Project Proposal

Time Optimization of HEVC Encoder over X86 Processors using SIMD

Spring 2013

Multimedia Processing EE5359

Advisor: Dr. K. R. Rao

Department of Electrical Engineering

University of Texas, Arlington

Kushal Shah

1000857252

kushal.shah7@mavs.uta.edu

ACRONYMS AND ABBREVIATIONS

- HEVC: High Efficiency Video Coding
- JCT-VC: Joint Collaborative Team on Video Coding
- SIMD : Single Instruction Multiple Data
- ME: Motion Estimation
- SAD: Sum of Absolute Differences
- SATD: Sum of Absolute Transformed Differences (SATD)
- CTU: Coding Tree Unit
- CB: Coding Block
- PB: Prediction Unit
- TB: Transform Unit
- MC: Motion Vector
- SSE: Streaming SIMD Extensions
- MOS: Mean Opinion Score

Overview:

The High Efficiency Video Coding (HEVC) standard is the most recent joint video project of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) standardization organizations, working together in a partnership known as the Joint Collaborative Team on Video Coding (JCT-VC) [1]. The first edition of the HEVC standard has been finalized in January 2013, resulting in an aligned text that has been published by both ITU-T and ISO/IEC. Additional work is planned to extend the standard to support several additional application scenarios, including extended-range uses with enhanced precision and color format support, salable video coding, and 3-D/stereo/multi view video coding. In ISO/IEC, the HEVC standard will become MPEG-H Part 2 (ISO/IEC 23008-2) and in ITU-T it is likely to become ITU-T Recommendation H.265.

Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 [2] and H.263 [3], ISO/IEC produced MPEG-1 [4] and MPEG-4 Visual [5], and the two organizations jointly produced the H.262/MPEG-2 Video [6] and H.264/MPEG-4 Advanced Video Coding (AVC) [7] standards. The two standards that were jointly produced have had a particularly strong impact and have found their way into a wide variety of products that are increasingly prevalent in our daily lives. Throughout this evolution, continued efforts have been made to maximize compression capability and improve other characteristics such as data loss robustness, while considering the computational resources that were practical for use in products at the time of anticipated deployment of each standard. The major video coding standard directly preceding the HEVC project was H.264/MPEG-4 AVC, which was initially developed in the period between 1999 and 2003, and then was extended in several important ways from 2003–2009. H.264/MPEG-4 AVC has been an enabling technology for digital video in almost every area that was not previously covered by H.262/MPEG-2 Video and has substantially displaced the older standard within its existing application domains. It is widely used for many applications, including broadcast of high definition (HD) TV signals over satellite, cable, and terrestrial transmission systems, video content acquisition and editing systems, camcorders, security applications, Internet and mobile network video, Blu-ray Discs, and real-time conversational applications such as video chat, video conferencing, and telepresence systems. However, an increasing diversity of services, the growing popularity of HD video, and the emergence of beyond- HD formats (e.g., 4k×2k or 8k×4k resolution) are creating even stronger needs for coding efficiency superior to H.264/ MPEG-4 AVC's capabilities. The need is even stronger when higher resolution is accompanied by stereo or multiview capture and display. Moreover, the traffic caused by video applications targeting mobile devices and tablets PCs, as well as the transmission needs for video-on-demand services, are imposing severe challenges on today's networks. An increased desire for higher quality and resolutions is also arising in mobile applications.

Block Diagram:

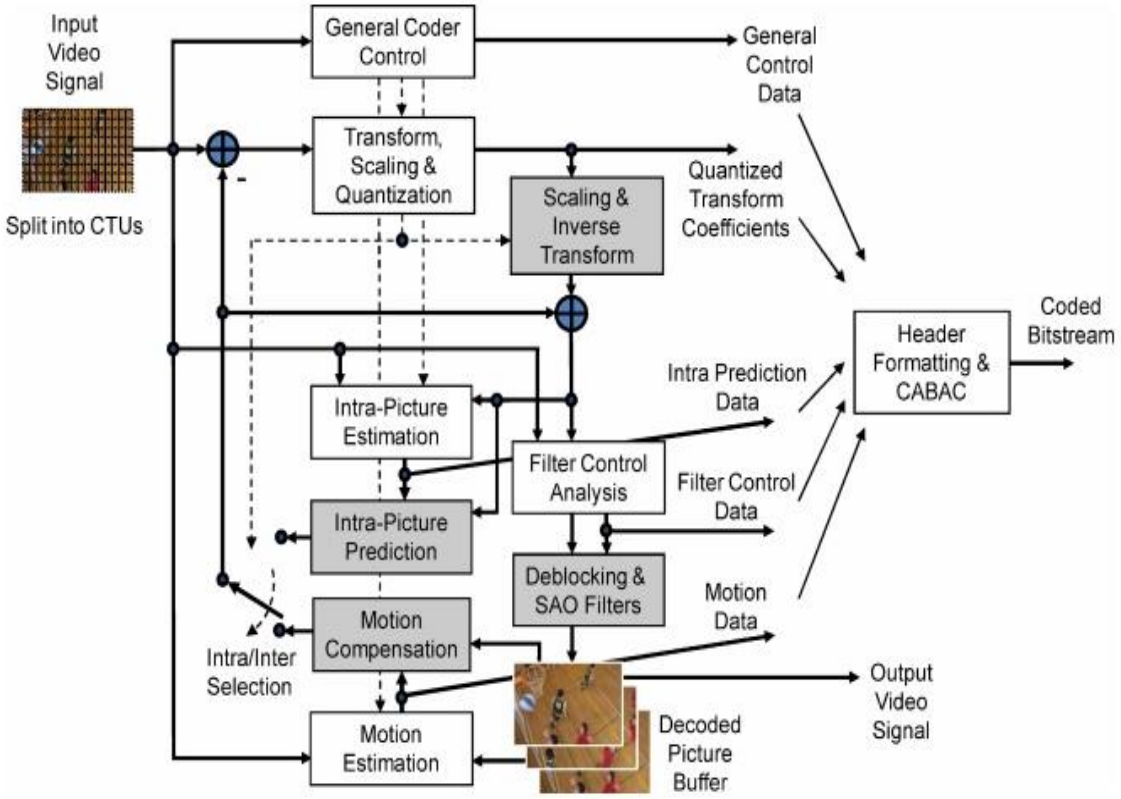

Figure 1: Block Diagram of HEVC Encoder [19]

In the following, the various features involved in hybrid video coding using HEVC are highlighted as follows:

1) **Coding tree units and coding tree block (CTB) structure:** The core of the coding layer in previous standards was the macroblock, containing a 16×16 block of luma samples and, in the usual case of 4:2:0 color sampling, two corresponding 8×8 blocks of chroma samples; whereas the analogous structure in HEVC is the coding tree unit (CTU), which has a size selected by the encoder and can be larger than a traditional macroblock. The CTU consists of a luma CTB and the corresponding chroma CTBs and syntax elements. The size L×L of a luma CTB can be chosen as L = 16, 32, or 64 samples, with the larger sizes typically enabling better compression. HEVC then supports a partitioning of the CTBs into smaller blocks using a tree structure and quadtree-like signaling [8].

2) **Coding units (CUs) and coding blocks (CBs):** The quadtree syntax of the CTU specifies the size and positions of its luma and chroma CBs. The root of the quadtree is associated with the CTU. Hence, the size of the luma CTB is the largest supported size for a luma CB. The splitting of a CTU into luma and chroma CBs is signaled jointly. One luma CB and ordinarily two chroma CBs, together with associated syntax, form a coding unit (CU). A CTB may contain only one CU or may be split to form multiple CUs, and each CU has an associated partitioning into prediction units (PUs) and a tree of transform units (TUs).

3) **Prediction units and prediction blocks (PBs):** The decision whether to code a picture area using interpicture or intrapicture prediction is made at the CU level. A PU partitioning structure has its root at the CU level. Depending on the basic prediction-type decision, the luma and chroma CBs can then be further split in size and predicted from luma and chroma prediction blocks (PBs). HEVC supports variable PB sizes from 64×64 down to 4×4 samples.

4) **TUs and transform blocks:** The prediction residual is coded using block transforms. A TU tree structure has its root at the CU level. The luma CB residual may be identical to the luma transform block (TB) or may be further split into smaller luma TBs. The same applies to the Chroma TBs. Integer basis functions similar to those of a discrete cosine transform (DCT) are defined for the square TB sizes 4×4, 8×8, 16×16, and 32×32. For the 4×4 transform of luma intrapicture prediction residuals, an integer transform derived from a form of discrete sine transform (DST) is alternatively specified.

5) **Motion vector signaling:** Advanced motion vector prediction (AMVP) is used, including derivation of several most probable candidates based on data from adjacent PBs and the reference picture. A merge mode for MV coding can also be used, allowing the inheritance of MVs from temporally or spatially neighboring PBs. Moreover, compared to H.264/MPEG-4 AVC, improved skipped and direct motion inference is also specified.

6) **Motion compensation:** Quarter-sample precision is used for the MVs, and 7-tap or 8-tap filters are used for interpolation of fractional-sample positions (compared to six-tap filtering of half-sample positions followed by linear interpolation for quarter-sample positions in H.264/MPEG-4 AVC). Similar to H.264/MPEG-4 AVC, multiple reference pictures are used. For each PB, either one or two motion vectors can be transmitted, resulting either in unipredictive or bipredictive coding, respectively. As in H.264/MPEG-4 AVC, a scaling and offset operation may be applied to the prediction signal(s) in a manner known as weighted prediction.

7) **Intrapicture prediction:** The decoded boundary samples of adjacent blocks are used as reference data for spatial prediction in regions where interpicture prediction is not performed. Intrapicture prediction supports 33 directional modes (compared to eight such modes in H.264/MPEG-4 AVC), plus planar (surface fitting) and DC (flat) prediction modes. The selected

intrapicture prediction modes are encoded by deriving most probable modes (e.g., prediction directions) based on those of previously decoded neighboring PBs.

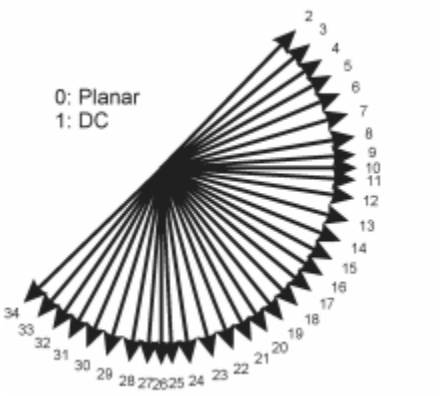

Fig 2: Modes and directional orientations for intrapicture prediction [19]

8) **Quantization control:** As in H.264/MPEG-4 AVC, uniform reconstruction quantization (URQ) is used in HEVC, with quantization scaling matrices supported for the various transform block sizes.

9) **Entropy coding:** Context adaptive binary arithmetic coding (CABAC) is used for entropy coding. This is similar to the CABAC scheme in H.264/MPEG-4 AVC, but has undergone several improvements to improve its throughput speed (especially for parallel-processing architectures) and its compression performance, and to reduce its context memory requirements.

10) **In-loop deblocking filtering:** A deblocking filter similar to the one used in H.264/MPEG-4 AVC is operated within the interpicture prediction loop. However, the design is simplified in regard to its decision-making and filtering processes, and is made more friendly to parallel processing.

11) **Sample adaptive offset (SAO):** A nonlinear amplitude mapping is introduced within the interpicture prediction loop after the deblocking filter. Its goal is to better reconstruct the original signal amplitudes by using a look-up table that is described by a few additional parameters that can be determined by histogram analysis at the encoder side.

Implementation of HEVC encoder with SIMD optimization:

1) **Complexity Analysis of HEVC Encoder**:

In order to identify the most time-consuming modules in the HEVC encoder, we analyze the execution time of the HM 6.2 encoder over Intel i5-750 2.67GHz CPU with 4G Byte memory under the Windows 7 operating system. For experiment, sequences of 720p (1280x720)

resolution in random-access situation are used. The Intra period is set to 32 and the GOP size is 8. We run 100 frames for each sequence under the QP value of 32.

The proportion of the average execution time of HM 6.2 encoder major modules is shown in Fig. 1, from which we can find that the most time-consuming modules are MC, Hadamard transform, SAD and SSD calculations, integer transform, memory operations and rate-distortion optimization quantization (RDOQ). The computational performance of the first four modules can be effectively improved by using SIMD methods.

2) **Intel SSE instructions:**

Streaming SIMD Extensions (SSE) is the SIMD instruction set extension over the x86architecture. It is further enhanced to SSE2, SSE3, SSSE3 and SSE4 subsequently. SSE contains eight 128-bit registers originally, known as XMM0 through XMM7. And the number of register is extended to sixteen in AMD64. Each 128-bit register can be divided into two 64-bit integers, four 32-bit integers, eight 16-bit short integers or sixteen 8-bit bytes. With SSE series instructions, several XMM registers can be operated at the same time, indicating considerable data-level parallelism. In Fig. 3, the most time consuming modules are all integer algorithm modules. Thus we only use integer SIMD instructions.

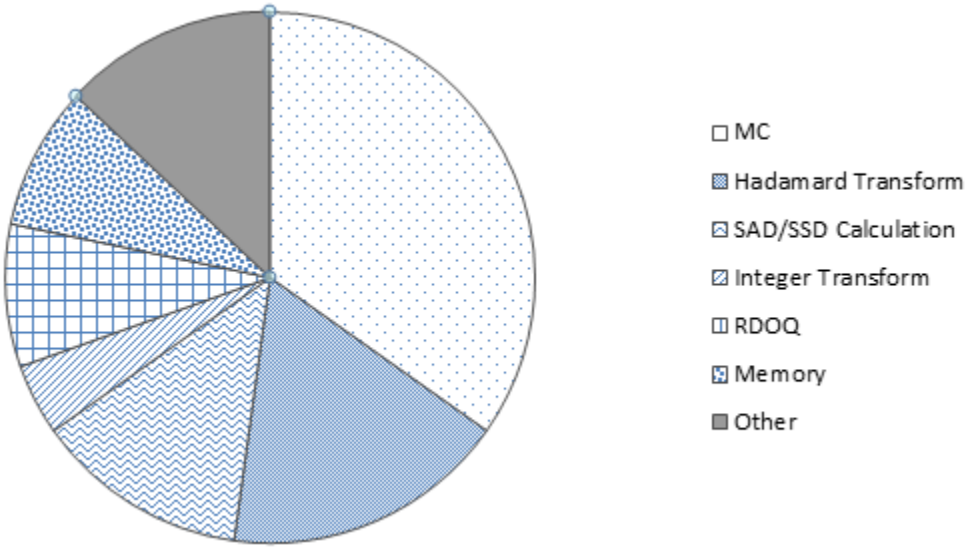

Figure 3: Execution Time Analysis of HEVC Encoder [19]

3) **Motion Compensation:**

MC is based on 1/4 pixel accuracy reconstructed pictures in HEVC. 8-tap luminance interpolation filters and 4-tap chrominance interpolation filters are utilized to obtain sub-pixels at

fractional positions. The 8-tap luminance interpolation and 4-tap chrominance methods are shown in Fig. 4 and Fig. 5.

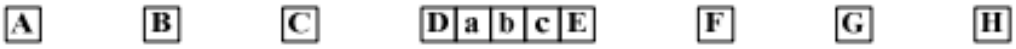

Figure 4: 8-tap Luminance Interpolation [17]

Figure 5: 4-tap Chrominance Interpolation [17]

In Fig.4 and Fig.5, the uppercase letters denote integer pixels and the lowercase letter denote sub-pixels. Several neighbor pixels are used to get the sub-pixels [17]:

$$a = c0*A+c1*B+c2*C+c3*D+c4*E+c5*F+c6*G+c7*H, \quad (1.1)$$

and

$$e = c0'*I+c1'*J+c2'*K+c3'*L, \quad (1.2)$$

where c0-c7 and c0'-c3' are interpolation coefficients. This is a typical vector product algorithm. We propose a fast implementation of MC by optimizing row and column sub-pixel interpolation for luminance and chrominance components based on Intel SIMD instructions. The vector product in MC is somewhat tricky. One vector contains continuously aligned pixels, which are unsigned bytes. And the other contains interpolation coefficients, which are signed bytes. The SSSE3 instruction PMADDUBSW can be used to compute the vector product for row luminance sub pixel interpolation. The PMADDUBSW instruction takes two 128-bit SSE registers as operands, with the first one containing sixteen unsigned 8-bit integers, and the second one containing sixteen signed 8-bit integers. With this instruction, we only need to sum the values in the result register to get the final results. We consider 8-pixel horizontal luminance interpolation first. To calculate eight pixels, we need fifteen pixels in total, which can be loaded into a single register. Then we get all the necessary vectors by four PSHUFB instructions. With four PMADDUBSW instructions, vector product is finished. Finally, we use three PHADDW to summarize the result into one register. The register contains the eight aligned necessary results then. The implementation is shown in Fig. 6.

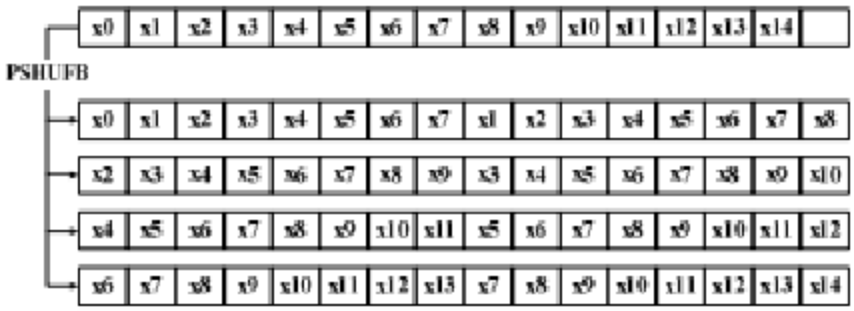

Figure 6: Luminance Horizontal Interpolation [17]

It can be seen that, with four PSHUFB instructions, wanted vectors are obtained and make the most use of four registers. The solution for the 4-pixel horizontal luminance interpolation is almost the same. We can use the first two PSHUFB instructions to process it. Then only two PHADDW instructions are needed to summarize. Then we consider 8-pixel horizontal chrominance interpolation. To interpolate eight pixels, we need eleven pixels, which can be loaded into a single register. Then we get all the necessary vectors by two PSHUFB instructions. With two PMADDUBSW instructions, vector product is finished. Finally, we use one PHADDW to summarize the result into one register. The implementation is shown in Fig. 7.

Figure 7: Chrominance Horizontal Interpolation [17]

The solution for 4-pixel horizontal chrominance interpolation is similar. With only one PSHUFB, one PMADDUBSW and one PHADDW instructions, we can finish this task. For vertical interpolation, we should apply vector products on 16-bit intermediates. We use a simple algorithm in this step. If the block width is 4, we load four intermediates with one MOVQ instruction, expand each 16-bit intermediate into 32-bit with one PMOVSXWD instruction, then multiply it with one PMULLUD instruction and accumulate it with one PADDD instruction. If the block width is 8 or larger, we load eight intermediates each time with one MOVDQU instruction, multiply it with one PMULLW and one PMULHW instructions, then shuffle the two result register with one PUNPCKLWD and one PUNPCKHWD instructions, and finally accumulate them with two PADDD instructions.

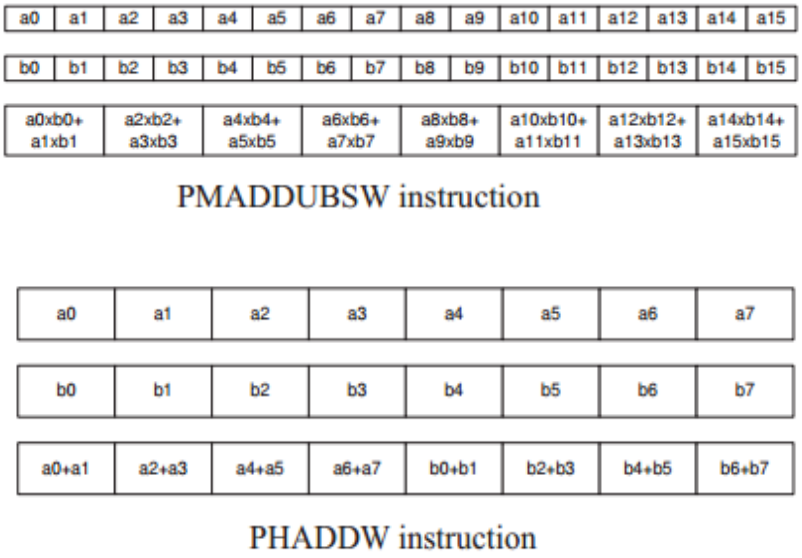

Fig 8: Instruction structure [18]

Interpolation Algorithm:

In the latest standard HEVC, three types of 8-tap filters are adopted as shown in equation (1)-(3). The detail of the equations can be found in [16]. According to the fractional position to be predicted, one of the three filters is applied for interpolation.

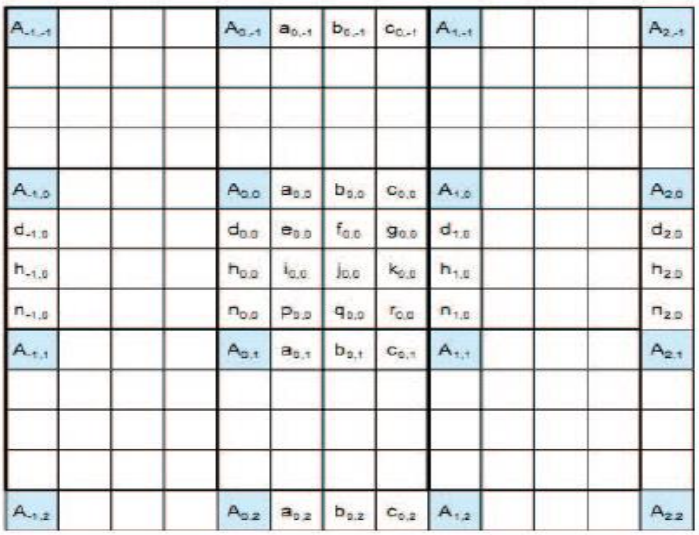

Figure 9: Integer, Half and Quarter Pixel of Luma Prediction. [17]

$$A_{0,0} = (-A_{-3,0} + 4 * A_{-2,0} - 10 * A_{-1,0} + 57 * A_{0,0} + 19 * A_{1,0} - 7 * A_{2,0} + 3 * A_{3,0} - A_{4,0} + 32) >> 6 \quad (1)$$

$$B_{0,0} = (-A_{-3,0} + 4 * A_{-2,0} - 11 * A_{-1,0} + 40 * A_{0,0} + 40 * A_{1,0} - 11 * A_{2,0} + 4 * A_{3,0} - A_{4,0} + 32) >> 6 \quad (2)$$

$$C_{0,0} = (-A_{-3,0} + 3 * A_{-2,0} - 7 * A_{-1,0} + 19 * A_{0,0} + 57 * A_{1,0} - 10 * A_{2,0} + 4 * A_{3,0} - A_{4,0} + 32) >> 6 \quad (3)$$

Fig. 8 shows the integer, half and quarter positions of luma component. Capital letters represent the integer position, it is directly output without calculation. a, b, c are calculated by applying the equation (1)-(3)[18] to the nearest integer pixels in horizontal direction, respectively. And d, h, n are calculated by applying the above equation to the integer pixels in vertical direction, respectively. Rest fractional position pixels are calculated by applying above equation to the unrounded intermediate of d, h, n. For example, e, f, g is calculated by applying the equation (1)-(3) to the unrounded intermediate value of di,0, I =-3, -2.. 4 in the horizontal direction, respectively. The calculation process for i, j, k, p, q, r is the same as e, f, g.

Thus, there are three calculation patterns for luma interpolation. One pattern is in position of integer, a, b and c, only 1 filter operation is needed to obtain the predicted values. The second pattern is d, h, n and the third pattern is the rest 9 fractional positions. So in the worst case, 11x11 reference pixel values are required for interpolation of one 4x4 sub-block. A pipelined architecture should be designed suitable for these two calculation patterns. Besides, since the hardware architecture of the filter essentially determines the performance of luma interpolation engine, optimized filter architecture is required to be proposed to reduce implementation area.

Interpolation Process with SSE Instruction:

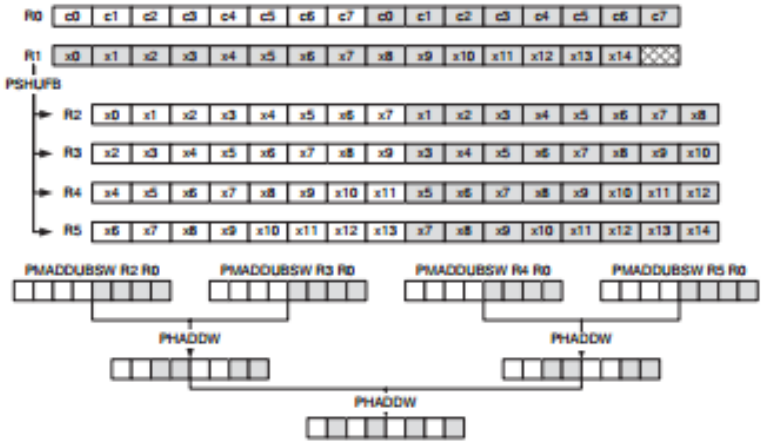

Figure 10: Luminance Row Interpolation [18]

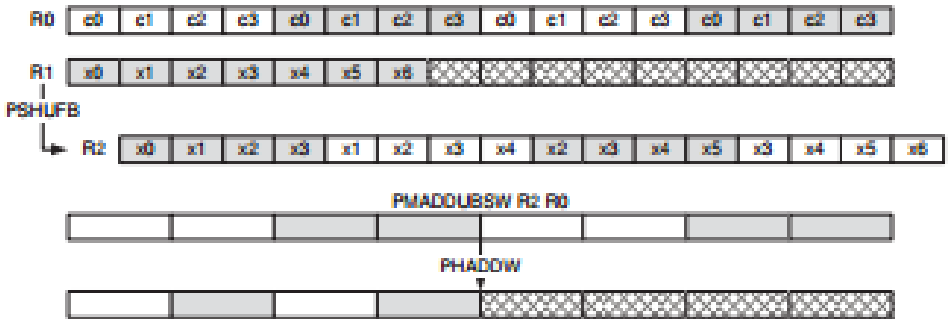

Figure 11: Chrominance Row Interpolation [18]

Figure 10 shows an example of the row interpolation process. We first load the proper interpolation parameter (eight 8-bit signed integers) twice into a 128-bit SSE register R0, and load 8+8-1=15 relative pixels into R1. Then we shuffle R1 to R2 in the right order, using SSSE3 instruction PSHUFB. Now that we get all the data needed to calculate the first two sub-pixels, we perform PMADDUBSW instruction on R2 and R0 and get eight 16-bit integers in R2, with the sum of each four being a new sub-pixel. After that, we shuffle R1 to R3 and perform PMADDUBSW again, and thereby get the third and the fourth sub-pixel in R3. By repeating this, we get eight new subpixels in R2-R5, with each split in four 16-bit integers. In the end, we perform PHADDW instruction on (R2,R3), (R4,R5), and (R2,R4), and finally get eight complete new pixel in R2, each being a 16-bit value, which can be stored with one single MOVDQA instruction. In most cases, the addresses of the source pixels in MC are not aligned, thus it is of great importance to reduce the load/store instructions to minimize the performance penalties.

Since the proposed method calculates eight sub-pixels with only two load/store instructions and avoids waste of parallel computing power of SIMD, it is expected to greatly outperform the speed of the original method.

For column interpolation, the problem is simpler. We use R0 as an accumulator. In each iteration, we load one row of data into R1, do the packed multiplication with proper co-efficient and add the results to R0. After processing all eight rows, we get the required eight 16-bit sub-pixels in R0. For chrominance interpolation, HEVC uses 4-tap filters, thus in row interpolation we can use one PMADDUBSW instruction and one PHADDW instruction to get four results with proper data shuffled according to Figure 6. For column chrominance interpolation, similar method to luminance column interpolation is applied.

Goal:

As proposed by implementing SIMD on various block HEVC encoder there will be significant optimization on time scale without affecting the throughput and quality of video compression. This will take advantage of hardware used for the implementation and will leverage all the performance needed for compression. Output will be time comparison between normal HEVC encoder and encoder leveraging SIMD technology using specific video sequences.

References

[1] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm, and T. Wiegand, High Efficiency Video Coding (HEVC) Text Specification Draft 9, document JCTVC-K1003, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2012.

[2] Video Codec for Audiovisual Services at px64 kbit/s, ITU-T Rec. H.261, version 1: Nov. 1990, version 2: Mar. 1993.

[3] Video Coding for Low Bit Rate Communication, ITU-T Rec. H.263, Nov. 1995 (and subsequent editions).

[4] Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s—Part 2: Video, ISO/IEC 11172-2 (MPEG-1), ISO/IEC JTC 1, 1993.

[5] Coding of Audio-Visual Objects—Part 2: Visual, ISO/IEC 14496-2 (MPEG-4 Visual version 1), ISO/IEC JTC 1, Apr. 1999 (and subsequent editions).

[6] Generic Coding of Moving Pictures and Associated Audio Information— Part 2: Video, ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG 2 Video), ITU-T and ISO/IEC JTC 1, Nov. 1994.

[7] Advanced Video Coding for Generic Audio-Visual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC JTC 1, May 2003 (and subsequent editions).

[8] H. Samet, “The quadtree and related hierarchical data structures,” Comput. Survey, vol. 16, no. 2, pp. 187–260, Jun. 1984.

[9] Intel Corp., Intel® 64 and IA-32 Architectures Software Developers Manual. http://download.intel.com/products/processor/manual/325383.pdf

[10] JCT-VC, “HM6: High Efficiency Video Coding (HEVC) Test Model 6 Encoder Description,”JCTVC-H1002, Feb. 2012.

[11] D. Marpe et al., “Video compression using nested quadtree structures, leaf merging, and improved techniques for motion representation and entropy coding,” IEEE Trans. Circuits Syst. Video Technol., vol. 20, no. 12, pp. 1676 –1687, Dec. 2010.

[12] JCT-VC, “On the motion compensation process,” JCTVC-F537, Jul. 2011.

[13] H. S. Malvar, A. Hallapuro, M. Karczewicz and L. Kerofsky, “Low-complexity transform and quantization in H.264/AVC,” IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 598 – 603, Jul. 2003.

[14] JCT-VC, “ALF using vertical-size 5 filters with up to 9 coefficients,” JCTVC-F303, Jul. 2011.

[15] Y.-K. Chen, E. Q. Li, X. Zhou and S. L. Ge, “Implementation of H.264 encoder and decoder on personal computers,” J. Visual Commun. Image Representation, vol. 17, no. 2, pp. 509 – 532, Apr. 2006.

[16] ITU-T, WD3: Working Draft 3 of High-Efficiency Video Coding, JCTVC-E603, March, 2011.

[17] Leju Yan; Yizhou Duan; Jun Sun; Zongming Guo , “Implementation of HEVC decoder on x86 processors with SIMD optimization,” VCIP, pp. 1-6, Nov. 2012.

[18] Keji Chen; Yizhou Duan; Leju Yan; Jun Sun; Zongming Guo,“Efficient SIMD optimization of HEVC encoder over X86 processors,” APSIPA ASC 2012 Asia-Pacific , pp. 1-4, Dec. 2012.

[19] G. J. Sullivan, J.-R.Ohm, W.-J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard”, IEEE Trans. Circuits and Systems for Video Technology, Vol. 22, No. 12, pp. 1649-1668, Dec. 2012.