IEIE Transactions on Smart Processing and Computing, vol. 3, no. 1, February 2014
http://dx.doi.org/10.5573/IEIESPC.2014.3.1.1

IEIE Transactions on Smart Processing and Computing

Low-latency SAO Architecture and its SIMD Optimization for HEVC Decoder

Yong-Hwan Kim[1,2], Dong-Hyeok Kim[2], Joo-Young Yi[2], and Je-Woo Kim[2]

[1] Smart Media Research Center, Korea Electronics Technology Institute / Seoul, 121-835, Korea yonghwan@keti.re.kr
[2] Multimedia IP Research Center, Korea Electronics Technology Institute / Seongnam-si, Gyeongggi-do 483-816, Korea
{kdh007, jyyi, jwkim}@keti.re.kr

***** Corresponding Author: Yong-Hwan Kim

Received October 20, 2013; Revised October 31, 2013; Accepted November 15, 2013; Published February 28, 2014

** Regular Paper*

Abstract**:** This paper proposes a low-latency Sample Adaptive Offset filter (SAO) architecture and its Single Instruction Multiple Data (SIMD) optimization scheme to achieve fast High Efficiency Video Coding (HEVC) decoding in a multi-core environment. According to the HEVC standard and its Test Model (HM), SAO operation is performed only at the picture level. Most realtime decoders, however, execute their sub-modules on a Coding Tree Unit (CTU) basis to reduce the latency and memory bandwidth. The proposed low-latency SAO architecture has the following advantages over picture-based SAO: 1) significantly less memory requirements, and 2) low-latency property enabling efficient pipelined multi-core decoding. In addition, SIMD optimization of SAO filtering can reduce the SAO filtering time significantly. The simulation results showed that the proposed low-latency SAO architecture with significantly less memory usage, produces a similar decoding time as a picture-based SAO in single-core decoding. Furthermore, the SIMD optimization scheme reduces the SAO filtering time by approximately 509% and increases the total decoding speed by approximately 7% compared to the existing look-up table approach of HM.

Keywords**:** HEVC, SAO, Low-latency, Multi-core, SIMD

1. Introduction

HIGH Efficiency Video Coding (HEVC), which is also known as the H.265 video codec, is the latest video compression standard developed by the Joint Collaborative Team on Video Coding (JCT-VC) group, which was established by the ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Expert Group (VCEG) [1, 2]. The HEVC is expected to achieve an up to 50% bit-rate reduction with the same visual quality relative to the former Advanced Video Coding (AVC/H.264) standard [3].

In HEVC, pictures are divided uniformly into square blocks called Coding Tree Units (CTUs), which are similar to the Macroblocks used in earlier standards. These CTUs are divided further in a quadtree structure to form Coding Units (CUs), which form the basic processing unit. In-loop filtering of the HEVC consists of two stages. The first stage is deblocking filter (DF) and second stage is the Sample Adaptive Offset (SAO) filter, as shown in Fig. 1.

The SAO is a newly adopted tool in HEVC to enhance both the subjective and objective quality [4]. The DF reduces blocking artifacts, and the SAO reduces both ringing and banding artifacts. In addition, the SAO can enhance the edge sharpness or smoothing.

According to the HEVC standard and its Test Model (HM), SAO operation is performed only at the picture level [1, 5]. That is, SAO filtering is an inherently picture-based operation because it requires deblocked pixels from all of its eight neighbors (left, above, right, below, above-left, above-right, below-left, and below-right) CTUs. Most realtime decoders, however, execute their sub-modules on a CTU basis for reducing the latency and memory bandwidth. In addition, multi-core decoding is widely used since large picture, such as UHD (3840x2160), decoding is not possible in single-core decoding. Therefore, a CTU-based, low-latency SAO filtering architecture is essential for utilizing pipelined parallel decoding efficiently. Parveen.G.B et al. proposed a low-latency SAO encoding architecture for a multi-core encoding environment, where method-B appears to be a good trade-off between the

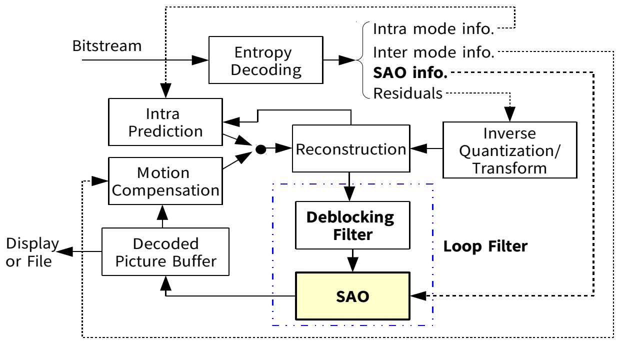


Fig. 1. Block diagram of HEVC decoder.

latency and compression ratio [6]. P. N. Subramanya et al. proposed a low-latency SAO decoding architecture for a multi-core HW decoding environment, where method C provides CTU-based low-latency SAO filtering [7]. Method C has two drawbacks: 1) the side information overhead is high because it processes the pixel data from four CTUs simultaneously, 2) less efficient in Single Instruction Multiple Data (SIMD) optimization, which is indispensable to all SW codecs, due to the many conditional branches and data alignment problems [8].

This paper presents a low-latency CTU-based SAO filtering architecture to enable efficently pipelined parallel decoding and effective SIMD optimization. In addition, a SIMD optimization scheme of SAO filtering is proposed to reduce the SAO filtering time significantly.

This paper is organized as follows. Section 2 explains a SAO overview and the existing picture-based SAO filtering architecture. The proposed algorithms are presented in detail in Section 3. Section 4 reports the experimental results and compares those with the existing algorithm, and Section 5 concludes the paper.

2. SAO Overview

SAO is a non-linear amplitude mapping filter that operates on deblocked pixels [1, 2, 4]. Each CTU can have one of three types of SAO filtering: (1) No SAO filtering, (2) Band Offset (BO), and (3) Edge Offset (EO). Both BO and EO add a certain offset value to a sample, where the offset value of BO is chosen from the received lookup table by the sample magnitude. On the other hand, the offset value of EO is chosen from the received lookup table by a edge direction and gradient.

2.1 Band Offset (BO)

BO classifies all pixels of a CTU into multiple bands, where each band contains pixels with the same intensity. For example, 8-bit pixel intensities (0-255) are divided into 32 fixed bands with the width of the band as 8 samples. Only four consecutive bands and offsets are selected and signaled to a decoder, as shown in Fig. 2. Note that BO does not use the pixels of the neighboring CTUs, and four band offsets can be wrapped around. The BO filtering equation is expressed as Eq. (1).

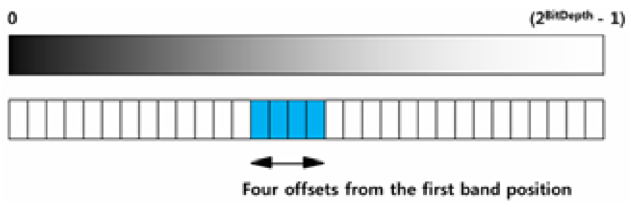


Fig. 2. Band offsets.

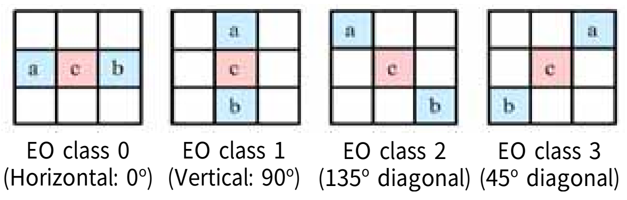

a c b
a c b
a c b
a c b
EO class 0 (Horizontal: 0°)
EO class 1 (Vertical: 90°)
EO class 2 (135° diagonal)
EO class 3 (45° diagonal)

Fig. 3. Four EO classes.

Table 1. EO sample classification rule.

Category (Edge Type)	Condition	Description
0	c==a && c==b	No SAO filtering
1	c < a && c < b	Local minima
2	(c < a && c==b) || (c==a && c < b)	Negative edge
3	(c > a && c==b) || (c==a && c > b)	Positive edge
4	c > a && c > b	Local maxima

$$m = src[x][y] >> (bitDepth-5),$$
$$dst[x][y] = Clip3(0, (1<<bitDepth)-1, src[x][y] + BoOffsetTab[m]) \quad (1)$$

where Clip3(min, max, a) = (a > max) ? max : ((a < min) ? min : a). The src[x][y] and dst[x][y] represent a deblocked pixel and SAO-filtered pixel, respectively, m means a quantized pixel, and BoOffsetTab[0..31] holds four BO offset values and 28 zero values. x and y are pixel coordinates with a range of values: x=0..CtbWidth, y=0..CtbHeight, where CtbWidth and CtbHeight represent Coding Tree Block (CTB) width and height in pixel units, respectively.

2.2 Edge Offset (EO)

EO uses the neighboring pixels including those of neighboring CTUs to compute edge directional information. Fig. 3 shows four EO classes that specify the edge direction, where the c represent current pixel and both b and c represent the neighboring pixels. Note that in a CTU, one EO class and four edge offset values according to pre-defined category are signaled to a decoder.

Given a EO class, each sample is classified into five categories, as shown in Table 1, where category 0 means no SAO filtering.

$$m = Sign(src[x][y] - src[x-1][y]),$$
$$n = Sign(src[x][y] - src[x+1][y]), \quad (2)$$

dst[x][y] = Clip3(0, (1<<bitDepth)-1,
src[x][y] + EoOffsetTab[m + n + 2])

Eq. (2) shows EO class 0 filtering equation, where Sign(a) = (a > 0) ? 1 : ((a==0) ? 0 : -1) and EoOffsetTab [0..4] has five EO offset values including one zero value. The other EO class filtering equations can be derived easily by adjusting the x and y coordinates according to the angle of Fig. 3.

2.3 Picture-based SAO Architecture

A picture-based SAO architecture is the default method of performing SAO filtering after decoding and deblocking a picture in the HEVC standard and HM [1, 5]. Fig. 4 shows a flowchart of the architecture, where saoPicBuf is the backup buffer for retaining the original deblocked pixels. The srcPicBuf holds the original deblocked pixels and SAO filtered pixels as a result. Unlike DF, SAO filtering always uses the original deblocked pixels, not the previously SAO-filtered pixels. This is why the picture-sized backup buffer is required. The iCtu represents the CTU number and iSizeInCtu means the total number of CTUs in a picture. The saoPicBuf is the input and the srcPicBuf is the output of the SaoFilterCTU_HM() function, which filters the current CTU pixels by selectively referencing the neighboring CTUs.

After normal SAO filtering for a picture using iterative CTU-based filtering, the PCM samples and losslessly coded samples are restored in the RestorePcmSamplesPic() function.

This architecture has two major drawbacks: 1) high latency, which makes it difficult to achieve efficiently pipelined parallel decoding; and 2) large memory requirements due to picture backup buffer (i.e., saoPicBuf).

3. The Proposed Scheme

To overcome the drawbacks of a picture-based SAO architecture, low-latency CTU-based SAO architecture was designed, which also takes into account the following two points: 1) SIMD-friendly structure, and 2) Easy implementation covering various Tile and slice combinations. Finally, SIMD optimization of SAO filtering is proposed to reduce significantly the SAO filtering time itself.

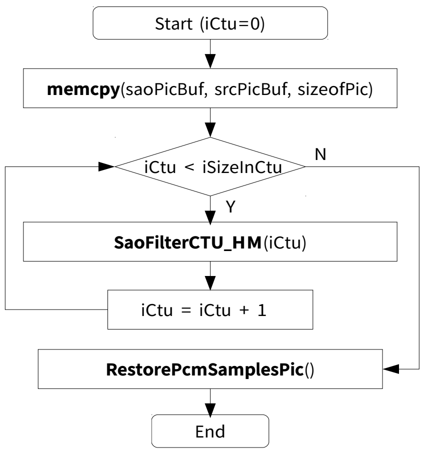
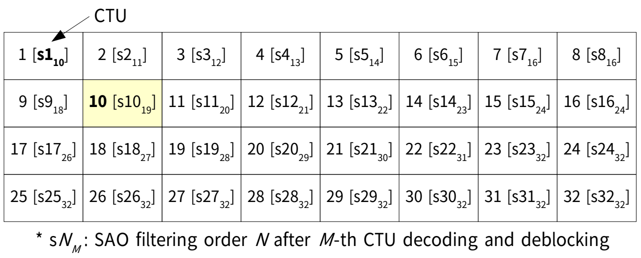


Fig. 4. Flowchart of picture-based SAO architecture.

3.1 Low-latency SAO Architecture

Unlike the HM, which executes DF in picture-basis, we assume that DF is performed on a CTU-basis. Basically, the SAO of the current CTU can begin filtering after decoding and deblocking its below-right CTU because it requires deblocked pixels of the eight neighboring CTUs. For example, after deblocking CTU #10, the SAO of CTU #1 can be started, as shown in Fig. 5. The proposed architecture follows the basic nature of the SAO precisely, unlike previous work [7]. That is, SAO filtering is performed on the entire pixels of a CTU after deblocking its below-right CTU. Note that last column CTUs are processed immediately after its left CTU, and the last CTU row is processed separately after deblocking the last CTU of a picture. Fig. 6 shows the filtering order in the case of the Tile structure. Fig. 7 presents a flowchart of the proposed architecture, where iCtuX and iCtuY represent the x and y coordinates of a CTU, respectively. The iWidthInCtu and iHeightInCtu means the number of CTUs in a CTU row and column, respectively. The proposed architecture has the following three steps: 1) process an above-left CTU if available, 2) process an above CTU if the current CTU is the right-most CTU, and 3) process the last CTU row if the current CTU is the last one in the picture. Note that Fig. 7 explains SAO filtering order of Figs. 5 and 6.

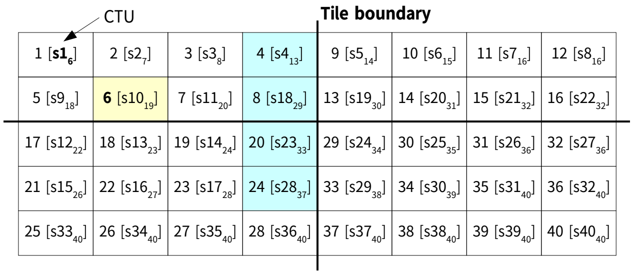
To reduce the backup buffer overhead of the picture-based SAO and enable low-latency SAO filtering, the CTU-based SAO buffer structure was designed. First, the proposed CTU-based SAO architecture requires four types of buffer, as shown in Fig. 8 and Table 2. That is, the original deblocked pixels such as all the internal bottom-



1 [$\mathbf{s1_{10}}$]	2 [$s2_{11}$]	3 [$s3_{12}$]	4 [$s4_{13}$]	5 [$s5_{14}$]	6 [$s6_{15}$]	7 [$s7_{16}$]	8 [$s8_{16}$]
9 [$s9_{18}$]	**10** [$s10_{19}$]	11 [$s11_{20}$]	12 [$s12_{21}$]	13 [$s13_{22}$]	14 [$s14_{23}$]	15 [$s15_{24}$]	16 [$s16_{24}$]
17 [$s17_{26}$]	18 [$s18_{27}$]	19 [$s19_{28}$]	20 [$s20_{29}$]	21 [$s21_{30}$]	22 [$s22_{31}$]	23 [$s23_{32}$]	24 [$s24_{32}$]
25 [$s25_{32}$]	26 [$s26_{32}$]	27 [$s27_{32}$]	28 [$s28_{32}$]	29 [$s29_{32}$]	30 [$s30_{32}$]	31 [$s31_{32}$]	32 [$s32_{32}$]

* sN_M: SAO filtering order N after M-th CTU decoding and deblocking

Fig. 5. Low-latency SAO filtering order without a Tile.


CTU
Tile boundary

1 [$\mathbf{s1_{6}}$]	2 [$s2_{7}$]	3 [$s3_{8}$]	4 [$s4_{13}$]	9 [$s5_{14}$]	10 [$s6_{15}$]	11 [$s7_{16}$]	12 [$s8_{16}$]
5 [$s9_{18}$]	**6** [$s10_{19}$]	7 [$s11_{20}$]	8 [$s18_{29}$]	13 [$s19_{30}$]	14 [$s20_{31}$]	15 [$s21_{32}$]	16 [$s22_{32}$]
17 [$s12_{22}$]	18 [$s13_{23}$]	19 [$s14_{24}$]	20 [$s23_{33}$]	29 [$s24_{34}$]	30 [$s25_{35}$]	31 [$s26_{36}$]	32 [$s27_{36}$]
21 [$s15_{26}$]	22 [$s16_{27}$]	23 [$s17_{28}$]	24 [$s28_{37}$]	33 [$s29_{38}$]	34 [$s30_{39}$]	35 [$s31_{40}$]	36 [$s32_{40}$]
25 [$s33_{40}$]	26 [$s34_{40}$]	27 [$s35_{40}$]	28 [$s36_{40}$]	37 [$s37_{40}$]	38 [$s38_{40}$]	39 [$s39_{40}$]	40 [$s40_{40}$]

Fig. 6. Low-latency SAO filtering order with four Tiles.

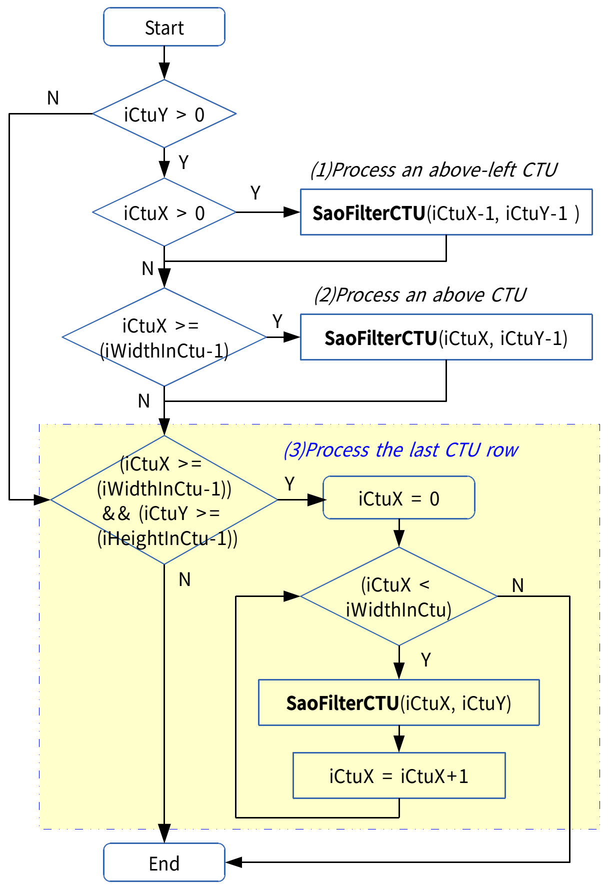

Start
iCtuY > 0
N
Y
iCtuX > 0
Y
(1)Process an above-left CTU
SaoFilterCTU(iCtuX-1, iCtuY-1)
N
iCtuX >= (iWidthInCtu-1)
Y
(2)Process an above CTU
SaoFilterCTU(iCtuX, iCtuY-1)
N
(3)Process the last CTU row
(iCtuX >= (iWidthInCtu-1)) && (iCtuY >= (iHeightInCtu-1))
Y
iCtuX = 0
N
(iCtuX < iWidthInCtu)
N
Y
SaoFilterCTU(iCtuX, iCtuY)
iCtuX = iCtuX+1
End

Fig. 7. Flowchart of the low-latency SAO filtering architecture.

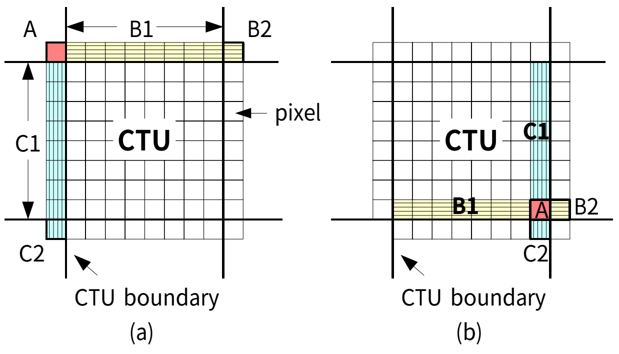


Fig. 8. CTU boundary pixels (a) Neighboring pixels required for filtering current CTU, (b) Current CTU's pixels (A, B1, B2, C1, and C2) to be stored for filtering the other CTUs.

most (B1) pixels, a external right-bottom (B2) pixel, internal right-most (C1) pixels, an external below-right (C2) pixel, and an internal bottom-right pixel (A) of current CTU should be stored for SAO filtering of the other CTUs. Note that A pixel should be stored at separate buffer for the below-right CTU because the same pixel in the B and C buffers is replaced with other one in the below and right CTU. In Table 2, iCtbWidthY and iCtbWidthC represent Y and Cb/Cr CTB width, respectively.

For example, assume the UHD video (3840x2160) stream, where typically iCtbWidthY = 64 and iCtbWidthC

Table 2. Four buffers for the proposed SAO architecture.

Buffer	Size	Description
A	Y: iWidthInCtu * (iHeightInCtu-1) Cb: the same as above Cr: the same as above	An Above-left pixel
B (B1 and B2)	Y: iWidthInCtu * (iCtbWidthY+1) Cb: iWidthInCtu * (iCtbWidthC+1) Cr: the same as above	Above and Above-right pixels [*line buffer*]
C (C1 and C2)	Y: iHeightInCtu * (iCtbWidthY+1) Cb: iHeightInCtu * (iCtbWidthC+1) Cr: the same as above	Left and below-left pixels [*line buffer*]

D	Y:(iCtbWidthY+2)*(iCtbWidthY+2)
	Cb:(iCtbWidthC+2)*(iCtbWidthC+2)
	Cr: the same as above
		Current CTU buffer including all neighboring pixels

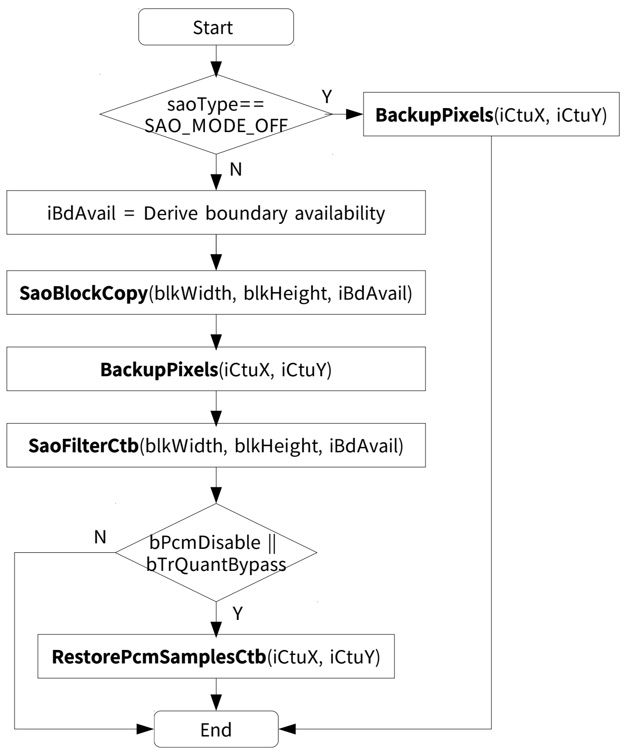

Start
saoType== SAO_MODE_OFF
Y
BackupPixels(iCtuX, iCtuY)
N
iBdAvail = Derive boundary availability
SaoBlockCopy(blkWidth, blkHeight, iBdAvail)
BackupPixels(iCtuX, iCtuY)
SaoFilterCtb(blkWidth, blkHeight, iBdAvail)
N
bPcmDisable || bTrQuantBypass
Y
RestorePcmSamplesCtb(iCtuX, iCtuY)
End

Fig. 9. Flowchart of the SaoFilterCTU().

= 32. The picture-based SAO requires a buffer with size equal to 3840x2160x1.5=11.9 Mbytes. The proposed low-latency CTU-based SAO architecture requires a buffer with a size equal to 24.3 Kbytes (Table 2), which amounts to only 0.2% compared to the picture-based SAO.

Fig. 9 presents a flowchart of the proposed SAO filtering algorithm using the four buffers in Table 2, which is SaoFilterCTU() in Fig. 7. In Fig. 9, only Y filtering is shown for simplicity. The BackupPixels() function stores some pixels of the current, right, and below CTUs before SAO filtering, where the A, B, and C buffers are used. The SaoBlockCopy() function copies pixels of: (a) current CTU from srcPicBuf buffer, (b) right and below CTUs from srcPicBuf buffer if available, (c) above-left CTU from A buffer, (d) above and above-right CTUs from B

```
If ((iCtbY+1) < iHeightInCtu)
{
  (1)Store bottom line pixels of current CTU
     to B_buffer[iCtbX][] (B1);
  If (iCtbX < (iWidthInCtu-1))
  {
    (2)Store an external right-bottom pixel
       of current CTU to
       B_buffer[iCtbX][iCtbWidth] (B2);
    (3)Store an internal bottom-right pixel
       of current CTU to
       A_buffer[iCtbY][iCtbX];
  }
}
If (iCtbX < (iWidthInCtu-1))
{
  (4)Store internal right-most pixels of
     current CTU to C_buffer[iCtbY][] (C1);
  If ((iCtbY+1) < iHeightInCtu)
  {
    (5)Store an external below-right pixel
       of current CTU to
       C_buffer[iCtbY][iCtbWidth] (C2);
  }
}
```

Fig. 10. Pseudo code of BackupPixels().

buffer, (e) left and left-below CTUs from C buffer, to D buffer. The SaoFilterCtb() function filters the entire pixels of the current CTU from D buffer as the input and srcPicBuf buffer as the output using Eq. (1) or (2) according to the SAO type and EO class.

The PCM and lossless samples were restored on a CTU-basis. The RestorePcmSamplesCtb() function selectively restores the original samples in a CTU.

Figs. 10 and 11 show the pseudo code of BackupPixels() and SaoBlockCopy() functions, respectively. The blkWidth and blkHeight of Figs. 9 and 11 represent the width and height of the Coding Block (CB) to be filtered, respectively, which can be different from iCtbWidth in the case of the right-most CTUs and bottom-most CTUs.

Picture-based pipelined parallel decoding [9] is possible using the proposed low-latency CTU-based SAO architecture, as shown in Fig. 12. For example, picture 2 (core 2) decoding can be started immediately after the SAO filtering second CTU row of picture 1 (core 1). In addition, picture 3 (core 3) decoding can be started immediately after SAO filtering the second CTU row of picture 2 (core 2). Therefore, in this example, three cores can decode the pictures simultaneously in pipelined manner. This property is very efficient for large picture decoding, such as a UHD video, because the decoding overhead is decentralized to the multi-core CPU. Note that the thread syncronization unit between the cores can be one out of a CTU, multiple CTUs, or CTU row.

3.2 SIMD Optimization Scheme

Eq. (2) was optimized using SIMD instruction [8]. The entire pixels of the current CTU are filtered in the proposed low-latency SAO architecture.

Fig. 13 shows the SIMD pseudo code. The proposed scheme is composed of seven steps: 1) load left 16 pixels, right 16 pixels, and current 16 pixels, 2) conversion from 8-bit pixels to 16-bit pixels, 3) calculation of the edge type per pixel using Table 1, 4) table lookup of EoOffsetTab[0...15], 5) add pixels and offset values, 6) save 16 pixels to destination buffer, 7) conditional pixel restoration. The CALC_ETYPE() macro calculates the 8 edge types simultaneously using Table 1. Sixteen table lookup operations are performed at once using _mm_shuffle_epi8() instruction, as shown in Fig. 14. Note that the blkWidth value is assumed to be equal to a multiple of 16 for simplicity. SIMD optimization of the remaining EO class 1-3 can be designed easily by adjusting step 1 and proper handling of conditional pixel

```
If (above CTU is available)
{
  If (above-left CTU is available)
  {
    (1)Copy A_buffer[iCtbY-1][iCtbX-1] to
       D_buffer[-1][-1];
  }
  (2)Copy B_buffer[iCtbX][] to
     D_buffer[-1][] (B1);
}
if (above-right CTU is available)
{
  (3)Copy B_buffer[iCtbX][blkWidth] to
     D_buffer[-1][blkWidth] (B2);
}

If (left CTU is available)
{
  (4)Copy C_buffer[iCtbY][] to
     D_buffer[][-1] (C1);
  If ( below-left CTU is available)
  {
    (5)Copy C_buffer[iCtbY][blkHeight]
       to D_buffer[blkHeight][-1] (C2);
  }
}

(6)Copy current CTU's pixels to
   D_buffer[][];

If (iCtbX < (iWidthInCtu-1))
{
  (7)Copy right CTU's left pixels to
     D_buffer[][blkWidth-1];
}
If (below CTU is available)
{
  (8)Copy below CTU's top pixels to
     D_buffer[blkHeight][];
}
```

Fig. 11. Pseudo code of SaoBlockCopy().

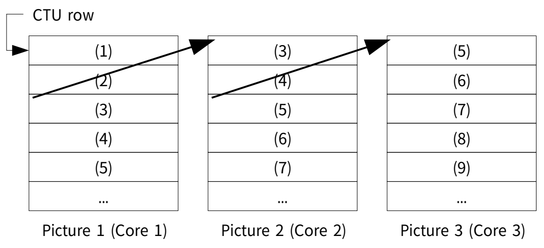


Fig. 12. Picture-based pipelined parallel decoding architecture.

```
#define CALC_ETYPE(srcA, srcC, srcB) {
  xmm3  = _mm_cmpgt_epi16(srcC, srcA);
  xmm4  = _mm_cmpgt_epi16(srcC, srcB);
  srcA  = _mm_cmplt_epi16(srcC, srcA);
  srcB  = _mm_cmplt_epi16(srcC, srcB);
  xmm3  = _mm_srli_epi16(xmm3, 15);
  xmm4  = _mm_srli_epi16(xmm4, 15);
  xmm3  = _mm_or_si128(xmm3, srcA);
  xmm4  = _mm_or_si128(xmm4, srcB);
  srcA  = _mm_add_epi16(xmm3, xmm4);
/* srcA => edge_type: -2, -1, 0, 1, or 2 */
}

blkWidth16 = blkWidth >> 4;
xmm7 = _mm_set1_epi8(2); // [2...2]
xmm8 = _mm_load_si128(EoOffsetTab);
for (y = 0; y < blkHeight; y++) {
  for(x = 0; x < blkWidth16; x++) {
  // (1) load left, current, right pixels
    xmm1 = _mm_loadu_si128(srcBlk-1);
    xmm0 = _mm_srli_si128(xmm1, 1);
    xmm2 = _mm_srli_si128(xmm1, 2);
  // (2) 16-bit conversion
    xmm1 = _mm_cvtepu8_epi16(xmm1);
    xmm0 = _mm_cvtepu8_epi16(xmm0);
    xmm2 = _mm_cvtepu8_epi16(xmm2);
  // (3) calculation of 8 edge types
    CALC_ETYPE(xmm1, xmm0, xmm2);

    xmm5 = _mm_loadu_si128(srcBlk-1 + 8);
    xmm6 = _mm_srli_si128(xmm5, 1);
    xmm2 = _mm_srli_si128(xmm5, 2);
    xmm5 = _mm_cvtepu8_epi16(xmm5);
    xmm6 = _mm_cvtepu8_epi16(xmm6);
    xmm2 = _mm_cvtepu8_epi16(xmm2);
    CALC_ETYPE(xmm5, xmm6, xmm2);

  // (4) 16 edge types and table lookup
    xmm1 = _mm_packs_epi16(xmm1, xmm5);
    xmm1 = _mm_add_epi8(xmm1, xmm7);
    xmm2 = _mm_shuffle_epi8(xmm8, xmm1);
  // (5) addition of pixels and offsets
    xmm3 = _mm_cvtepi8_epi16(xmm2);
    xmm4 = _mm_cvtepi8_epi16(
             _mm_srli_si128(xmm2, 8));
    xmm0 = _mm_add_epi16(xmm0, xmm3);
    xmm6 = _mm_add_epi16(xmm6, xmm4);
  // (6)save 16 pixels to destination buffer
    xmm0 = _mm_packus_epi16(xmm0, xmm6);
    _mm_store_si128(dstBlk, xmm0);
    srcBlk += 16;
    dstBlk += 16;
  }
  srcBlk -= blkWidth;
  dstBlk -= blkWidth;

  // (7) conditional pixel restoration
  If (left CTU is not available)
    dstBlk[0] = srcBlk[0];
  If (right CTU is not available)
    dstBlk[blkWidth-1] = srcBlk[blkWidth-1];

  srcBlk += srcStride;
  dstBlk += dstStride;
}
```

Fig. 13. SIMD pseudo code of SaoFilterCtb() in the case of EO class 0.

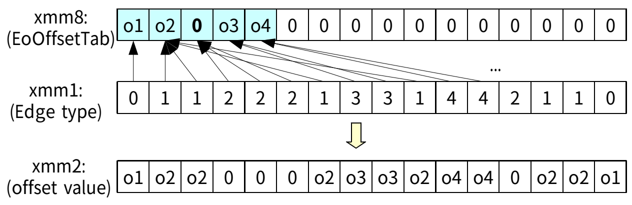


Fig. 14. illustration of xmm2 = _mm _shuffle _epi (xmm8, xmm1) operation in EO filtering.

```
If (iBandStartPos <= 28))
{
  shiftBit = bitDepth - 5;
  blkWidth16 = blkWidth>>4;
  // (1) Prepare new band offset table
  Ofs16[16] = {0};
  Ofs16[0] = BoOffsetTab[iBandStartPos+0];
  Ofs16[1] = BoOffsetTab[iBandStartPos+1];
  Ofs16[2] = BoOffsetTab[iBandStartPos+2];
  Ofs16[3] = BoOffsetTab[iBandStartPos+3];

  xmm7  = _mm_set1_epi16(iBandStartPos);
  xmm6  = _mm_set1_epi16(iBandStartPos+4);
  xmm8  = _mm_load_si128(Ofs16);
  for (y = 0; y < blkHeight; y++) {
    for (x = 0; x < blkWidth16; x++) {
 // (2) load pixels and 16-bit conversion
      xmm0 = _mm_cvtepu8_epi16(*(srcBlk));
      xmm1 = _mm_cvtepu8_epi16(*(srcBlk+8));
  // (3) quantize pixels
      xmm2 = _mm_srli_epi16(xmm0, shiftBit);
      xmm3 = _mm_srli_epi16(xmm1, shiftBit);
  // (4) manipulation of quantized pixels
      xmm4 = _mm_subs_epi16(xmm2, xmm7);
      xmm5 = _mm_subs_epi16(xmm3, xmm7);
      xmm2 = _mm_cmpgt_epi16(xmm2, xmm6);
      xmm3 = _mm_cmpgt_epi16(xmm3, xmm6);
      xmm2 = _mm_or_si128(xmm2, xmm4);
      xmm3 = _mm_or_si128(xmm3, xmm5);
  // (5) table lookup
      xmm2 = _mm_packs_epi16(xmm2, xmm3);
      xmm2 = _mm_shuffle_epi8(xmm8, xmm2);
  // (6) addition of pixels and offsets
      xmm3 = _mm_cvtepi8_epi16(xmm2);
      xmm4 = _mm_cvtepi8_epi16(
               _mm_srli_si128(xmm2, 8));
      xmm0 = _mm_add_epi16(xmm0, xmm3);
      xmm1 = _mm_add_epi16(xmm1, xmm4);
  // (7)save 16 pixels to destination buffer
      xmm0 = _mm_packus_epi16(xmm0, xmm1);
      _mm_store_si128(dstBlk, xmm0);
      srcBlk   += 16;
      dstBlk   += 16;
    }
    srcBlk += (srcStride - blkWidth);
    dstBlk += (dstStride - blkWidth);
  }
}
Else {
  Fall back to default HM code
}
```

Fig. 15. SIMD pseudo code of SaoFilterCtb() in the case of BO.

restoration (i.e., step 7).

Fig. 15 presents BO filtering optimization of Eq. (1) using SIMD instruction. The proposed scheme is composed of seven steps: 1) prepare new band offset table, in which a zero index has a starting band offset, 2) load 16 pixels and convert those to 16-bit pixels, 3) quantize the pixels by using Eq. (2), 4) reduce the quantized pixel range and magnify the unwanted pixel values to over 0x80, 5) table lookup with a new band offset table, 6) add pixels and offset values, and 7) save 16 pixels to the destination

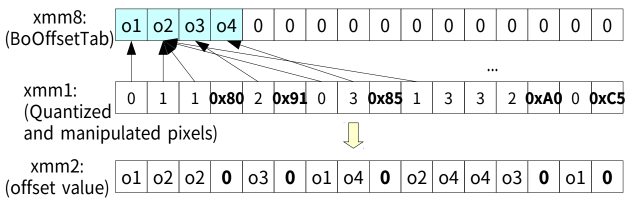

xmm8:
(BoOffsetTab)
o1 o2 o3 o4 0 0 0 0 0 0 0 0 0 0 0 0
xmm1:
(Quantized and manipulated pixels)
0 1 1 0x80 2 0x91 0 3 0x85 1 3 3 2 0xA0 0 0xC5
xmm2:
(offset value)
o1 o2 o2 0 o3 0 o1 o4 0 o2 o4 o4 o3 0 o1 0

Fig. 16. Illustration of xmm2 = _mm_shuffle_epi (xmm8, xmm1) operation in the BO filtering.

buffer. Note that, in step 4, the quantized pixels are manipulated to use the property of the _mm_shuffle_epi() instruction, which returns 0 if a value is greater than or equal to 0x80, as shown in Fig. 16. In Fig. 15, the iBandStartPos represents the first band offset position, which is signaled to a decoder. Note that if the iBandStartPos has a value greater than 28, the band offsets are wrapped around. In such a case, default table lookup codes of HM are used.

4. Performance Evaluation

For the simulation, the proposed algorithm was applied to a sub-optimized Main profile HEVC decoder, which heavily uses SIMD instruction. JCT-VC test sequences were used and encoded by HM12.1. An experiment was performed four times in a row in the workstation[1]. The purpose of the first replication was to load the program code and the bitstream into the disk cache and at least partially into the L2 cache. The median value of the three other replications was reported. We disabled the rendering of the video and writing of the output files, and also subtracted files reading time from total time to minimize the effect of the I/O operations on the execution time. The decoding time was measured using the QueryPerformanceCounter() function, which is the most precise timer function in the system. All the experiments were performed by only exchanging the SAO architecture and SAO SIMD functions in the sub-optimized HEVC decoder.

Table 3 lists the performance of the proposed low-latency CTU-based SAO architecture in the single-core decoding environment. In the table, two streams (People on street, Traffic) were cropped UHD (2560x1600, 30Hz) sequences and the others were Full-HD (1920x1080, 60Hz) sequences. The SAO ratio was calculated using the following Eq. (3):

$$\text{SAO ratio} = (\text{NumSAO\_Y}*4 + \text{NumSAO\_C}*2) / (\text{NumCTU}*6) * 100 \qquad (3)$$

where NumSAO_Y and NumSAO_C represent the number of SAO filtered Y CTB and Cb/Cr CTB of all pictures, respectively. The NumCTU means the number of CTU of all pictures. The dT represents the delta time between PicSAO and CtuSAO calculated using Eq. (4):

[1] Two Zeon E5-2667@2.9GHz (six-core), 16GB DDR3 RAM, and Windows 7 SP1

Table 3. Total decoding speed comparison between picture-based SAO (PicSAO) and the proposed CTU-based SAO (CtuSAO).

Sequences	Bitrate [kbps]	SAO ratio [%]	PicSAO [fps]	CtuSAO [fps]	dT [%]
People on street	8390.2	18.6	14.56	14.56	0.00
Traffic	3054.5	11.5	26.10	25.89	-0.80
BQTerrace	11560.8	32.9	33.40	34.57	0.70
BasketballDrive	23017.9	52.1	22.67	22.58	-0.40
ParkScene	3912.7	9.4	41.28	41.30	0.05
Tennis	9072.7	49.4	38.80	38.69	-0.28
Average	9834.8	28.98	29.62	29.60	-0.12

Table 4. SAO decoding speed comparison between the HM's table lookup SAO and the proposed SIMD SAO.

Sequences	SAO type	CTL [sec]	SIMD [sec]	Gain [%]
BQTerrace	EO class 0	0.389	0.079	392.4
	EO class 1	1.331	0.231	476.2
	EO class 2	0.046	0.027	70.4
	EO class 3	0.257	0.052	394.2
	BO	0.157	0.026	503.8
Tennis	EO class 0	0.152	0.021	623.8
	EO class 1	0.141	0.022	540.9
	EO class 2	0.305	0.038	702.6
	EO class 3	0.186	0.016	1062.5
	BO	0.030	0.007	328.6
Average	-	0.299	0.052	509.5

$$dT = (\text{CtuSAO} - \text{PicSAO}) / \text{PicSAO} * 100 \qquad (4)$$

As shown in Table 3, the proposed low-latency CTU-based SAO architecture has a similar speed to that of the picture-based SAO architecture, in spite of additional backup and copy operations. Note that the decoding speed of Table 3 includes the proposed SAO SIMD scheme.

Table 4 lists the performance of the SAO SIMD scheme for two sequences, where CTL represents the C table lookup codes of HM and SIMD represents the proposed SAO SIMD schemes. The proposed SIMD scheme is approximately 509% faster on average than that of HM.

Table 5 lists the total decoding speed of the decoder with the CTL as well as the proposed SIMD SAO scheme, combined with the proposed low-latency SAO architecture.

In Table 5, the speed-up gain (i.e., dT) is quite different due to the variable SAO ratio. Generally, a high SAO ratio results in a high speed-up. In the total decoding time, the proposed SIMD SAO scheme is faster than the HM SAO scheme by approximately 6.86% on average.

Although the proposed low-latency CTU-based SAO architecture was not tested in the multi-core HEVC decoder, the previous results show the significant decoding speed-up ratio up to 295% using four cores [9].

Table 5. Total decoding speed comparison between HM's table lookup SAO and the proposed SIMD SAO.

Sequences	Bitrate [kbps]	SAO ratio [%]	CTL [fps]	SIMD [fps]	dT [%]
People on street	8390.2	18.6	14.02	14.56	3.85
Traffic	3054.5	11.5	24.91	25.89	3.93
BQTerrace	11560.8	32.9	32.07	34.57	7.80
Basketball Drive	23017.9	52.1	20.92	22.58	7.93
ParkScene	3912.7	9.4	40.10	41.30	2.99
Tennis	9072.7	49.4	33.74	38.69	14.67
Average	9834.8	28.98	27.63	29.60	6.86

Furthermore, the decoding speed-up ratio can be greater than the previous result because of two facts: 1) the final HEVC standard omits the Adaptive Loop Filter (ALF), which increases the inter-frame synchronization latency, 2) the proposed SIMD optimization of SAO filtering can reduce the latency. The proposed algorithm has advantages over a picture-based SAO architecture: 1) significantly less memory required as shown in Table 2, 2) low-latency property enables efficient multi-core decoding. In addition, the proposed SIMD optimization scheme is suitable for the low-latency CTU-based SAO architecture.

The sub-optimized Main profile HEVC decoder with the proposed algorithms passed all the HEVC conformance bitstreams [10] except for those of the Main10 profile and the proprietary 223 test bitstreams including various Tile and slice combinations.

5. Conclusion

This paper proposed a low-latency CTU-based SAO architecture and a SAO filtering optimization scheme by SIMD instructions, which can be used for the realtime decoding of 4K video in a multi-core environment. The proposed architecture showed a similar speed to other existing schemes in single-core decoding environment. The architecture has two advantages over the existing picture-based SAO architecture: 1) significantly less memory requirements, and 2) low-latency property enabling efficient multi-core decoding. In addition, the proposed architecture is suitable for efficient SIMD optimization compared to the existing CTU-based SAO filtering architecture in the SW codec. The proposed SIMD scheme for SAO filtering significantly sped up all SAO filtering classes by approximately 509% on average. Although we simulated the proposed algorithm in the decoder, the proposed methods can also be applied to a HEVC encoder without modification.

Acknowledgement

This study was supported by a grant from the Seoul R&BD program (SS110004M0229111).

References

[1] ITU-T Rec. H.265, *High Efficiency Video Coding*, ITU-T, March 2013. Article (CrossRefLink)

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," *IEEE Trans. CSVT*, Vol. 22, No. 12, pp. 1649-1668, December 2012. Article (CrossRefLink)

[3] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards-including High Efficiency Video Coding (HEVC)," *IEEE Trans. CSVT*, Vol. 22, No. 12, pp. 1669-1684, December 2012. Article (CrossRefLink)

[4] C.-M. Fu, et al, "Sample adaptive offset in the HEVC standard," *IEEE Trans. CSVT*, Vol. 22, No. 12, pp. 1755-1764, December 2012. Article (CrossRefLink)

[5] JCT-VC, HEVC Test Model (HM) reference software 12.1. Article (CrossRefLink)

[6] Parveen.G.B and R. Adireddy, "Analysis and approximation of SAO estimation for CTU-level HEVC encoder," *Proc. Int. Conf. VCIP*, Nov. 2013. Article (CrossRefLink)

[7] P. N. Subramanya, R. Adireddy, and D. Anand, "SAO in CTU decoding loop for HEVC video decoder," *Proc. Int. Conf. Signal Processing and Communication*, December 2013. Article (CrossRefLink)

[8] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2, June 2013. Article (CrossRefLink)

[9] J.-Y. Yi, Y.-H. Kim, J. Park, and J.-W. Kim, "Implementation of HEVC decoder S/W using frame-based multi-threading method," *Proc. ITC-CSCC*, Sapporo, Japan, July 2012.

[10] T. Suzuki, G. Sullivan, and W. Wan, HEVC conformance draft 5, JCTVC-O1004, 15th meeting, Geneva, CH, October 2013. Article (CrossRefLink)

Yonghwan Kim received his B.S. and M.S. degrees in electrical engineering from Chung-Ang University, Seoul, Korea in 1996 and 1998, respectively, and a Ph.D. degree in image engineering from Chung-Ang University, Seoul, Korea, in 2008. From 1999 to 2001, he had worked for SungJin C&C, Seoul, Korea, where he optimized the MPEG-1/2 Video CODEC for DVR. Since 2001 he has worked for Korea Electronics Technology Institute (KETI), Seongnam, Korea. He is currently a managerial researcher in the Multimedia IP Research Center, KETI. His current research interests are in the area of 2D and 3D video coding including HEVC, SHVC, and RGB video coding, and its implementation.

Donghyeok Kim received his B.S. and M.S. degrees in the Department of Information and Communication Engin-eering, Dongguk University in Seoul, South Korea, in 2008 and 2010, respectively. In 2014, he joined the Multimedia IP Research Center of Korea Electronics Technology Institute (KETI), Seongnam, Korea. His research interests include High Efficiency Video Coding (HEVC), filter banks and wavelets, image processing.

Jooyoung Yi received her B.S. and M.S. degrees in electronic engineering from Chonbuk National University, Jeonju, Korea, in 2005 and 2007, respectively. In 2007, she joined the Multimedia IP Research Center of Korea Electronics Technology Institute (KETI), Seongnam, Korea. She is currently involved in the development of video codec, such as HEVC and SHVC.

Jewoo Kim received his B.S. and M.S. degree in control and instrument engineering from the University of Seoul, Seoul, Korea in 1997 and 1999. Since 1999 he has worked for Korea Electronics Technology Institute (KETI), Seongnam, Korea. He is currently a managerial researcher in the Multimedia IP Research Center, KETI. He has been involved in various projects including Multi-view 3D video system, video transcoding system, UHD recording system, etc. His current interests are in the area of UHD broadcasting and realistic media processing including HEVC video coding, UHD contents production system, and audio signal processing, and its implementation.