed a pipelined hardware implementation of in-loop deblocking filter. The dedicated hardware approach can achieve highly efficient deblocking operations but it is lack of flexibility. A new standard often requires a whole new design of a dedicated hardware.

Single instruction multiple data (SIMD) is a commonly used approach to improve the performance of media processing on a general purpose processor, such as, MMX, SSE1, SSE2 and SSE3[9]. Warrington et al. [10] applied SIMD instructions to improve the performance of H.264/AVC deblocking filtering. Nevertheless, the complexity of the deblocking filter comes mainly from the huge amount of data comparisons involved. Each 4×4 block has to be examined by series of conditional checks. The requirement of pixel-by-pixel conditional checks limits its performance gain obtainable from SIMD instructions. Unlike the interpolation operations, the performance of deblocking filter cannot easily be enhanced by using SIMD instructions only. As shown in [11] the speedup ratios for chrominance motion compensation and that of deblocking filter are 10.2 and 1.1, respectively. Figure 1 shows the execution time distribution of a SIMD-optimized decoder/footnote. The SIMD-optimized decoder used in this study is derived on the basis of our previous work [12]. We can see that deblocking filter still accounts for 38.6% of the execution time of SIMD-optimized H.264/AVC decoding.

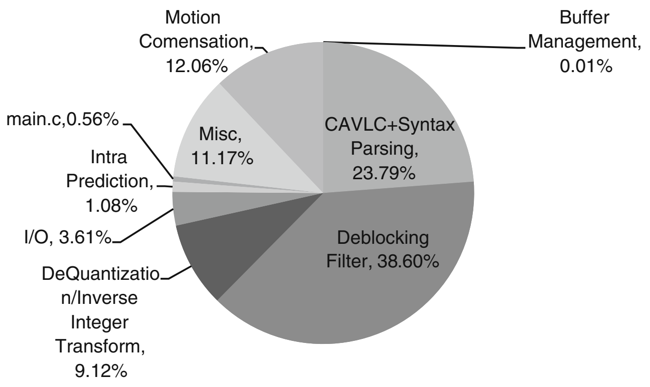


Figure 1 Time breakdown of SIMD-optimized H.264/AVC decoder.

In addition to conventional optimization schemes, with the increasing popularity of multi-core processors, parallelizing heavy computational tasks becomes an effective way for performance enhancement. Unfortunately, it is not easy to develop effective algorithms that can take advantage of parallel processors. Parallelism does not guarantee speedup. Without cautious arrangement, the associated synchronization overhead may impair the benefits of parallel execution dramatically. In this work, we propose a novel parallel algorithm for H.264/AVC in-loop deblocking filter that requires very little synchronization overhead. The proposed algorithm is based on the observation that results of each filtering step only affect a limited region of pixels. This is referred to as "*limited propagation effect*" in this paper. Based on this observation, the proposed parallel deblocking filter algorithm partitions a frame (or a region containing multiple macroblocks) into d independent rectangles, where d is the number of available processors. Since there are no dependencies among rectangles allocated to different processors, a processor needs only to perform synchronization once in the entire frame. Therefore, the proposed parallelism needs only d synchronization operations. Experimental results show that applying the proposed parallelizing approach to a SIMD optimized sequential deblocking filter achieves up to 95.31% and 224.07% speedup for two-core and four-core processors, respectively. The synchronization overhead of the proposed algorithm is almost negligible (less than 1%) for both 2-core and 4-core processors. In contrast, the well-known wavefront parallelization approach [24] does not show performance improvement on SIMD optimized codes due to significant synchronization overhead, up to 30.78% of the deblocking execution time. The experimental results also show that our scheme achieves a significant speedup for H.264/AVC decoding, 21% and 34% on a two-core and four-core processor, respectively.

The remainder of this paper is organized as follows. The related works are introduced in Section 2. A brief summary of deblocking filtering and the conventional parallel deblocking filtering method are described in Section 3. Section 4 addresses the "limited propagation effect" in the deblocking filter of H.264/AVC. Then, we present the proposed parallel deblocking filtering algorithm in Section 5. Experimental results are given in Section 6. Finally, the concluding remarks and future research directions are addressed in Section 7.

2 Related Works

In general, there are two ways to parallelize an application: task-level parallelism and data-level parallelism. Task-level parallelism assigns independent tasks to different processors and data-level parallelism process independent data concurrently. Task-level parallelism is more suitable for asymmetric multi-core architecture since the loading of each task is usually not balanced [13, 14]. Task-level parallelism are also preferred in hardware implementations [15–19], software-hardware co-design [20], and software pipelining [25]. For parallelizing video coding on symmetric multi-core architecture, data-level parallelism is more preferable.

A video sequence is composed of a series of *Group of Pictures* (GOPs) and a GOP contains a series of frames. A video frame comprises several slices which are independent to each other and can be further divided into non-overlapped MacroBlocks (MBs). This hierarchical data structure naturally provides different level of granularities and gives different degree of effects on parallelism. From the GOP level to slice level, Rodriguez et al. [21] proposed a hierarchical parallelization encoding scheme. In [22], Chen et al. presented a frame level to slice level data parallelization video coding scheme. In general, to fully utilize the available cores, the number of slices per frame should be large when the number of cores increases.

Without doubt, a significant speedup of slice-level parallelism can be expected because less synchronization and communication overheads existing in the slice level; however, it brings negative impacts as well. Considering the encoding efficiency, the result of [22] and [23] show that more independent slices degrades the bit-rate performance. The main reason is that, in this case, MBs cannot exploit the correlation across slice boundaries. It is clear that speedup can be obtained from encoding independent slices in parallel; however, decoding multiple independent slices in parallel is inapplicable for a client because the number of multiple slices for a frame cannot always be guaranteed. For example, if only one slice completes a frame, the number of slices that can be processed in parallel is limited due to their temporal prediction dependencies.

After analyzing the data dependency among MBs, it is clear that the current MB relies only on its neighboring four MBs, i.e. Top-left, Top, Top-right, and left ones. By rearranging the data partition and task scheduling, several independent MBs can be executed concurrently. Since the positions of independent MBs are wavefront-like, this kind of parallelism method is named as the wavefront parallelization. For the encoder, the wavefront MB decomposition has been adopted in [26], which provides good coding efficiency. For the decoder, similar approaches has also been addressed in [25, 28]. More detailed information about MB wavefront parallelization will introduced in Section 3.2.

MB-wavefront provides a finer granularity of parallelism than slices and prevents the data dependency violation. However, as compared with slice-level parallelism, MB-level parallelism has two disadvantages. The first one is that entropy decoding cannot be parallelized in MB-level. An H.264/AVC video bitstream is composed of a series of *Network Abstraction Layer* Units (NALs) with fixed length headers [27]. Since the length of the header is fixed, the *Slice* type which is the lowest level of NAL type can be individually parsed from the bitstream. Nevertheless, MBs cannot be individually extracted without conducting entropy decoding in order. That is, independent MBs can be simultaneously issued only when entropy decoding has been performed. The second shortcoming is the penalty brought from finer granularity of parallelism. The wavefront MB decomposition requires frequent synchronization among processing units, which could possibly impair the speedup obtainable from parallel processing.

3 Background

3.1 Deblocking Filtering of H.264/AVC

H.264/AVC adopts an *in-loop* deblocking filter for removing block-edge artifacts that are introduced by block transformation and block motion compensation. As the name of "In-loop" implies the moment to apply filtering while prior reconstruction step is within the motion compensation loop. Figure 2 shows the block diagram of H.264/AVC decoding process and where the insertion point of deblocking filter is. To adaptively alleviate blocking artifact, the deblocking stage is divided into three levels: slice level, block level and pixel level. At the slice level, encoder sets an overall strength for the filter. At the block level, a 4×4 block-dependent *Boundary Strength* (BS) value is determined according to the coding information, such as coding mode and coded residues. Finally, at the pixel level, pixels on either side of an edge are examined and then filtered. More specifically, if the difference between these pixels is lower than a *Quantization Parameter* (QP) dependent threshold and the slice level offset, the edge is considered as a blocking artifact. The filtering stage is applied to each 4×4 luminance and chrominance block edge within each MB, in a specific order, as shown

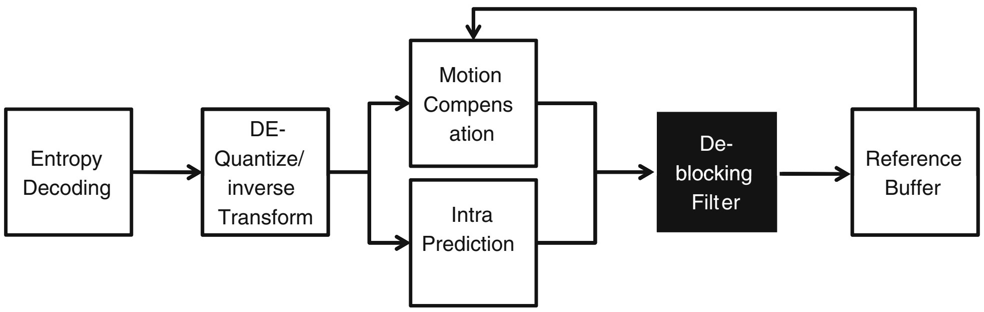


Figure 2 Block diagram of H.264/AVC decoding process.

in Fig. 3: vertical boundary edges are filtered first, followed by the horizontal ones.

All filtering steps are taken place from left to right and from top to bottom. Moreover, MBs are processed in a raster-scan order over the frame. This particular filtering order must be followed to ensure both the encoder and the decoder obtaining the same result.

For MB boundary (MB-B) edges, four pixels on either side of each edge are evaluated. Up to three pixels on either side of the edge may be influenced. Non-MB-boundary (N-MB-B) edges are less affected by the blocking effect, thus, processed by a filter with shorter length. Three pixels on either side of an N-MB-B edge are evaluated, and only up to two pixels on either side may be influenced. By convention [27], pixels on either side of an edge in adjacent blocks p and q are denoted by p_i and q_i with $i = 0,...,3$, as shown in Fig. 4. More details about the edge filtering can be found in [27]. Table 1 summarizes all possible influenced pixels and their respective referenced pixels.

Since the steps of boundary edges filtering within an MB obey the prescribed ordering (c.f. Fig. 3), the evaluated pixels on an specific edge filtering depend on the filtered pixels values of its previous edge filtering. We divide the influences upon pixels into two parts: the impact from the filtering of current MB and the impact from adjacent MBs. That is, the pixels influenced by the antecedent edge filtering will be used as the pixels of the current MB in the subsequent edge filtering. Moreover, pixels of the current MB will affect the values of pixels for its adjacent MBs due to MB-boundary filtering.

3.2 MB-Wavefront Realization of Deblocking Filter and Its Synchronization Overhead

The wavefront method is a commonly used data partition method for parallelization[24]. Data are processed in the wavefront order as shown in Fig. 5. Data blocks marked the same number are computed concurrently. For H.264/AVC decoding, since the computation of the current MB depends on the data of its neighboring four MBs, we can group the MB computations along the $\pi - \arctan\frac{1}{2} \simeq 153°$ diagonals. Therefore, the wavefront parallelization method can be adopted to parallelize H.264/AVC decoding [28]. This method is called MB-wavefront in this paper. In this section, we discuss how the wavefront approach is applied to deblocking filter, and the associated synchronization overheads.

The MB-wavefront realization of deblocking filter is demonstrated by the pseudo codes shown in Algorithm 1. In Algorithm 1, the application programming interface (API) of pthread_mutex_lock, pthread_mutex_unlock and pthread_cond_wait are provided by

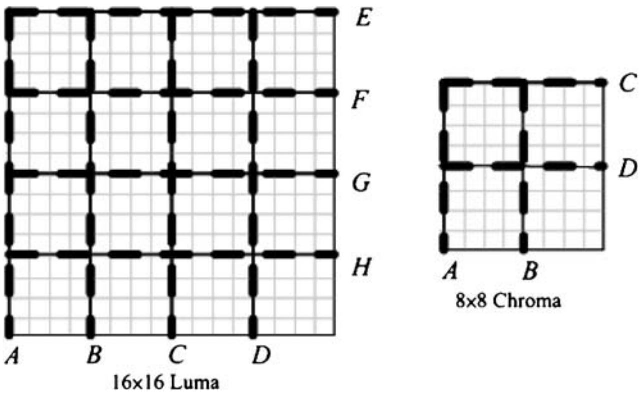


Figure 3 Edge filtering order in an MB.

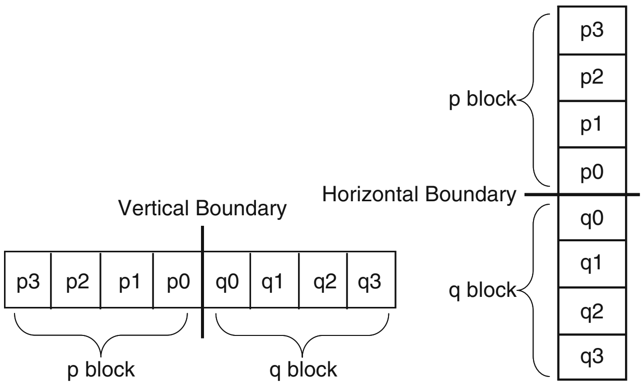


Figure 4 Pixels adjacent to vertical and horizontal boundaries.

Table 1 Pixels which could be influenced and evaluated for MB-B and N-MB-B filters.

Filter type	Filtered pixel	Evaluated pixels
MB-B filter	p_2	$p_3\ p_2\ p_1\ p_0\ q_0$
	p_1	$p_2\ p_1\ p_0\ q_0$
	p_0	$p_2\ p_1\ p_0\ q_0\ q_1$
	q_0	$q_2\ q_1\ q_0\ p_0\ p_1$
	q_1	$q_2\ q_1\ q_0\ p_0$
	q_2	$q_3\ q_2\ q_1\ q_0\ p_0$
N-MB-B filter	p_1	$p_2\ p_1\ p_0\ q_0$
	p_0	$p_1\ p_0\ q_0\ q_1$
	q_0	$p_1\ p_0\ q_0\ q_1$
	q_1	$q_2\ q_1\ q_0\ p_0$

an open library: POSIX threads (http://sourceware.org/pthreads-win32/). In general, to avoid spending too much time on creating and destroying threads in POSIX thread library, we could include the design of *thread pool* [29] into the parallel programs. The function *thread pool* provides two important functionalities: a pool of concurrent working threads, *workers* and a queue for waiting service tasks, where the workers sustainedly consume tasks from the queue. The *thread pool* creates all workers at the beginning of the program and destroys all at the end. Meanwhile, to avoid race condition, the number of workers should be equal to the number of available processors in our parallel program. The queue size is decided by the maximal number of concurrent tasks. For example, the queue sizes of the MB-wavefront are 40, 23 and 11 for 720p, 480p and CIF test sequences, respectively. Line 7 in Algorithm 1 uses the design of the *thread pool* as described above. Every wavefront iteration introduces a synchronization barrier, from Line 9 to Line 13 in Algorithm 1.

The synchronization of the wavefronting parallelism occurs at wavefront boundaries. More specifically, referring to Algorithm 1, the iteration number,w, of the outmost loop directly affects the number of synchronization. In addition to the synchronization

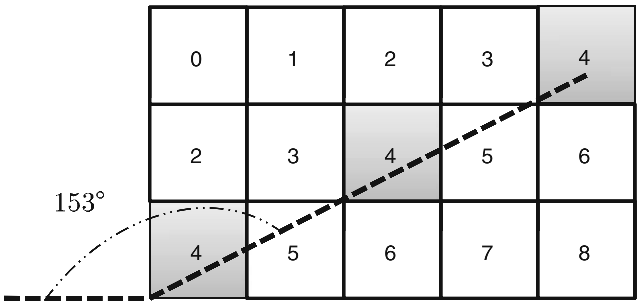


Figure 5 A wavefronting example: Each rectangular represents an MB. The number d in each MB indicates the d-th iteration execution.

Algorithm 1 The Wavefronting Scheduling

```
1:  $w \leftarrow$ the number of wavefront iterations
2:  $map_i \leftarrow$ MB positions of the $i_{th}$ wavefront iteration
3:  $MB_i \leftarrow$ the number of MBs of the $i_{th}$ wavefront iteration
4:  count $\leftarrow 0$
5:  for $i \leftarrow 1, w$ do
6:      for all MBs within $map_i$ do
7:          add Deblock(MB) into queue
8:      end for
9:      pthread_mutex_lock
10:     while task queue is not Empty do        ▷ wait until all tasks are finished
11:         pthread_cond_wait
12:     end while
13:     pthread_mutex_unlock
14: end for
15: procedure Deblock(MB)
16:     Retrieve data from left, top-left and top of the current MB
17:     Processing Deblocking MB abides by [27]
18: end procedure
```

between wavefront iterations, within each iteration, synchronization is also required among workers (i.e, the available processors). Therefore, the amount of synchronization overhead incurred in MB-wavefront is $d \times w$, where d is number of processors and $w = M_h + (M_v - 1) \times 2$. M_h and M_v are the horizontal and vertical frame size, in terms of the number of MBs, respectively. Normally, the w is a large integer. For example, M_h and M_v for a 1280×720 (i.e., 720p) video sequence are 80 and 45, respectively (i.e, $w = 168$). Therefore, as the number of cores increases, the synchronization overhead $d \times w$ is also expected to grow substantially.

Another drawback of the MB-wavefront method is poor cache utilization. If MBs with dependency are allocated to different processors, frequent data exchange between processors are expected. This results in poor cache utilization, and also incurs significant cache coherency traffic.

4 Limited Propagation Effect of Filtering Stages

From Section 3.1, we knew why edge filtering will affect the subsequent filtering steps. In the following, we will characterize how far the effect of each filtering step can propagate. The influence of filtered pixels comes mainly from the stage of MB boundary filtering and the stage of block edge filtering within each MB. To find out the

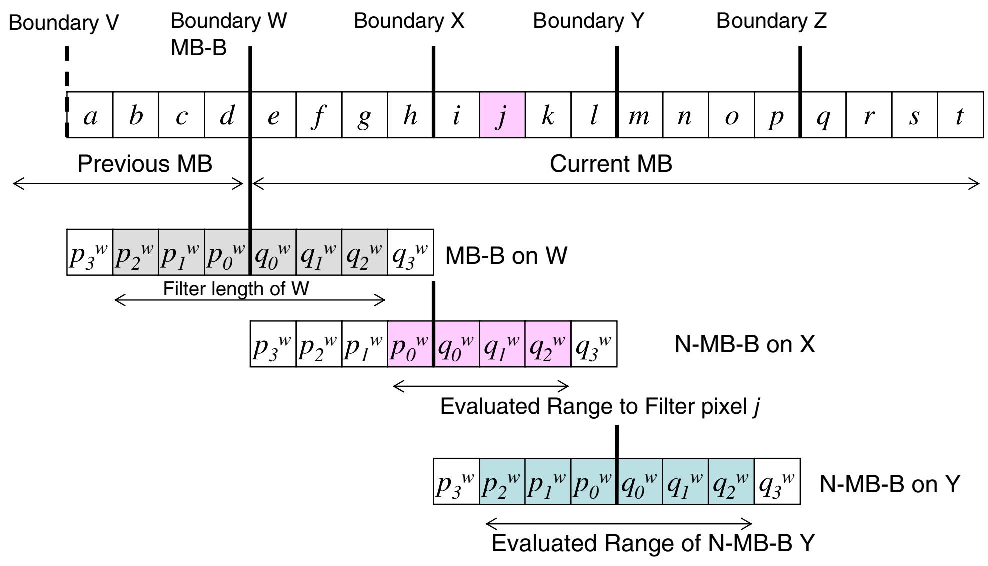


Figure 6 The *superscript* of each *pixel* indicates its identity. *Shaded pixels* may be modified during succeeding filtering stages.

maximal range of influence, we investigate the pixels which are influenced by the MB-B filtering.

The H.264/AVC deblocking filter is a two-dimensional filter (i.e., edge filtering along both vertical and horizontal directions). To fully characterize its propagation behavior, mutual influences of two directions' filtering between adjacent MBs must be considered. Let's examine the vertical edge filtering first. From Table 1, when the strongest filter is applied to MB boundaries, there are at most six pixels (i.e. the pixels p_2, p_1, p_0, q_0, q_1 and q_2) will be influenced. According to Table 1, the evaluated pixels which served to adjust the left/upward most three pixels (p_1, p_0, and q_0) in the subsequent N-MB-B filtering may be modified by the antecedent MB-B filtering. As a result, the values of these three pixels certainly hinge upon the results of its antecedent MB-B filtering. However, the value of the right/downward most pixel q_1 in the subsequent N-MB-B filtering has nothing to do with the result of the antecedent MB-B filtering. The reason is that the subsequent N-MB-B filtering determines the value of the filtered pixel q_1 merely by evaluating the antecedent unmodified pixels, as shown in Fig. 6. More specifically, pixel g behaves not only as the q_2 pixel of the MB-B filter W, but also as the p_1 pixel of the N-MB-B filter X. One can see that, pixel j, which behave as the q_1 pixel of X, is determined by pixels h, i, j, and k which behave as q_2, q_1, q_0 and p_0 pixels of X, respectively. In other words, these four pixels are outside the filter length of the MB-B filter W.

To sum up, considering the pixel j, the evaluated pixels within the stage of N-MB-B filter X will not be influenced by the MB-B filter W, even if the maximal impact of MB-B filter W is taken into account. Going one step further, the succeeding filter is the vertical N-MB-B filter Y. Since the N-MB-B filter evaluates only three pixels on either side of an edge, the farthest referenced pixel is the aforecited pixel j, we called it the *strongest-filter-independent* pixel. That is, the strongest MB-boundary filtering influences only limited pixels to subsequent filtering stages. The pixels influenced by the vertical MB-B filtering are shown in Fig. 7, which can be

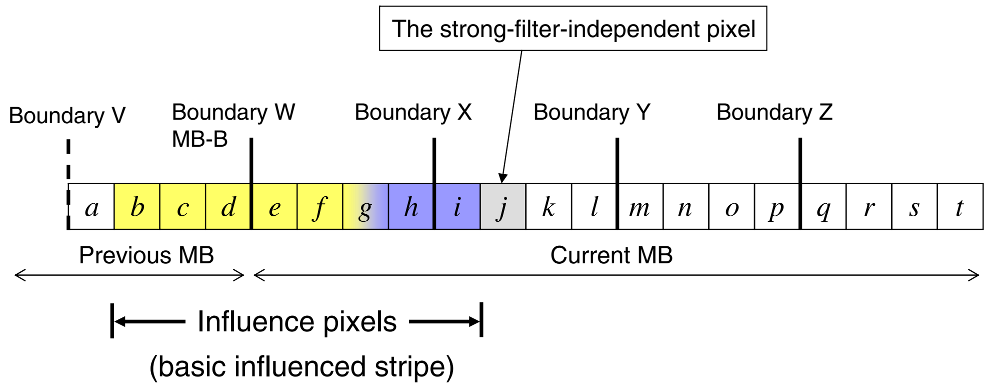

The strong-filter-independent pixel
Boundary V
Boundary W
MB-B
Boundary X
Boundary Y
Boundary Z
a b c d e f g h i j k l m n o p q r s t
Previous MB
Current MB
Influence pixels
(basic influenced stripe)

Figure 7 The maximal influence range of an MB-boundary filtering. Eight pixels are influenced by the filter W. The pixel g will be altered two times because of the filtering of boundaries W and X. The pixels after pixel j are independent of the MB-boundary filtering.

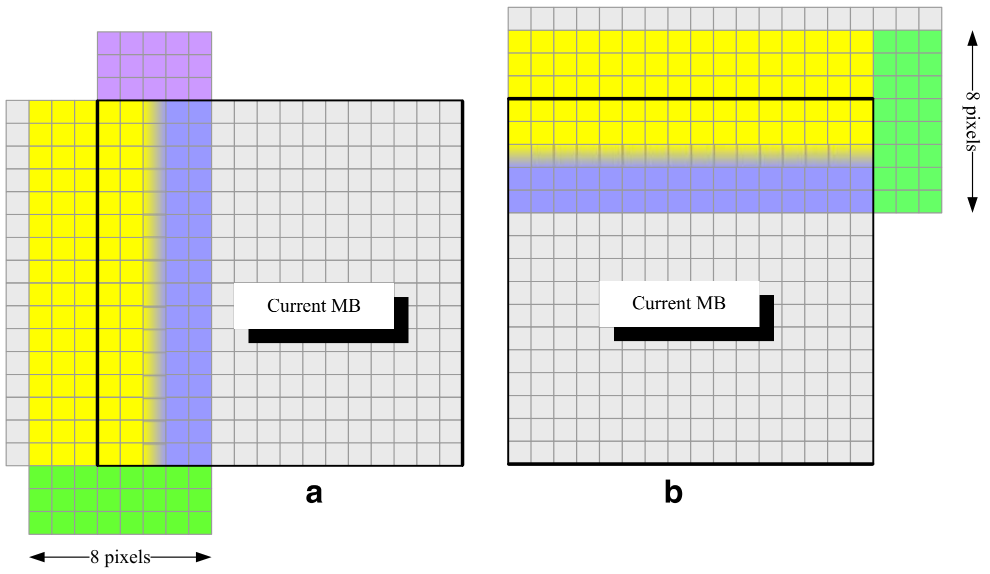


Figure 8 The maximal influence range of **a** a vertical MB-boundary filtering and **b** a horizontal MB-boundary filtering.

treated as a basic influenced stripe (represented by the set of colored pixels).

As discussed above, the influenced region of vertical MB-B filtering can now represented by the yellow area of Fig. 8a. The blue area indicates the additional filtered pixels of the next N-MB-B filtering (i.e., the edge filter *B* in Fig. 3) which referring to the pixels in the yellow area. Considering the influenced stripe described in Fig. 7, the influenced region caused by the antecedent (i.e., the left-hand-side) vertical MB-B filter is limited, as shown in the yellow and the blue areas of Fig. 8a. Notice that we have only considered the vertical filters up to now. Another filtering step that must be taken into account is the top-left horizontal MB-B filtering of the current MB. It follows that the purple pixels in Fig. 8a will be influenced. Similarly the horizontal MB-B filtering of the bottom-left boundary of the current MB needs to evaluate the green pixels shown in Fig. 8a. Again, the influence will also propagate to only a limited range, an eight-pixel wide stripe extending along the vertical direction. Similarly, considering the impact of the horizontal MB-B filtering, the maximal influenced region of horizontal MB-B filtering will be an eight-pixel wide stripe extending along the horizontal direction, as shown in Fig. 8b.

As for the influence of N-MB-B filtering upon the vertical boundaries X, Y and Z, we show the maximal influenced pixels by the N-MB-B filtering in Fig. 9.

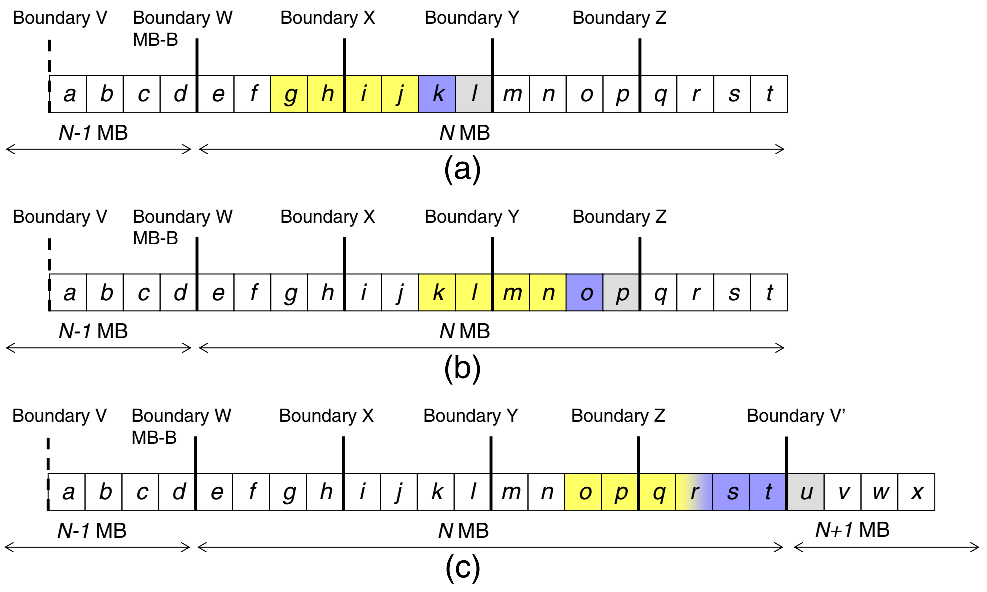


Figure 9 The influenced ranges of the N-MB-boundary filtering. Each *gray pixel* is independent of the antecedent N-MB-boundary filtering. Five pixels are influenced by filtering **a** the boundary X and **b** the boundary Y; **c** six pixels are influenced by filtering the boundary Z. Notice that pixel *r* will be altered twice because of the filtering of boundaries Z and V'.

Correspondingly, the overall impact of both the vertical and the horizontal N-MB-B filters is shown in Fig. 10.

Consequently, the above observation implies that any affection caused by the filtering stages propagates to only a limited range, we named this fact as the "*limited propagation effect*" In other words, the filtering stages of left/upward and right/underneath regions, which are separated by the colored regions are self-governing, and therefore, can be filtered concurrently. Notice that the filtering stages within the colored regions should obey the processing order described in Section 3.1. The data dependency analyzed in this section builds the foundation of our parallel deblocking algorithm described in Section 5.

5 Parallel Deblocking Algorithm

5.1 Proposed Algorithm

Based on the limited propagation effect discussed in the previous section, we classify the data regions involved in the filtering stages into two categories: the regions can be filtered concurrently without incurring any error (say R_1) and the regions R_2 which depend on the results of R_1. In the proposed parallel deblocking algorithm, along with the block boundary, we divide the $w \times h$ region into N rectangular units, u_i, which belong to R_1. That is,

$$R_1 = \bigcup_{i=0}^{N-1} u_i. \tag{1}$$

Notice that the $w \times h$ region can be considered as a partial region of a frame. For easy presentation, we take $w \times h$ region as an entire frame hereafter.

Since as indicated in the previous section, MB-B and N-MB-B filters of the filtering stages influence only limited pixels, any error caused by the preceding filtering will not propagate over the basic influenced stripe. By definition, u_i are outside the impacted ranges, so we can issue u_i all together. Notice that the filtering stages within each u_i should follow the processing order specified in Section 3.1. To effectively utilize the computation power of multi-core architecture, the number of parallel filtering units, N, is determined by the number of available processing units. Furthermore, to better utilize the memory locality, the direction of segmentation, horizontal or vertical, is decided on the basis of the memory architecture. For row-major machines, horizontal segmentation is preferred; otherwise, vertical segmentation is adopted (c.f. Fig. 11).

Assume that the region is stored in a one-dimensional array buffer, thus, the specific position of u_i in the buffer is represented as

$$[S_i, E_i). \tag{2}$$

In Eq. 2, S_i can be formulated as

$$S_i = \begin{cases} 0, & \text{if } i = 0 \\ \left(\begin{array}{r} \left\lfloor \frac{i \cdot \alpha}{4 \cdot N} \right\rfloor \times 4 + 5\delta[\sigma] + 3\delta[\sigma - 1] \\ +3\delta[\sigma - 2] \\ +4\delta[\sigma - 3] \end{array} \right) \times \beta, & \text{if } i > 0 \end{cases} \tag{3}$$

where $\delta[n]$ denotes the delta function and

$$\sigma = \left(\left\lfloor \frac{i \cdot \alpha}{4 \cdot N} \right\rfloor \right) \bmod 4. \tag{4}$$

E_i can be formulated as

$$E_i = \begin{cases} \left\lfloor \frac{(i+1) \cdot \alpha}{4 \cdot N} \right\rfloor \times 4 \times \beta, & \text{if } 0 \le i < N - 1 \\ \alpha \times \beta, & \text{if } i = N - 1 \end{cases} \tag{5}$$

The (α, β) is determined by the row-major or column-major memory architecture and whose values are (h, w) or (w, h), respectively. The formula Eq. 3 is composed of the start address of 4×4 block boundary and the offset, which are $\lfloor \frac{i \cdot \alpha}{4 \cdot N} \rfloor \times 4$ and $5\delta[\sigma] + 3\delta[\sigma - 1] + 3\delta[\sigma - 2] + 4\delta[\sigma - 3]$, respectively. Since the influenced region is dependent on the MB-B and N-MB-B filters, the offset is proportional to the influence range of the filter. Therefore, coefficients of 5, 3, 3 and 4, are the pixels in vertical/horizontal direction of the influence range of the MB-B filter (c.f. Fig. 8), cases (a)(b) in Fig. 10, cases (c)(d) in Fig. 10 and cases (e)(f) in Fig. 10, respectively. More specifically, the $\lfloor \frac{i \cdot \alpha}{4 \cdot N} \rfloor \times 4$ represents the start address of the block boundary (either the MB boundary or the red line in Fig. 10) and the coefficients of 5, 3, 3 and 4 are the vertical/horzional pixels, which indicate the remaining influence area started after the block boundary (i.e., $\lfloor \frac{i \cdot \alpha}{4 \cdot N} \rfloor \times 4$).

The remaining filtering stages are conducted over the infected areas, v_i, of u_i and are denoted as R_2. That is,

$$R_2 = \bigcup_{i=0}^{N-2} v_i. \tag{6}$$

The filtering stages of v_i are processed only when those of the u_i are completed because v_i hinges on the results of u_i. Since we segment a region along the block boundaries, the impact of each MB-B or N-MB-B filtering is within a fixed wide stripe, as shown in the colored regions of Figs. 8 and 10. Each v_i is, thus, the collection

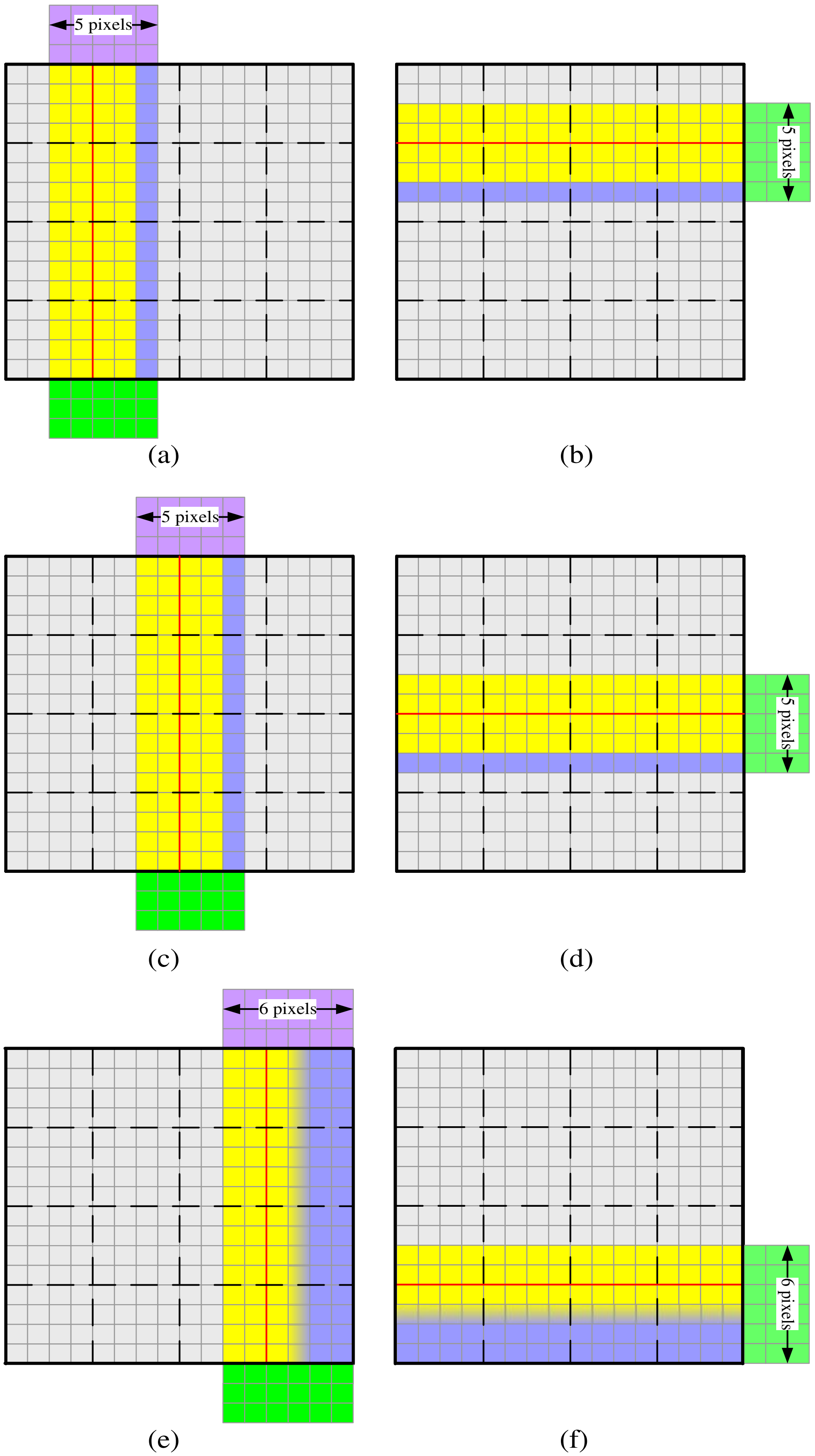


Figure 10 The overall impact, as shown by the colored regions, of N-MB-B filters. The *red lines* indicate the upon filtered boundaries: **a b** boundary X, **c d** boundary Y and **e f** boundary Z.

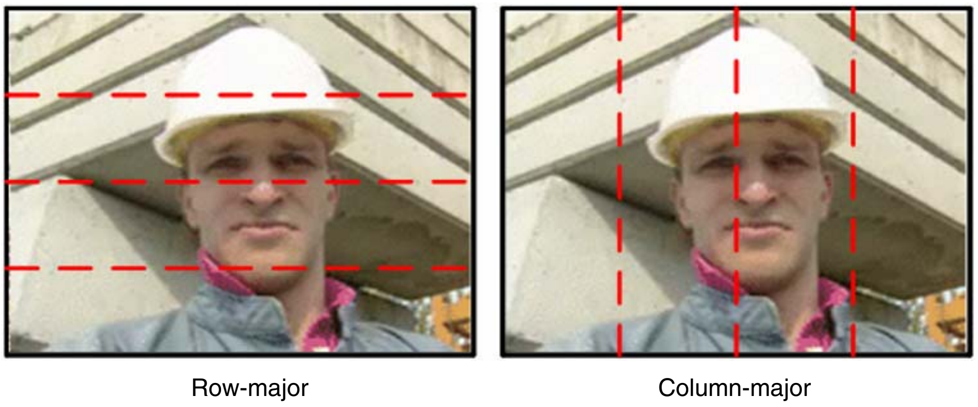


Figure 11 We segment one frame into N rectangles based on the memory architecture ($N = 4$ for quad-core architecture here).

of the fixed wide stripes spanning across the frame, and the specific position of which can be represented as

$$\left[E_i - A_i, S_{i+1}\right), \tag{7}$$

where

$$A_i = (3\delta[\sigma] + 2\delta[\sigma - 1] + 2\delta[\sigma - 2] + 2\delta[\sigma - 3]) \times \beta. \tag{8}$$

The proposed parallelism schedule is shown in Fig. 12, which dispatches the data-dependent regions, u_i and v_i, to the processor P_i. Figure 13 illustrates an example where the number of available processing units is assumed to be two, P_0 and P_1. In Fig. 13, u_0 and u_1 are dispatched to P_0 and P_1 and are processed concurrently. Once u_0 is finished, P_0 processes v_0 immediately. The design goal of our parallel algorithm is to minimize the amount of required synchronization and communication overhead while maximizing parallelization degree.

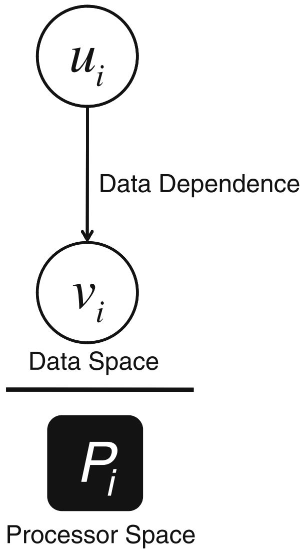


Figure 12 A schedule for dispatching the data-dependence u_i and v_i to the processor P_i The *directed line* indicates that the data in region v_i depends on the data in region u_i.

The proposed parallel deblocking filter algorithm has the following three advantages over the MB-wavefront:

1. It could utilize all the available cores: in the MB-wavefront method, the number of concurrent tasks depends on the number of MBs in the current wavefront iteration. In our scheme, as mentioned earlier, a frame is partitioned into n regions, where n is the number of available cores.
2. It incurs less synchronization overheads: Since there are no dependencies among regions allocated to different processors, a processor needs only to perform synchronization once in the entire frame. Therefore, the proposed parallel deblocking filter needs only d synchronization operations, where d is the number of processors.
3. It achieves better cache utilization: as explained in Section 3.2, the MB-waveafront method incurs frequent data exchange between processors. This results in poor cache utilization. While in our scheme, since there are no dependencies between MBs allocated to different processor, it does not require data exchange between processors.

5.2 Discussion

In this section, we first investigate how to combine the conventional parallelization technique with the proposed method and then discuss the load-balancing of the proposed parallel deblocking filtering.

The wavefront technique (i.e., the MB-wavefront approach), described in Section 3.2, takes an MB as the basic data partition unit. Nevertheless, as shown in [28], the data partition unit in the wavefront technique could be multiple MBs. It implies that a coarser-granularity grouping approach has the advantage of less synchronization overhead and better cache utilization as compared with the MB-wavefront approach. Suppose a

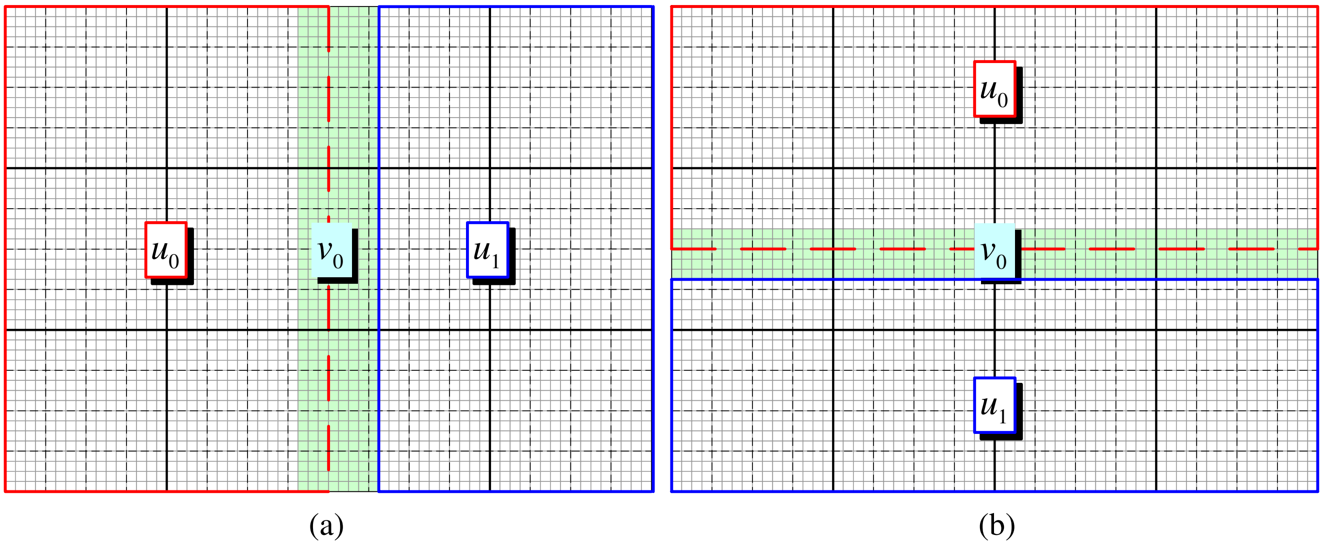


Figure 13 The specific positions of R_1 and R_2 in the memory buffer, where $N = 2$ and $(w, h) = (64, 48)$. Assume that the image is stored in a one-dimensional array, thus, the values a and b in $[a, b)$ are used to specify the physical location in the memory. By convention, the top-left pixel of the frame begins from *zero*. R_1 includes u_0 and u_1, which are bounded by the *red* box and the *blue* box, respectively. R_2 only includes v_0, the green colored region. **a** For the column-major memory architecture, u_0, u_1 and v_0 are located at $[0, 1536)$, $[1776, 3072)$ and $[1392, 1776)$, respectively. **b** For the row-major memory architecture, u_0, u_1 and v_0 are located at $[0, 1536)$, $[1728, 3072)$ and $[1408, 1728)$, respectively.

frame contains $M_h \times M_v$ MBs. We partition a frame into $\frac{M_h}{x} \times M_v$ *wavefront block units* (WBU). WBUs are processed in parallel along the 153° diagonal like that of the MB-wavefront. The MBs within a WBU are processed in sequential order. Once several WBUs have gone through the decoding stages preceding deblocking filter (i.e., entropy decoding, de-quantization, inverse transformation, motion compensation, and intra prediction), the proposed deblocking algorithm is then invoked. We use Fig. 14 to illustrate this process. We group 5 MBs as a WBU and process WBUs in parallel like the MB-wavefront. As long as the 1_st, the 3_{rd}, 5_{th}, 7_{th}, 9_{th} and 11_{th} WBU have gone through the decoding steps before deblocking filter, and they are currently in the deblocking filter stage which adopts the proposed parallel algorithm. It is clear that the data partition granularity (i.e., the size of WBU) and the number of WBUs are critical parameters. However, how to decide these parameters is out of the scope of this paper. Another issue is the load-balancing problem. The tasks with light workload have to wait for the completeness of the threads with heavy workload, and in consequence the multi-core system could not be fully utilized. To enable the efficient use of multi-core system, the load balancing is one of the important issues for improving performance. Conventionally, to solve this performance issue is to observe the load balancing problem and then dynamically schedule the threads to process tasks. The proposed parallel deblocking filtering divide the deblocking process into several tasks executed on different processors. However, to observe the load balancing problem and further dynamically schedule the filtering is out of the intend of this paper.

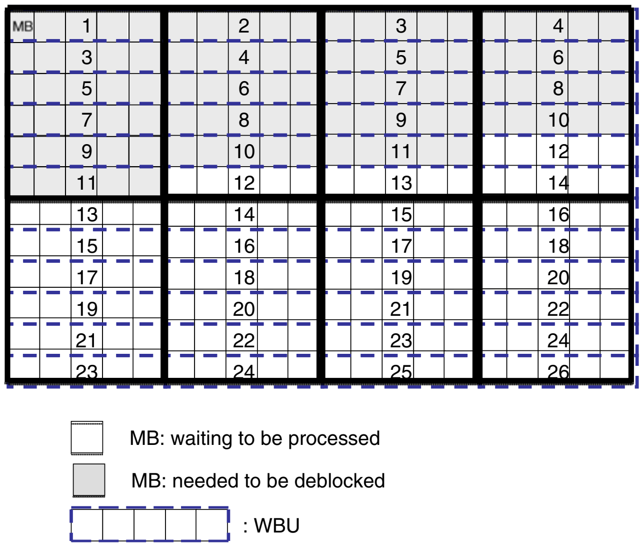


Figure 14 An example of WBU with 5 MBs. Each rectangle represents an MB and the number of WBUs indicates the execution order. The rectangular region surrounded by *bold line* is adopted by the proposed deblocking algorithm.

6 Experimental Results

In this section, we evaluate the effectiveness of the proposed parallel deblocking filter algorithm. We compare our scheme with the MB-wavefront algorithm described in Section 2. We implemented both approaches on an SIMD-optimized H.264/AVC decoder. The SIMD-optimized decoder used in this study is based on our previous work [12], which achieves 45.94 frame per second (fps) decoding rate for 720p standard test sequences. The code segment of the optimized decoder is less redundant compared with that of the reference software (http://iphome.hhi.de/suehring/tml/download/old_jm/jm73.zip) and bit-by-bit correctness of the decoded bitstreams has been verified; therefore, the experimental results are believed to be able to reflect the effectiveness of deblocking filter in practice.

All parallel programs are realized using the open POSIX threads library (http://sourceware.org/pthreads-win32/) on the Intel Core 2 CPU with clock rate 2.13 GHz, and Intel Core 2 Quad CPU with clock rate 2.66 GHz. We implemented the MB-wavefront for deblocking filter in Algorithm 1, as mentioned in Section 3.2. The testbed contains 10 streams with three resolution levels, 1280 × 720(720p), 720 × 480(480p) and 352 × 288(CIF), which are encoded in H.264/AVC baseline profile by using the reference software JM73 (http://iphome.hhi.de/suehring/tml/download/old_jm/jm73.zip) The frame reconstruction includes the stages of entropy decoding, de-quantization, inverse integer transform, intra prediction and motion compensation.

Below we first analyze the speedup on deblocking filter obtained from proposed parallel deblocking

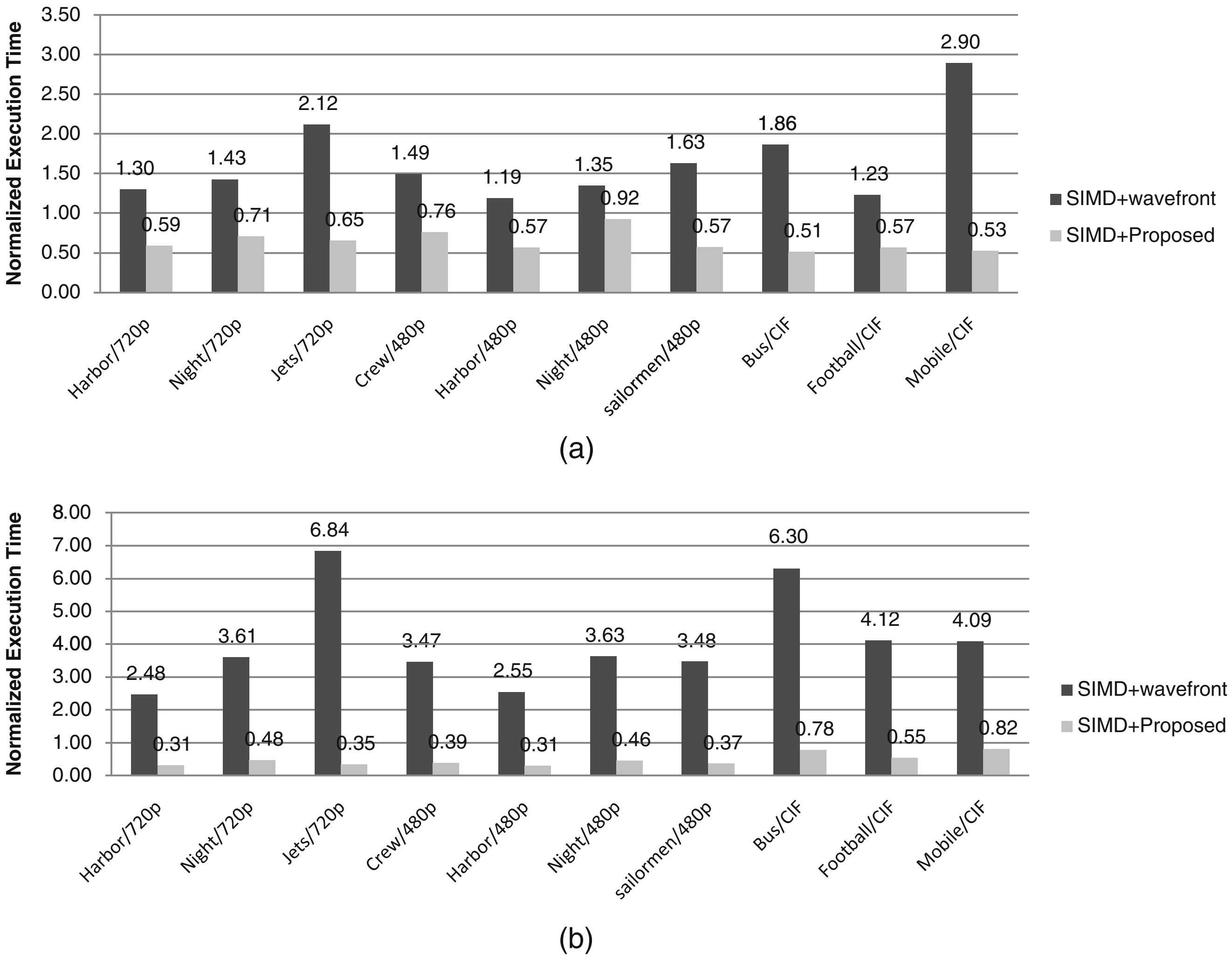


Figure 15 The performance comparison of the MB-wavefront and the proposed approach on **a** two-core and **b** four-core processers.

Table 2 The average normalized execution time of the MB-wavefront and the proposed approach, for the task of deblocking filtering.

	Two-core		Four-core	
	Wavefront	Proposed	Wavefront	Proposed
720p	1.62	0.65	4.31	0.38
480p	1.42	0.71	3.28	0.38
CIF	2	0.54	4.84	0.72

algorithm and the MB-wavefront approach, respectively[1]. We then show the overall decoding performance improvement achieved by our scheme. Finally, we perform synchronization overhead analysis.

6.1 Speedup Analysis

Figure 15 compares the performance of the MB-wavefront and the proposed approach. The y-axis shows the execution time of deblocking filter normalized to the SIMD optimized decoder. The average normalized execution time is summarized in Table 2. Notice that the performance of CIF sequences degrades when the number of cores increases. The reason is that low resolution of frames hardly allows large amount of parallel blocks in the proposed algorithm, so the performance is easily affected by the synchronization overhead. The results show that the proposed approach provides significant performance gain, up to 95% and 224.07% speedup for two-core and four-core processors, respectively. In contrast, applying the MB-wavefront to the SIMD-optimized code actually degrades performance.

The effective speedup gains of the proposed algorithm on the entire decoding process, for 2-core and 4-core platforms, are shown in Tables 3, and 4, respectively. The SIMD (ms) columns show the execution time of the SIMD-optimized decoder executed on the p-core platform, the p-core (ms) columns in the Tables show the execution time of our proposed algorithm executed on p-core platform, and *Speedup* columns in the Tables show the obtained speedup gains, where the speedup value is measured by

$$Speedup = \frac{\text{SIMD(ms)}}{p-\text{core(ms)}}.$$

[1]For this set of analysis, the input frames to deblocking filter has gone through the decoding stages preceding deblocking filter.

Table 3 Entire decoding speedup (2-core)

Bitstream	SIMD (ms)	2-core (ms)	Speedup
Harbour-720p	8250	7140	1.16
Jets-720p	2437	2202	1.11
Night-720p	6062	5498	1.10
Crew-480p	3121	2815	1.11
Harbour-480p	4034	3577	1.13
Night-480p	2468	2107	1.17
Sailormen-480p	2904	2592	1.12
bus-CIF	720	609	1.18
football-CIF	1139	965	1.18
mobile-CIF	1515	1253	1.21

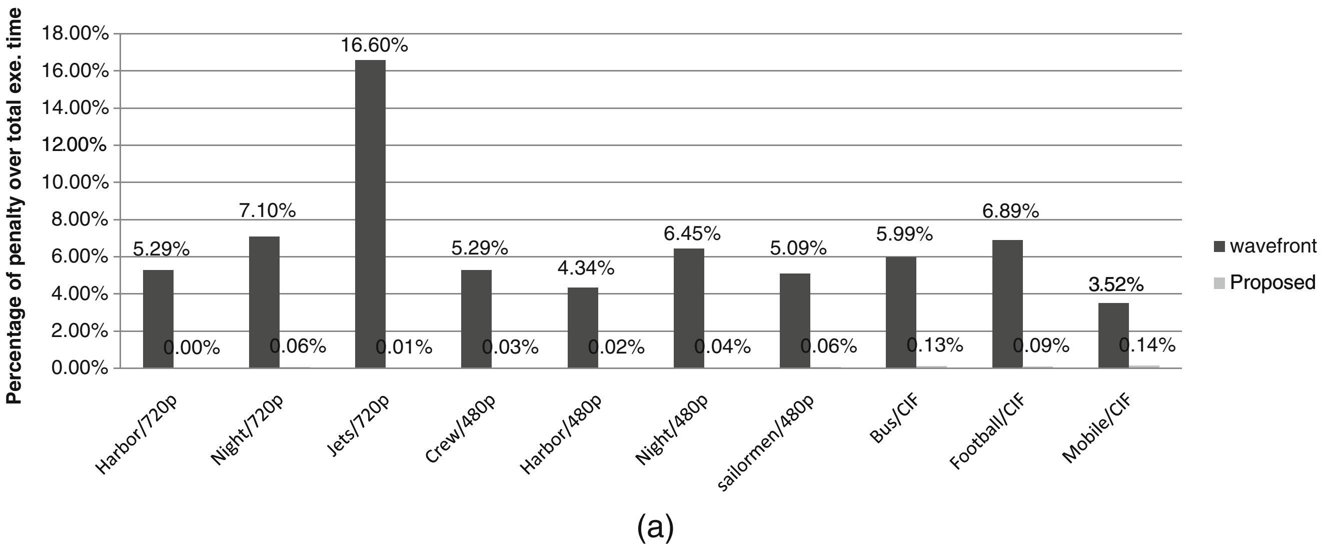
The results of Tables 3 and 4 show that the proposed method achieves up to 1.21x and 1.34x speedup gains on the overall encoding performance on the dual-core and the quad-core processors, respectively.

6.2 Synchronization Overhead Analysis

We analyze the synchronization overheads by measuring the lock/unlock operations of the POSIX thread library t(*pthreadVC2.dll*) using the *Intel VTune Performance Analyzer* (http://www3.intel.com/cd/software/products/asmo-na/eng/239144.htm). Figure 16 shows the contribution of lock/unlock operations to the total execution time of deblocking filter for the proposed parallel algorithm and the MB-wavefront approach. The results show that the synchronization overhead of the proposed algorithm is almost negligible (less than 1%) for both 2-core and 4-core processors. In contrast, the MB-wavefronting approach, as expected, shows much higher synchronization overheads, up to 16.60% and 30.78% for two-core and a four-core processors, respectively. These results indicate that our method provides better scalability than the MB-wavefront method.

Table 4 Entire decoding speedup (4-core)

Bitstream	SIMD (ms)	4-core (ms)	Speedup
Harbour-720p	6640	5295	1.25
Jets-720p	2047	1593	1.28
Night-720p	4826	3857	1.25
Crew-480p	2484	1859	1.34
Harbour-480p	3170	2625	1.21
Night-480p	1845	1531	1.21
Sailormen-480p	2295	1828	1.26
bus-CIF	483	389	1.24
football-CIF	859	764	1.12
mobile-CIF	1094	905	1.21



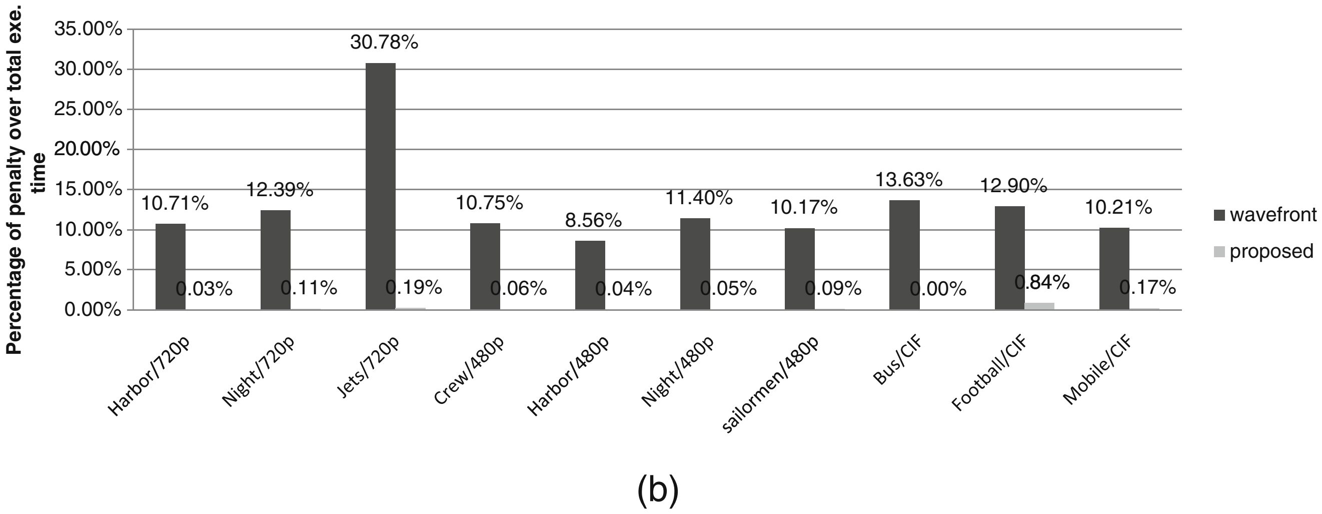


Figure 16 The comparison of parallelization penalty between the wavefronting and the proposed approach on three resolution levels, 720p, 480p and CIF, for **a** two-core and **b** four-core real machines.

7 Conclusion and Future Work

In this paper, we present a novel parallel algorithm for H.264/AVC deblocking filter based on an important observation named as "*limited propagation effect*". By exploiting the characteristics of limited influenced range, the proposed algorithm outperforms existing parallel approaches in terms of both the execution speed and the synchronization penalty.

Unlike the wavefront parallelization, there is no restriction on the number of concurrently executable processes at a time, thus, the proposed algorithm is believed to be adequate for parallelizing deblocking filtering when the number processing units is further increased in the future. On the other hand, we show that the performance might not be improved from the general wavefront parallelization even if the SIMD optimized decoder is taken into account. The main reason lies in the fact that the associated parallelization penalty impairs the benefit obtainable from fine-grain parallelization. Therefore, to effectively utilize the power of multi-core processors, how to reduce the parallelization overhead would play a rather important role in designing the parallel algorithm. One of our future work directions is to find the best tradeoff between the fine-grain parallelism and the synchronization/communication overheads by using the algorithmic-architectural co-design methodology.

References

1. List, P., Joch, A., Lainema, J., Bjntegaard, G., & Karczewicz, M. (2003). Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 614–619.
2. Lappalainen, V., Hallapuro, A., Hamalainen, T.D., Center, N.R., & Tampere, F. (2003). Complexity of optimized H. 26L video decoder implementation. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 717–725.
3. Chen, T.C., Fang, H.C., Lian, C.J., Tsai, C.H., Huang, Y.W., Chen, T.W., et al. (2003). Algorithm analysis and architecture design for HDTV applications-a look at the H. 264/AVC video compressor system. *IEEE Transactions on Circuits and Devices Magazine, 22*(3), 22–31.
4. Huang, Y.W., Chen, T.W., Hsieh, B.Y., Wang, T.C., Chang, T.H., & Chen, L.G. (2003). Architecture design for deblocking filter in H.264/JVT/AVC. *IEEE Proceedings of International Conference on Multimedia and Expo, 1*, 693–699.
5. Sima, M., Zhou, Y., & Zhang, W. (2004) An efficient architecture for adaptive deblocking filter of H.264/AVC video coding. *IEEE Transactions on Consumer Electronics, 50*(1), 292–296.
6. Chang, S.C., Peng, W.H., Wang, S.H., & Chiang, T. (2005). A platform based bus-interleaved architecture for de-blocking filter in H.264/MPEG-4 AVC. *IEEE Transactions on Consumer Electronics, 51*(1), 249–255.
7. Cheng, C.C., Chang, T.S., & Lee, K.B. (2006). An in-place architecture for the deblocking filter in H.264/AVC. *IEEE Transactions on Circuits and Systems: Analog and Digital Signal Processing, 99*, 530–534.
8. Khurana, G.K., & Mi, A.A.T.P.C. (2006). A pipelined hardware implementation of in-loop deblocking filter in H.264/AVC. *IEEE Transsactions on Consumer Electronics, 52*(2), 536–540.
9. Intel Corporation. IA-32 Intel architecture optimization reference manual. ftp://download.intel.com/design/Pentium4/manuals.
10. Warrington, S., Shojania, H., & Sudharsanan, S. (2006). Performance improvement of the H.264/AVC deblocking filter using SIMD instructions. *IEEE Proceedings of International Symposium on Circuits and Systems*.
11. Zhou, X., Li, E.Q., & Chen, Y.K. (2003). Implementation of H. 264 decoder on general-purpose processors with media instructions. *SPIE Conference on Image and Video Communications and Processing*.
12. Wang, S.W., Yang, Y.T., Li, C.Y., Tung, Y.S., & Wu, J.L. (2004). The optimization of H.264/AVC baseline decoder on low-cost TriMedia DSP processor. *Proceedings of SPIE, 5558*.
13. Lin, W., Goh, K.H., Tye, B.J., Powell, G.A., Ohya, T., & Adachi, S. (1997). Real time H. 263 video codec using parallel DSP. *Proceedings IEEE International Conference on Image Processing*, 586–589.
14. Chiu, C.N., Tseng, C.T., & Tsai, C.J. (1997). Tightly-coupled MPEG-4 video encoder framework on asymmetric dual-core platforms. *IEEE International Symposium on Circuits and Systems*, 586–589.
15. Dutta, S., Singh, D., Mehra, V., Semicond, P., & Sunnyvale, C.A. (1999). Architecture and implementation of a single-chip programmabledigital television and media processor. *IEEE Workshop on Signal Processing Systems*, 321–330.
16. Wyland, D.C. (2000). Media processors using a new microsystem architecture designed for the Internet era. In *Media Processors 2000, Proceedings of the SPIE* (vol. 3970, pp. 2–15). San Jose, CA, USA.
17. Sudharsanan, S., Sriram, P., Frederickson, H., & Gulati, A. (2000). Image and video processing using MAJC 5200. In *Proceedings of the 2000 IEEE International Conference on Image Processing*. Vancouver, Canada.
18. de With, P.H.N., & Jaspers, E.G.T. (1999). A video display processing platform for future TV concepts. *IEEE Transactions on Consumer Electronics, 45*, 1230–1240.
19. van der Tol, E.B., & Jaspers, E.G.T. (2002). Mapping of MPEG-4 decoding on a flexible architecture platform. In *Media Processors 2002, Proceedings of the SPIE* (pp. 1–13). San Jose, CA, USA.
20. Wang, S.H., Peng, W.H., He, Y., Lin, G.Y., Lin, C.Y., Chang, S.C., et al. (2005). A software-hardware co-implementation of MPEG-4 advanced video coding (AVC) decoder with block level pipelining. *The Journal of VLSI Signal Processing, 41*,(1), 93–110.
21. Rodriguez, A., Gonzalez, A., & Malumbres, M.P. (2006). Hierarchical parallelization of an H.264/AVC video encoder. *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), 00*, 363–368.
22. Chen, Y.K., Tian, X., Ge, S., & Girkar, M. (2004). Towards efficient multi-level threading of H. 264 encoder on Intel hyper-threading architectures. *Proceedings of 18th International Parallel and Distributed Processing Symposium*.
23. Chen, Y.K., Li, E.Q., Zhou, X., & Ge, S. (2005). Implementation of H.264 encoder and decoder on personal computers. *Journal of Visual Communications and Image Representations, 17*, 509–532.
24. Aho, A.V., Sethi, R., & Ullman, J.D. (2007). *Compilers: principles, techniques, and tools*. Boston: Addison-Wesley Longman.
25. Schoffmann, K., Fauster, M., Lampl, O., & Boszormenyi, L. (2007). An evaluation of parallelization concepts for baseline-profile compliant H. 264/AVC decoders. *Lecture Notes in Computer Science, 4641*.
26. Zhao, Z., & Liang, P. (2006). Data partition for wavefront parallelization of H.264 video encoder. *IEEE Proceedings of International Symposium on Circuits and Systems*.
27. JVT (2003). Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264| ISO/IEC 14496-10 AVC). Joint video team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050.
28. van der Tol, E.B., Jaspers, E.G., & Gelderblom, R.H. (2003). Mapping of H. 264 decoding on a multiprocessor architecture. *Proceedings SPIE Conference on Image and Video Communications and Processing*.
29. Nichols, B., & Buttlar, D. (1996). *Pthreads programming*. Sebastopol: O'Reilly.

Sung-Wen Wang received his Ph.D. degree in computer science from National Taiwan University, Taipei, Taiwan, in 2008. His general research interests are in the field of digital video coding, codec-processor architecture co-design and multimedia systems optimization, especially in video coding technology optimization.

Shu-Sian Yang received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Taiwan, in 2005 and 2007, respectively. His current research interests include video compression, image processing, and multimedia application. He is currently working at PixArt Imaging Inc., HsinChu, Taiwan as a senior engineer.

Hong-Ming Chen received the B.S. degree in computer science and information engineering from National Taiwan University, Taiwan, in 2007. He is currently pursuing the M.S. degree in the same department in National Taiwan University. His current research interests include video compression, image processing, digital content analysis, and multimedia application.

Chia-Lin Yang received the B.S. degree from the National Taiwan Normal University, Taiwan, R.O.C., in 1989, the M.S. degree from the University of Texas at Austin in 1992, and the Ph.D. degree from the Department of Computer Science, Duke University, Durham, NC, in 2001. In 1993, she joined VLSI Technology Inc. (now Philips Semiconductors) as a Software Engineer. She is currently an Associate Professor in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. Her research interests include energy-efficient microarchitectures, memory hierarchy design, and multimedia workload characterization. Dr. Yang is the recipient of a 2000-2001 Intel Foundation Graduate Fellowship Award and 2005 IBM Faculty Award.

Ja-Ling Wu (SM '98, Fellow '08) received his Ph.D. degree in electrical engineering from Tatung Institute of Technology, Taipei, Taiwan, in 1986.

From 1986 to 1987, he was an Associate Professor of the Electrical Engineering Department, Tatung Institute of Technology. Since 1987, he transferred to the Department of Computer Science and Information Engineering(CSIE), National Taiwan University(NTU), Taipei, where he is presently a Professor. From 1996 to 1998, he was assigned to be the first Head of the CSIE Department, National Chi Nan University, Puli, Taiwan. During his sabbatical leave (from 1998 to 1999), Prof. Wu was invited to be the Chief Technology Officer of the Cyberlink Corp. In this one year term, he involved with the developments of some well-known audio-video softwares, such as the PowerDVD. Since Aug. 2004, Prof. Wu has been appointed to head the Graduate Institute of Networking and Multimedia, NTU. Prof. Wu has published more than 200 technique and conference papers. His research interests include digital signal processing, image and video compression, digital content analysis, multimedia systems, digital watermarking, and digital right management systems.

Prof. Wu was the recipient of the Outstanding Young Medal of the Republic of China in 1987 and the Outstanding Research Award three times of the National Science Council, Republic of China, in 1998, 2000 and 2004, respectively. In 2001, his paper "Hidden Digital Watermark in Images" (co-authored with Prof. Chiou-Ting Hsu), published in IEEE Transactions on Image Processing, was selected to be one of the winners of the "Honoring Excellence in Taiwanese Research Award", offered by ISI Thomson Scientific. Moreover, his paper "Tiling Slideshow" (co-authored with his students) won the Best Full Technical Paper Award in ACM Multimedia 2006. Professor Wu was selected to be one of the lifetime Distinguished Professors of NTU, November 2006.

Prof. Wu has been elected to be IEEE Fellow, since 1 January 2008, for his contributions to image and video analysis, coding, digital watermarking, and rights management.