VECTORIZED DEBLOCKING FILTER FOR HD H.264 DECODING ON CELL/B.E.

Huoding Li†, Rong Yan‡, Xing Liu†, Yu Yuan‡, Sheng Xu†

{lihuod, yanrong, xingliu, yuanyu, xusheng}@cn.ibm.com
† IBM STG Development Lab, 2F, Bldg.10, No.399, Ke Yuan Road, Zhang Jiang High Tech Park, Shanghai 200131, P.R.C.
‡ IBM China Research Lab, Diamond Building, Zhongguancun Software Park, Beijing 100193, P.R.C.

ABSTRACT

For high definition (HD) H.264 decoding, deblocking filter is one of the most time-consuming modules. This paper proposes several vectorization approaches to speed up it on a single synergistic processor element (SPE) of IBM Cell Broadband Engine (Cell/B.E.) processor, by which great performance improvements are achieved. The average deblocking speed is 42.4 frames per second (fps) for 141 1080p H.264 video streams. The steady-going performance is obtained for different streams of various contents and bitrates. With the vectorized deblocking filter, the new HD H.264 decoder is able to decode two 1080p H.264 video streams simultaneously on PlayStation® 3 (PS3) in real-time. The proposed approaches are practical on many other vector processor platforms, but not limited to Cell/B.E. processor.

Index Terms— HD H.264 decoder, deblocking filter, vectorization, IBM Cell/B.E. processor

1. INTRODUCTION

The H.264 standard [1] brings a new efficient video compression method to the multimedia industry. Deblocking filter is a new feature compared with the previous standards. It greatly enhances the subjective quality of the picture. Meanwhile, the complexity overhead is up to 6 % (access frequency) and 10 % (processing time) at the decoder side [2]. For HD video, it is the most time consuming module [3].

The IBM Cell/B.E. processor is a heterogeneous multi-core processor which consists of one 64-bit PowerPC Processor Element (PPE) and eight SPEs. The SPEs are optimized for compute-intensive workloads, based on the single-instruction multiple-data (SIMD) architecture. These SPEs account for much of the computational power of the Cell/B.E. processor [4]. At an operating frequency of 3.2 GHz, the eight SPEs in the first-generation Cell/B.E. chip can perform up to 204.8 GFLOPS.

We implement HD H.264 decoder on Cell/B.E. to take advantage of its computational power, and gains excellent performance of simultaneously decoding two 1080p H.264 streams in real-time on PS3 with only 6 SPEs are opened for use [5]. The work discussed in this paper is a part of the above decoder, and is critical to the end performance.

The most effective optimization method on Cell/B.E. is vectorization. However, it is very difficult to parallelize the deblocking filter operations because there are a lot of data-dependent branches in the boundary strength (BS) decision procedure. Besides, the data storage pattern stops parallelizing the vertical edge filtering procedure. Respectively, we propose a branch vectorization approach to eliminate those data-dependent branches for BS calculation and a fast vectorized macroblock (MB) rotation approach to get over the data storage pattern barrier.

Section 2 introduces the branch vectorization approach. Section 3 introduces the vectorization of the vertical & horizontal edge filtering kernels, wherein the fast vectorized MB rotation approach is described in details. Section 4 shows the experimental results. The conclusion is drawn in section 5.

2. BRANCH VECTORIZATION IN BOUNDARY STRENGTH DECISION PROCEDURE

A conditional filtering is applied to vertical or horizontal edge of a 4×4 block [1]. The first phase is to decide the BS. It is nearly impossible to parallelize the original boundary strength decision procedure due to the numbers of data-dependent branches. We notice that the following conditions together decide the final BS:

Table 1: Conditions for boundary strength decision.

C_1	The boundary is a slice boundary.
C_2	The boundary is between two intra coded MB.
C_3	The boundary is MB boundary.
C_4	The two sub-MBs have coefficiencies.
C_5	The two neighboring sub-MBs are in the same reference picture.
C_6	The two neighboring 4x4 blocks are in an 8x8 sub-MB.
C_7	The motion vector difference is greater than one pixel.

All the MBs are separated into two categories: the Intra MB and the Inter MB. For Intra MBs (C_2 is true), the four sub-MBs on the same edge share the same BS value. The BS calculation follows the rule: *if C_1, then BS=0; if C_3, then BS=4; else BS=3.* For Inter MBs, the conditional branches for BS calculation can be summarized as:

$$T_1 = !C_1 \,\&\, C_4 \,\&\, !C_6,$$
$$T_2 = !C_1 \,\&\, C_4 \,\&\, C_6,$$
$$T_3 = !C_1 \,\&\, !C_4 \,\&\, !C_5 \,\&\, !C_6,$$
$$T_4 = !C_1 \,\&\, !C_4 \,\&\, !C_5 \,\&\, C_6,$$
$$T_5 = !C_1 \,\&\, !C_4 \,\&\, C_5 \,\&\, !C_7 \,\&\, !C_6,$$
$$T_6 = !C_1 \,\&\, !C_4 \,\&\, C_5 \,\&\, !C_7 \,\&\, C_6,$$
$$T_7 = !C_1 \,\&\, !C_4 \,\&\, C_5 \,\&\, C_7,$$
$$T_8 = C_1, \quad where\ C_i \equiv 1, !C_i \equiv 0, \forall i \in \{1,2,3,4,5,6,7\}.$$

$!C_i$ denotes the condition when the *i-th* statement in Table 1 is false. And we construct the branch vector:

$$\vec{T} =< T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8 >, where\ T_i \in \{0,1\}, 1 \le i \le 8.$$

Obviously, $\exists\, j, T_j = 1 \Rightarrow \forall i \ne j, T_i = 0, 1 \le i, j \le 8,$ which means only one condition is true at a time. If we further construct the condition vectors as below:

$$\vec{C}_1 =< !C_1, !C_1, !C_1, !C_1, !C_1, !C_1, !C_1, C_1 >,$$
$$\vec{C}_4 =< C_4, C_4, !C_4, !C_4, !C_4, !C_4, !C_4, 1 >,$$
$$\vec{C}_5 =< 1, 1, !C_5, !C_5, C_5, C_5, C_5, 1 >,$$
$$\vec{C}_6 =< !C_6, C_6, !C_6, C_6, !C_6, C_6, 1, 1 >,$$
$$\vec{C}_7 =< 1, 1, 1, 1, !C_7, !C_7, C_7, 1 >,$$

the branch vector can be denoted as:

$$\vec{T} = \vec{C}_1 \,\&\, \vec{C}_4 \,\&\, \vec{C}_5 \,\&\, \vec{C}_6 \,\&\, \vec{C}_7 .$$

Only 0, 1, and 2 are valid BS values for Inter MB when the corresponding condition T_i is true, which is represented as the vector:

$$\vec{R} =< 2, 0, 1, 0, 1, 0, 0, 0 >.$$

The BS decision procedure for Inter MB now can be replaced by the vector operations through 3 steps:

1. Construct the condition vectors $\vec{C}_i$ according to the value of C_i ;
2. Calculate the value of $\vec{T}$ by carrying out five vector bit AND operations;
3. The final BS values of the inter coded MBs should be derived by

$$BS = R_i \ if\ T_i = 1, where\ R_i \in \vec{R}\ and\ T_i \in \vec{T}\ .$$

As the result, no matter what the conditions are, the BS decision procedure contains constant number of vector bit AND operations, and these vector operations are accelerated by the SPE, which leads to a much higher and more stable computing speed than the scalar design.

3. VECTORIZED FILTERING KERNELS

When BS values are decided, the filtering operation is first executed on the four vertical edges then on the four horizontal edges for each MB. Assume block *P* and block *Q* are neighboring 4x4 blocks, and denote the pixels on the *lth* row or column within these two blocks as p_{li} and q_{li} (before being filtered) respectively. The pixels standing on each side of the edge is in the below order:

$$q_{l3}\ q_{l2}\ q_{l1}\ q_{l0} \mid p_{l0}\ p_{l1}\ p_{l2}\ p_{l3}, \quad \forall i, l \in \{0,1,2,3\}\ 0 \le q_{li}, p_{li} \le 255\ .$$

We use the case of strong edge filtering (*BS=4*) for U/V component as an example to explain the vetorization process. In this case:

$$p'_{l0} = \frac{1}{4}\left(2p_{l1} + p_{l0} + q_{l1} + 2\right).$$

Correspondingly, q'_{l0} is calculated in the same way. Denote the pixel vectors as:

$$\vec{P}_i =< p_{1i}, p_{2i}, \ldots, p_{Ni} >, \vec{Q}_i =< q_{1i}, q_{2i}, \ldots, q_{Ni} >,$$

then define the vector addition and numerical multiplication as:

$$\vec{P}_i + \vec{Q}_i =< p_{1i} + q_{1i}, p_{2i} + q_{2i}, \ldots, p_{Ni} + q_{Ni} >,$$
$$k\vec{P}_i =< kp_{1i}, kp_{2i}, \ldots, kp_{Ni} >, k \in R,$$

the strong edge filtering polynomial for U/V component could be rewritten in the vector form as:

$$\vec{P}'_0 = \frac{1}{4}\left(2\vec{P}_1 + \vec{P}_0 + \vec{Q}_1 + \vec{2}\right),$$

which equals to

$$\begin{pmatrix} p'_{00} \\ p'_{10} \\ p'_{20} \\ p'_{30} \end{pmatrix} = \frac{1}{4}\left(2\begin{pmatrix} p_{01} \\ p_{11} \\ p_{21} \\ p_{31} \end{pmatrix} + \begin{pmatrix} p_{00} \\ p_{10} \\ p_{20} \\ p_{30} \end{pmatrix} + \begin{pmatrix} q_{01} \\ q_{11} \\ q_{21} \\ q_{31} \end{pmatrix} + \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \end{pmatrix}\right) = \begin{pmatrix} \frac{1}{4}(2p_{01} + p_{00} + q_{01} + 2) \\ \frac{1}{4}(2p_{11} + p_{10} + q_{11} + 2) \\ \frac{1}{4}(2p_{21} + p_{20} + q_{21} + 2) \\ \frac{1}{4}(2p_{31} + p_{30} + q_{31} + 2) \end{pmatrix}.$$

For horizontal edge filtering, the $\vec{P}_i$ and the $\vec{Q}_i$ are stored in rows and fit the vector order, so it is easy to perform the vetorization. For vertical edge filtering, obviously, rotating the MB in a two-dimension (2D) array before filtering is a feasible way for parallelizing the computations. The proposed vertical edge filtering process is illustrated in Figure 1, which reuses the horizontal edge filtering kernel with a pre-rotation (clockwise) and a post-rotation (counter-clockwise).

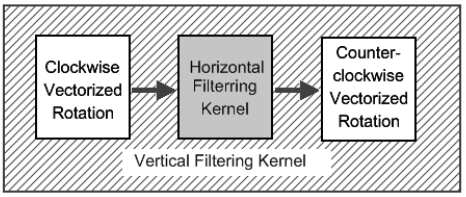

Figure 1: The proposed vertical edge filtering kernel.

Since there are two rotations performed for each MB, we propose a fast MB rotation approach to guarantee the end performance.

3.1. FAST VECTORIZED MB ROTATION

The 2D arrays, where the MBs are stored, should be rotated clockwise and then counter-clockwise.

For any $N \times N$ 2D array, where N is power of 2, if we use the common element exchanges, the complexity of the entire rotation is $O(N^2)$. However, by utilizing the vector instructions, we can introduce a fast vector rotation, whose complexity is $O(\log_2 N)$. With the vector instructions, 16 bytes of data can be manipulated at a time. The rotation steps can be represented as mappings.

Define a $N \times N$ matrix A, where $N=2^k$ and k is an positive integer.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},\ A_{ij}\ is\ \frac{N}{2} \times \frac{N}{2}\ submatrices.$$

Define mapping f as

$$f : W^{N\times N} \to W^{N\times N},\ where\ W = \{x \mid 0 \le x \le 255, x \notin Z\},$$

$$f(A) = f\left(\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\right) = \begin{pmatrix} A_{21} & A_{11} \\ A_{22} & A_{12} \end{pmatrix} = \tilde{A}.$$

Then the rotation consists of the following steps:

Step 1: $f(A)$;

Step 2: For each $2^{k-1} \times 2^{k-1}$ submatrices A_{ij}, do $f(A_{ij})$;

Step 3: Recursively execute step 1 and step 2 until the pixel level rotation is performed (k recursions).

Take a clockwise rotation of an 8x8 block as the example which is shown in Figure 2. First of all, the 8x8 block is divided into four 4x4 blocks at *Step A*. Then do the rotation mapping f to exchange the positions of the four 4x4 blocks clockwise at *Step B*. Recursively, divide each one of the four 4x4 blocks into four 2x2 blocks at *Step C*. And exchange the positions of four 2x2 blocks within each 4x4 block clockwise at *Step D*. The last step is to change the four pixels within each 2x2 block clockwise at *Step E*.

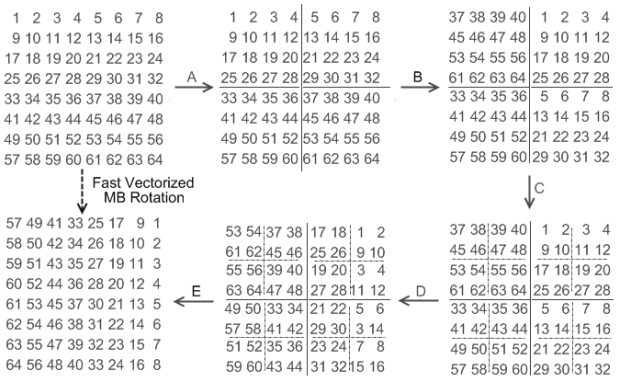

Figure 2: an example of fast vectorized MB rotation (clockwise).

Similarly, for the counter-clockwise rotation, only f^{-1} needs to be defined:

$$f^{-1} : W^{N\times N} \to W^{N\times N},\ where\ W = \{x \mid 0 \le x \le 255, x \notin Z\},$$

$$f^{-1}(A) = f^{-1}\left(\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\right) = \begin{pmatrix} A_{12} & A_{22} \\ A_{11} & A_{21} \end{pmatrix} = \tilde{A}.$$

The counter-clockwise rotation procedure is the same as the clockwise rotation except the rotation mapping f should be replaced with f^{-1}.

With the fast vectorized MB rotation approach, the horizontal edge filtering kernel is reused by the vertical edge filleting kernel, which not only takes the advantage of the vector acceleration power of SPE but also reduce the code size. Small code size is valuable for Cell/B.E. based application since the size of local store (LS) on SPE is limited to 256KB.

4. EXPERIMENTAL RESULTS

As the result of our implementation described above, the optimized deblocking filter is averagely 32 times faster than that of the simply optimized reference decoder JM11 [6]. The deblocking speed on single SPE is 42.4 fps averagely for 141 1080p H.264 video. We in the following tests use PS3 as the test platform. Further, denote deblocking filter of simply optimized reference decoder JM11 as JM_SO, where the code redundancy is removed from scalar code to make comparisons fairer. We note the deblocking filter of our decoder as Aeolus, where the proposed approaches are used.

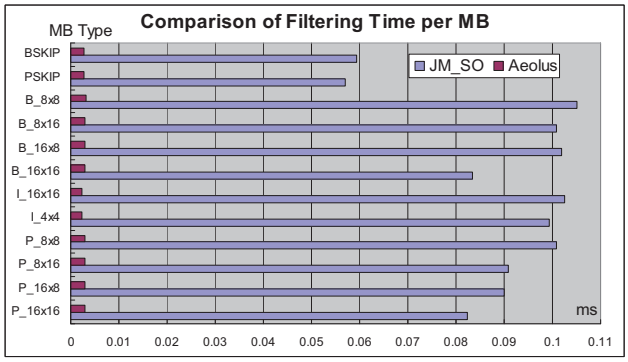

Figure 3: The comparison of filtering time per MB between JM_SO and Aeolus.

Figure 3 shows the comparison of filtering time per MB between JM_SO and Aeolus. The results are the averages for five 1080p bitstreams, the so-called 'filtering time' covers the whole deblocking filtering process. The five bitstreams are encoded by x264 encoder [7] from 5 different sequences, i.e. riverbed, rush_hour, sunflower, blue_sky and pedestrian_area. They are encoded at about 10Mbps with 2 references frames selection and IPBPB structure, each GOP includes 15 frames. The filtering time of Aeolus is less than 0.003ms/MB in average, compared

with that of JM_SO of averagely over 0.089ms/MB, there is about 32 times speedup. From Figure 3, we can see that for different types of MBs, the filtering time of JM_SO varies largely while that of Aeolus is steady-going. The reason is that the proposed branch vectorization approach eliminates the conditional branches, and almost all types of MB execute the same computation for BS decision.

The same conclusion is drawn from Figure 4 for results of pedestrian_area bistreams encoded at different bitrates. All the Aeolus curves are gathered at the bottom. They are very close to each other and nearly horizontal to the bottom line. Meanwhile, the JM_SO curves rise when the bitrate increases, especially for the P_16x16, B_16x16, P_16x8 and P_8x16.

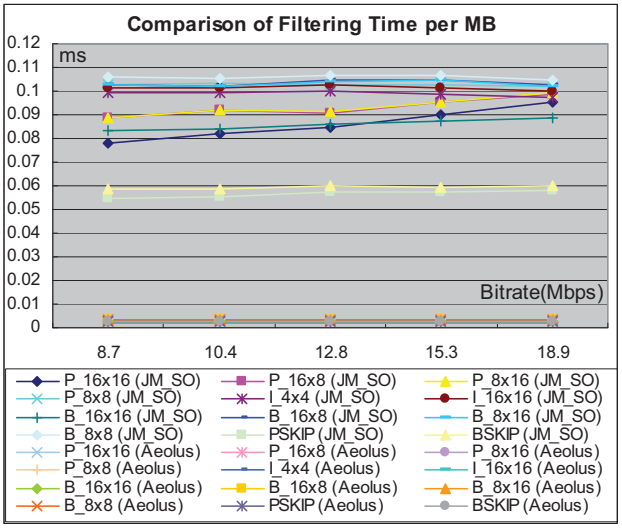

Figure 4: The comparison of filtering time per MB between JM_SO and Aeolus at different bit rates.

The stability makes this design valuable for a wide range of H.264 bitrates, so that it could be promoted to many different platforms and applications.

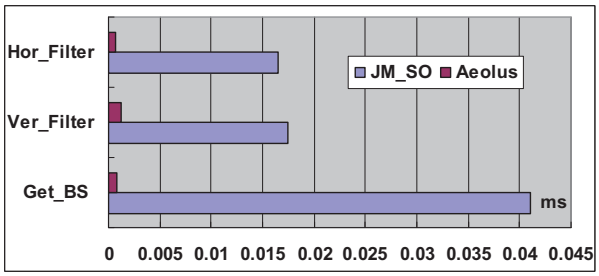

Figure 5: Time breakdown for deblocking filtering.

Figure 5 shows the time breakdown of the vectorization process where the performance progress is also clearly observed. The bitstreams used here are the same as that used in Figure 3. Get_BS denotes boundary strength decision procedure; Ver_Filter denotes vertical edge filtering procedure with fast vectorized MB rotation performed; and Hor_Filter edge denotes horizontal edge filtering procedure.

5. CONCLUSIONS

This paper introduces several vectorization approaches for the deblocking filter of HD H.264 video decoder.

In BS decision procedure, the data-dependent branches are replaced by the vector operations. The steady-going performance is obtained for varied contents and bit rates.

Filtering process for a set of samples across the horizontal and vertical block edges is parallelized by vector operations. A fast vectorized MB rotation approach of low complexity is proposed to ensure the final performance. It also reduces the code size since the computation kernel can be reused after MB rotation.

The vectorization approaches introduced in this paper are very practical methods for designing the deblocking filter. Its effectiveness and efficiency have been solidly proved on IBM Cell/B.E. processor but not limited to this specific platform. By using single SPE of Cell/B.E. processor, the deblocking speed gains averagely 42.4 fps for 141 1080p HD H.264 video streams. This number varies from 40 to 44 over the different streams. With the optimized deblocking filter, the overall decoder is able to decode two streams of 1080p H.264 video simultaneously in real time (over 33 fps) on PS3, where one Cell/B.E. processor is assembled and only 6 SPEs of it are opened for use.

6. REFERENCES

[1] ITU-T Recommendation H.264 & ISO/IEC 14496-10, "Advanced Video Coding for Generic Audiovisual Services", Version 4, 2005.

[2] S. Saponara, C. Blanch, K. Denolf, and J. Bormans, “The JVT Advanced Video Coding Standard: Complexity and Performance Analysis on a Tool-by-tool Basis,” *Packet Video Workshop (PV’03)*, Nantes, France, April 2003.

[3] M. Alvarez, E. Salamí, A. Ramirez, M. Valero, “A Performance Characterization of High Definition Digital Video Decoding using H.264/AVC,” *2005 IEEE International Symposium on Workload Characterization* Austin TX, USA, October 2005.

[4] T. Chen, R. Raghavan, J.N. Dale, and E. Iwata, ‘‘Cell Broadband Engine Architecture and its First Implementation—A Performance View,’’ *IBM J. Res. & Dev.*, vol. 51, no. 5, pp. 559–572, 2007.

[5] Y. Yuan, R. Yan, H.D. Li, X. Liu, and S. Xu, “High Definition H.264 Decoding on Cell Broadband Engine,” A*CM Multimedia '07, Demo session 2*, Augsburg, Germany, September 2007.

[6] Joint Video Team Reference Software, Version 11 (JM11), http://iphome.hhi.de/suehring/tml/download/.

[7] http://www.videolan.org/x264.html.