Hadamard Transform Improvement for HEVC using Intel AVX-512

Jackson Teh Ka Sing[1], Usman Ullah Sheikh[2], Musa Mokji[2], N. Ezaila Alias[2]
Intel Microelectronics (M) Sdn. Bhd., Pulau Pinang, Malaysia.
[2]*School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia,* Johor, Malaysia
jackson.ka.sing.teh@intel.com , usman@fke.utm.my, musa@fke.utm.my, ezaila@utm.my

***Abstract*—High Efficiency Video Coding (HEVC) doubles the data compression ratio compared to previous generation compression technology, Moving Picture Expert Group-Advanced Video Codec (MPEG-AVC/H.264) without sacrificing the image quality. However, this superior compression comes at the cost of more computation payload resulting in longer time for encoding and decoding. This work proposes the vectorization on HEVC data heavy computation algorithm, Hadamard Transform or Sum of Absolute Transform Difference (SATD) and Sum of Absolute Difference (SAD) to achieve optimized compression performance. Single Instruction Multiple Data (SIMD) acceleration will be based on the Intel AVX-512 (Advanced Vector Extension) Instruction Set Architecture (ISA). Since HEVC supports more coding tree block (CTB) sizes, SATD and SAD algorithms eventually become more complex compared to AVC. As a result, SATD and SAD algorithms with various block sizes will be subjected to SIMD acceleration. We provide performance evaluation based on different SIMD ISA and without SIMD implementation on HEVC SATD and SAD and found that AVX-512 optimized implementation performed faster when compared to non-optimized SATD and SAD but showed signs of reduced performance when compared to SSE optimized SATD and SAD.**

Keywords—High efficiency video coding, Intel Advanced Vector Extension, Hadamard transform, compression

I. Introduction

Advanced Video Coding (AVC)/H.264 is the dominant video coding technology used worldwide, while High Efficiency Video Coding (HEVC)/H.265 is the next generation video coding standard [1] that provides significantly improved compression performance relative to AVC standard without sacrificing the quality but with the cost of higher computing and memory requirements as the complexity of processing is increased tremendously.

HEVC video compression standard was a result of joint video project of the same two bodies, ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) standardization organizations. Both working together in a partnership known as the Joint Collaborative Team on Video Coding (JCT-VC) [2]. It is known as ISO/IEC 23008-2 (MPEG-H Part 2) by ISO/IEC, and H.265 recommended by ITU-T. The objective of this joint venture is to achieve higher compression efficiency with roughly 50% of bit-rate reduction without compromising the quality and to have equivalent visual quality compared to AVC (also known as H.264 recommended by ITU-T). The first version of the HEVC verification test showed that the HEVC achieved its goal and performed better on higher resolutions [3].

HEVC implements enhanced tools to improve compression efficiency at the cost of far more computational payload than the capacity of real-time video applications. Therefore, the time consumed in the encoding and decoding would be longer since HEVC supports more coding tree block (CTB) sizes, more transform sizes, an additional loop filter, and more intra prediction angles. All of these improvements over HEVC makes it more complex than its predecessor. Single Instruction Multiple Data (SIMD) instructions had been used to optimize video codec such as H.264/AVC. Thus, HEVC is well suited for SIMD acceleration. However, most SIMD implementations of HEVC are based on older versions of reference software and using SIMD ISA that did not include the Intel AVX-512. Besides, previous implementations did not consider still images, while the optimizations applied on HEVC were limited to using SIMD ISA up to AVX2.

HEVC itself contains support of intra and inter-coding images. Still image coding also known as intra-coding is a natural subset of HEVC codec while inter-coding is applied on image sequences such as burst image and animations. Intra-coding for a video is possible as it only relies on image data in the current frame of a video. As for inter-coding, it relies on image data in one or two reference pictures (before or after the current picture in display order). According to the comparison results with current state-of-the-art coding standards, HEVC is more superior compared with the second best overall performing coding scheme, VP9. On average, HEVC outperformed VP9 by 10% in terms of bit-rate saving, 23% more efficient compared to JPEG2000, 30% compared to JPEGXR, 44% compared to JPEG and 29% compared to WebP [4]. This paper focuses on still image coding on Hadamard transform and Sum of Absolute Difference for the intra-prediction.

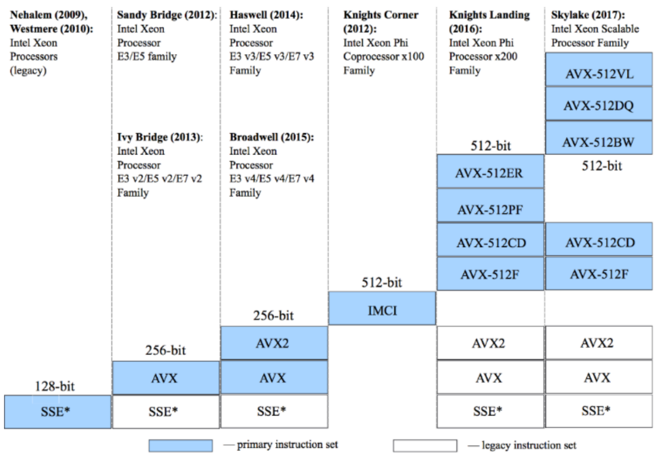
II. Background

A. Intel AVX-512

SIMD is a class of parallel computing in Flynn's taxonomy that takes an operation specified in one instruction and applies it to more than one set of data elements at the same time. The same operation is achieved towards all of the data by partitioning each register into sub-words. Thus, such machine exploits data level parallelism that significantly improves performance as compared to the scalar approach. Furthermore, the performance that can be further improved depends on one of the SIMD properties, the SIMD width. SIMD width is defined as the number of elements that can be processed in parallel within a register. For large SIMD width, fewer SIMD instructions are required that should reduce the processing time for a given data.

Current SIMD with the largest register width is the AVX-512 with 512-bits wide which was proposed by Intel in July 2013. AVX-512 was first only available and supported by Intel Xeon Phi Processor x200 family known as Knight

Landing that was introduced in 2016. In the coming years, Intel then introduced other processor families with AVX-512 support such as the Intel Xeon Scalable Processor, Skylake-X Core i9 and certain i7 families. One of the key features that can be highlighted for AVX-512 is the expansion of SIMD registers from 16 to 32 (`ZMM0-ZMM31`) where the width of the registers is 512 bits compared to previous generation Intel SIMD [5]. Besides, AVX-512 also provides eight opmask registers (`k0-k7`) in which it serves as a predicate operand to mask out the operation on certain bits. However, AVX-512 ISA contains a subset of instructions that are usually grouped by supporting processor generation as shown in Fig.1. This may introduce some inconvenience whenever byte and word operations are needed especially in HEVC which are not found in Knight Landing processor. Syntax representation for the AVX-512 instruction variants is shown in Table I. Intel's current approach on AVX-512 is more targeted on High Performance Computing (HPC), or enterprise workloads since processors with AVX-512 capability are more expensive for the average consumer, with AVX-512 for consumer product will be available in the future.



Nehalem (2009), Westmere (2010): Intel Xeon Processors (legacy)
Sandy Bridge (2012): Intel Xeon Processor E3/E5 family
Ivy Bridge (2013): Intel Xeon Processor E3 v2/E5 v2/E7 v2 Family
Haswell (2014): Intel Xeon Processor E3 v3/E5 v3/E7 v3 Family
Broadwell (2015): Intel Xeon Processor E3 v4/E5 v4/E7 v4 Family
Knights Corner (2012): Intel Xeon Phi Coprocessor x100 Family
Knights Landing (2016): Intel Xeon Phi Processor x200 Family
Skylake (2017): Intel Xeon Scalable Processor Family
128-bit
SSE*
256-bit
AVX
SSE*
256-bit
AVX2
AVX
SSE*
512-bit
IMCI
512-bit
AVX-512ER
AVX-512PF
AVX-512CD
AVX-512F
AVX2
AVX
SSE*
AVX-512VL
AVX-512DQ
AVX-512BW
512-bit
AVX-512CD
AVX-512F
AVX2
AVX
SSE*
— primary instruction set
— legacy instruction set

Fig.1. Current AVX-512 instruction variant sets supported by existing processor architectures [6].

B. Hadamard Transform and Sum of Absolute Difference

Hadamard transform also known as Sum of Absolute Transform Difference (SATD) is one of the cost functions available in HEVC encoder. It is utilized and applied for intra-prediction mode decision and Fractional pixel Motion Estimation (FME). Hadamard matrix is used on the residue block (the difference between the original block and a reference block). Another available cost function in HEVC is the Sum of Absolute Difference (SAD) where SAD is usually applied in the most frequently executed step instead of SATD. The reason behind this is that SATD is more complex compared to SAD and able to achieve better distortion estimation [7]. However, the distortion metric is entirely up to HEVC user's choice. The mode decision of SATD or SAD can be selected depending on the status of the Hadamard Transform flag.

Hadamard Transform and SAD algorithms are located in the `TComRdCost` class in the HEVC HM reference software. Since still image with different resolutions is used as the dataset for performance evaluation, all intra mode will be used throughout the whole HEVC encoding process. Although encoding time distribution for `TComRdCost` class comes in fifth place and does not consume the most time in the encoding process [8], it is still subjected to vectorization.

SATD is computed using Eq.1, and SAD using Eq.2, where $Diff(i,j) = O(i,j) - P(i,j)$, i and j are the pixel indices, and their ranges are determined by a block size. $O(i,j)$ and $P(i,j)$ are the original and predicted pixel values, respectively and T is the transformed coefficients.

$$SATD = \frac{\sum_{i,j}^{I,J} |T.Diff(i,j)|}{2} \quad (1)$$

$$SAD = \frac{\sum_{i,j}^{I,J} |Diff(i,j)|}{2} \quad (2)$$

TABLE I. AVX-512 INSTRUCTION VARIANTS

AVX-512-F	**F** for Foundation.
AVX-512-BW	Support for 512-bit **B**yte and **W**ord (16 and 32-bits) support.
AVX-512-CD	**C**onflict **D**etect (loop vectorization with possible conflicts).
AVX-512-DQ	Instructions for **D**ouble or **Q**uad math operations.
AVX-512-ER	**E**xponential and **R**eciprocal operations.
AVX-512-IFMA	**I**nteger **F**used **M**ultiply **A**dd with 52-bit precision.
AVX-512-PF	**Pref**etch Instructions
AVX-512-VBMI	**V**ector **B**yte **M**anipulation **I**nstructions.
AVX-512-VL	Foundation plus of less than 512-bit **V**ector **L**ength support.
AVX-512-4VNNIW	**V**ector **N**eural **N**etwork **I**nstructions **W**ord (variable precision).
AVX-512-4FMAPS	**F**used **M**ultiply **A**ccumulation **P**acked **S**ingle precision.

III. RELATED WORKS

The work in [9] showed the implementation of SIMD optimization for the entire HEVC decoder for all major SIMD ISA. The authors provided information on several related works that were proposed and reported before them. However, most of the works were only focused on one SIMD ISA and for one resolution and one platform. Thus, the main contribution found [9] is that the authors presented a detailed analysis of SIMD implementation on HEVC decoder by comparing the optimization done on several SIMD ISAs. All the relevant SIMD ISAs used included NEON, and from SSE2 up to AVX2. In [9], SIMD optimization was mainly implemented on the HEVC processing steps which include intra-prediction, inter-prediction, inverse transform, deblocking filter, Sample Adaptive Offset (SAO) filter and various memory operations. The authors also proved that SIMD optimization across all SIMD ISAs performs better with instructions used per frame reduced drastically. Another contribution is the performance evaluation that was conducted on 14 platforms which cover several modern architectures of the last ten years before 2015 for both main (1080p) and main-10 (2160p) profiles. The detailed analysis showed that the instruction per cycles (IPC) using SIMD ISAs is lower compared to the scalar approach. It also showed that AVX2 had the lowest IPC among all SIMD ISAs due to its wider vector properties. However, the work in [9] did not apply AVX-512 ISA.

In [10], the authors focused on the SIMD optimization of HEVC encoder on the Intel x86 processor. First, the authors identify the most time-consuming modules in HEVC encoder and then performed SIMD optimization on the module. The most time-consuming modules include motion compensation, Hadamard transform, SAD calculation, and integer transform. The main contribution is that the authors presented the detailed operation on how the SIMD optimization was performed on each of the time-consuming modules. Even though SIMD optimization was introduced in this paper, SIMD ISAs for AVX/AVX2 and AVX-512 were not utilized where further speed-up can be achieved in the encoder modules. Based on the results observed in this paper, the time saving for motion compensation, Hadamard transform, SAD calculation, and integer transform were 77.6%, 68.4%, 85.1%, and 56.4% respectively. Nevertheless, the results were based on one resolution (720p or HD) test sets only.

This work in [11] proposed using load-balanced slice-level parallelization by allocating the proper number of Coding Tree Units (CTUs) for each core after estimating the computational load for one slice. Incorporation of the above method with SIMD optimization resulted in an approximately ten times the encoding speed compared to the HEVC reference model (HM) software with minimal loss of coding efficiency. Detailed implementations using SIMD for SAD, SATD and inverse transform were provided including the SIMD instructions used to perform the cost 16 function and transform operation. Besides these works, other works include [12, 13] which were based on the ARM NEON SIMD ISA.

IV. Methodology

HEVC encoder Hadamard transform and SAD will be subjected to SIMD optimization using Intel AVX-512 ISA to improve the performance of intra-prediction mode decision. Next is to implement the SIMD optimization on Hadamard transform and SAD through Test Development Approach (TDD). Since AVX-512 is not widely available mainly on consumer laptops or desktops, Intel SDE will be utilized to simulate AVX-512 ISA on any device with an x86 processor. Then, performance evaluation of the HEVC will be conducted with and without SIMD optimization.

A. SAD and SATD in HM Reference Software

Before vectorization, SAD and SATD are identified in the HEVC reference software on how they are utilized and where they are located. `xGetSAD` and `xGetHAD` are the functions to implement the SAD and SATD respectively. Throughout intra prediction in HEVC reference software, SAD or SATD between original and prediction pixels is used to reduce the number of luma intra mode candidates before applying rate-distortion optimization (RDO). Since HEVC adopts additional prediction modes as compared to the previous version, SAD and SATD are more computational intensive which make them perfect subject to vectorization.

B. Software and Hardware Tools

AVX-512 instructions are not widely available on any consumer devices since this ISA is still new and recently introduced by Intel. An emulator provided by Intel called Software Development Emulator (SDE) is utilized to provide emulation of the AVX-512 instructions, especially on the SAD and SATD vectorization development. SDE is a software created by Intel to allow developers who do not own hardware at the time to run their program or test it out before deciding to purchase the processors.

The whole work was developed using C++ based on the HEVC TestModel reference software (HM 16.17) [14] on Visual Studio 2017 which supports over 1,300 intrinsics that corresponds to the AVX-512. Intrinsics are the preferred method since instrinsics are more portable, widely supported by various compilers, and easier to program [15]. In this work, three different C++ compilers were used to compile executable files which are Visual C++ version 15.7, Intel C++ version 18 and G++ version 7.3.0. Both Visual C++ and Intel C++ compilers are setup with Visual Studio on Windows Server 2016 and for G++ compiler on Red Hat 4.8.2-15.

The processor model used on the Windows environment is the Intel Core i9 7960X (Skylake Core X series, 16 cores, 2.8GHz base frequency, 4.2GHz turbo frequency, 22 MB cache, 165W TDP), while the Intel Xeon Phi 7250 (Knight Landing series, 68 cores, 1.4GHz base frequency, 1.6GHz turbo frequency, 32MB cache, 215W TDP) is used for testing under Linux operating system. Both processors offer AVX-512 instructions but with a different subset. The Intel company is providing the Intel Core i9 series used in this work together with the Purley platform (a server platform that incorporates the processor with other components). Whereas the Colfax Research Company, an Intel partner, offered access to the Intel Xeon Phi 7250.

C. Hadamard Transform AVX-512 Optimization

In HEVC, the Hadamard transform is divided into two different stages: the horizontal and vertical stages. CTU in HEVC is usually partitioned into larger than 8×8 block compared to the fixed 16×16 macroblock in AVC. Larger block is divided into several 8×8 blocks during the Hadamard transform operation. Fig.2 shows the overall flow of the SATD operation.

First, the original and the predicted reference pixels are loaded into the AVX-512 registers using `VPGATHERDQ` (Packed GATHER Dword Qword) instruction. `VPGATHERDQ` is used due to the nature of HEVC reference software's access to the pixels in which several pixels will be skipped to perform SATD and depends entirely on the block sizes (largest block size skip the most pixels). Pixel value used in HEVC reference software is 16-bit wide, and this allows as much as 32 pixel values to be loaded into the AVX-512 register at once to perform SATD operation. Then, the difference between the pixels is calculated using SIMD instructions and loaded into registers using `VPSUBSW` (Packed SUBtract signed SaturationWord) instruction. Sign extend is then performed on the result of the difference between original and predicted pixel value with `VPCMPW` (Packed CoMPare Word), `VPBROADCASTW` (Packed BROADCAST Word) as well as `VPUNPCKHWD` (Packed UNPaCK High Word Dword) and `VPUNPCKLWD` (Packed UNPaCK Low Word Dword) instruction. Since sign extension converts 16-bit to 32-bit per pixel value resulting in extra registers to store the results, hence there is not a single instruction that allows performing such an operation. Before going through the transform, sign extended pixel values will have to go through transpose operation. Transpose operation allows shuffling of pixel values within the registers by using twelve `VSHUFF32x4` (SHUFfle Floating-point 32 X 4) and four `VPERMPS` (PERMute Packed Single) instructions. As for the transform operation, addition and subtraction are performed to the pixel values within the registers with the use of six `VPADDD` (Packed ADD Dword), six `VPSUBD` (Packed SUB Dword) and eight `VSHUFI64x2` (SHUFfle Integer 64 X 2).

Both transpose and transform operations are performed twice, as the first is for the horizontal stage and the second is for the vertical stage. After that, absolute pixel values are obtained, and addition is completed across the register before returning the final value to the caller function in the HEVC reference software. SIMD optimization is also performed on block sizes of 4×4 and 2×2. However, due to insufficient pixels to be loaded into the registers, AVX-512 was not utilized in this situation, and SSE was used instead. Besides, utilizing AVX-512 instructions may introduce extra instructions compared to SSE because implementing Hadamard transform for 4×4 and 2×2 block sizes are less complex than 8×8 or larger.

D. Sum of Absolute Difference AVX-512 Optimization

The SAD cost function is less complex than Hadamard transform in HEVC since it does not perform any transformation. Hence, optimization is straight forward and less time consuming but still with some challenges especially when loading data and filling up the registers with pixel values. Fig. 3 presents the overall operation flow for SAD. Original and predicted pixel values are loaded into the respective registers using `VMOVDQU64` (MOVe DoubleQword Unaligned 64) instruction. Difference between both values is then performed followed by getting the absolute using `VPSUBSW` (Packed SUBtract signed Saturation Word) and `VPABSW` (Packed ABSolute Word) instructions respectively. Absolute values are then zero extended using `VPUNPCKHWD` (Packed UNPaCK High Word Dword) and `VPUNPCKLWD` (Packed UNPaCK LowWord Dword) instructions due to double amount of width per pixel value is needed, resulting in extra registers to store the values.

Last but not least, all values within the registers are added through the use of `VPADDD` (Packed ADD Dword) and `VPSUFD` (Packed SHUFfle Dwords) instruction. To cater to the SAD operation with different block sizes is a challenge when loading pixel values to the 512-bits register. There are no available instructions that allow a specific amount of data to be loaded unless the mask is presented in the instruction. Fortunately, most AVX-512 instructions come with a mask feature that allows certain data to be excluded from operation.

E. Performance Evaluation on SATD and SAD with AVX-512, SSE, and Non-optimized

Performance for SIMD optimization can be evaluated in terms of time-saving or speedup gain on the HEVC encoder’s Hadamard transform and SAD modules. Evaluation is done by calculating the time taken to execute the Hadamard transform and SAD. The evaluation will be based on the different set of still images with the range of FHD and UHD resolutions in all intra mode in the configuration file.

In HEVC reference software, the default does not provide timing evaluation. Thus, HEVC reference software was modified to allow timing data collection. The source was also modified in a way that only switches or flags are added in the command line to generate several executable files with different ISA enabled. Different executable files were then created using three distinct compilers which include Intel C++ v18, Visual Studio C++ v15.7 and G++ 7.3.0. Several HEVC executable files were created including non-optimized, with SSE enabled and with AVX-512 enabled with the maximum optimization switch provided (-O2 for Visual C++ and -O3 for both G++ and Intel C++). The test was executed for five encoding processes with the same settings to obtain accurate timing performance.

V. Results

A. FHD HEVC Encoding (1080p 8-bits)

One of the observations that can be seen from Table 2 is that the executable generated from the Intel C++ compiler executes in the least amount of time in finishing the whole encoding process while G++ compiler fared the worst. Intel C++ compiler generally can provide the best performance compared to other compilers because of the ability to perform aggressive microarchitecture optimizations. G++ compiler, on the other hand, has the longest encoding time partly because it could not generate efficient code for the Knight Landing target processor, and that the Knight Landing having a much lower base frequency of 1.4 GHz compared to Skylake Core X processor with 2.80 GHz base frequency.

As for the speedup gain analysis (Table 3), it is noticed that SAD generally has higher speedup gain compared to SATD when using the Intel C++ compiler. Both SSE and AVX-512 speedup gain produced the same result. However, the complete opposite speedup gain can be observed in the Visual C++ compiler, and this applies to both SSE and AVX-512 ISA. G++ compiler, on the other hand, has the same characteristics with Visual C++ on SSE ISA, but noticeable lowest speedup gain occurred in the SATD using AVX-512. Nevertheless, using AVX-512 ISA has a negative impact on speedup gain compared to SSE ISA, and this is true for all C++ compilers.

B. UHD HEVC Encoding (2160p 10-bits)

As for the 4K encoding in HEVC, a similar trend can be observed to FHD encoding results in term of speedup gain for all compilers. However, speedup gain is a little bit higher or improved in both Intel and Visual C++ compiler for SAD. As for SATD speedup gain, improvement is shown in SATD with AVX-512 using Visual C++ compiler but remains almost the same using SSE ISA in both Intel and Visual C++ compilers. The main differences between FHD and UHD encoding results are longer total execution time and significantly lower speedup gain of SATD using Intel C++ compiler. Longer execution time in both SAD and SATD for UHD encoding is due to higher resolution, which will have more CTU broken down to process. As a result, more costly computation is required for UHD 10-bits encoding.

TABLE II. COMPARISON OF EXECUTION TIME FOR SATD AND SAD IN FHD AND UHD ENCODING USING EXECUTABLES GENERATED FROM VISUAL C++, INTEL C++ COMPILER ON WINDOWS SERVER 2016 ON SKL PROCESSOR* AND G++ 7.3.0 ON RED HAT 4.8.2-15 ON KNL PROCESSOR**.

		Visual C++*	Intel C++*	GCC**
FHD	**Non-SIMD SAD**	953.8	848.2	7998.0
	Non-SIMD SATD	2253.4	1290.2	10426.0
	SSE SAD	597.6	492.6	7106.0
	SSE SATD	1321.6	1255.8	7886.0
	AVX512 SAD	685.0	591.4	7140.0
	AVX512 SATD	1584.8	1376.8	21516.0
UHD	**Non-SIMD SAD**	3901.2	3205.2	32724.0
	Non-SIMD SATD	9151.2	6218.2	43662.0
	SSE SAD	2365.6	1815.0	29860.0
	SSE SATD	5414.8	4704.6	33886.0
	AVX512 SAD	2829.8	2809.6	29676.0
	AVX512 SATD	5784.6	7067.0	89438.0

TABLE III. COMPARISON OF SPEEDUP FOR SATD AND SAD IN FHD AND UHD ENCODING USING EXECUTABLES GENERATED FROM VISUAL C++, INTEL C++ COMPILER ON WINDOWS SERVER 2016 ON SKL PROCESSOR* AND G++ 7.3.0 ON RED HAT 4.8.2-15 ON KNL PROCESSOR**.

		Visual C++*	Intel C++*	GCC**
FHD	**SSE SAD**	1.596	1.722	1.126
	SSE SATD	1.705	1.346	1.322
	AVX512 SAD	1.392	1.434	1.120
	AVX512 SATD	1.422	1.228	0.485
UHD	**SSE SAD Speedup**	1.649	1.766	1.096
	SSE SATD	1.690	1.322	1.288
	AVX512 SAD	1.379	1.534	1.103
	AVX512 SATD	1.582	0.880	0.488

VI. DISCUSSION AND CONCLUSION

Several reasons lead to lower speedup gain when using AVX-512 ISA. First, too high latency instructions have a higher tendency to bring up the total overall execution time. This could be seen for `VPGATHERDQ` instruction in AVX-512 whenever it tries to get memory content at the right position in a register. Another example would be in-lane shuffling instructions as seen in the SATD transpose operation. In-lane shuffling is still with lower latency on SKX and KNL. Thus, they cost as much as multiple simpler shuffles found in SSE ISA. Even with the reduction of instructions used compared to SSE ISA, really high latency can be a bottleneck in achieving overall lower total time execution. Besides, dynamic frequency scaling in the Intel architecture whenever AVX or AVX-512 are used could become a factor that lowers the speedup gain. Intel introduced this to keep power in check and avoid power throttling. The problem with such wide instructions is that they consume much power. Imagine a single instruction that does the work of 64 regular byte instructions, or 8 full blown 64-bit instructions. Besides, running AVX-512 even just on one core on CPU will reduce the base frequency to less than 2 GHz while running AVX-512 on all cores will cut it to even lower frequency. Last but not least, SATD speedup gain using G++ compiler was the lowest among other SATD on other compilers and having no improvement at all compared to SAD. The reason is that the processor used to test this was the Knight Landing where byte and word operations (AVX-512BW) are not available compared to Skylake-X. This could lead to extra instructions introduced to the algorithm to achieve the same goal.

In conclusion, speedup gain using AVX-512 on SAD with UHD encoding had a slight improvement over the FHD with 1.5x and 1.4x speedup respectively using the Intel C++ compiler. As for SATD, best speedup gains obtained were 1.42x and 1.52x on FHD and UHD encoding respectively using Visual C++ compiler. However, overall the speedup gain for using AVX512 ISA is lower compared to SSE ISA. Even though speedup gain on AVX512 is much smaller than SSE instructions, speedup gain is still visible when compared to non-optimized code.

ACKNOWLEDGMENT

The authors thank the Ministry of Education Malaysia and Universiti Teknologi Malaysia (UTM) for their support under the Research University Grant (GUP), grant number Q.J130000.2523.19H61.

REFERENCES

[1] Sze, V., Budagavi, M. and Editors, G. J. S. Integrated Circuits and Systems High Efficiency Video Coding (HEVC). ISBN 9783319068947.

[2] Sullivan, G. J., Wang, Y.-k. and Wiegand, T. High Efficiency Video Coding (HEVC) text specification draft 10, 2013.

[3] Sullivan, G. High Efficiency Video Coding (HEVC) and its Extensions H .264 / MPEG-4 Advanced Video High Efficiency Video Coding. 2015. 2(January 2013): 1–10.

[4] Main, H. and Picture, S. Objective Performance Evaluation of the. 2015, 25(5): 790–797.

[5] Cauldron, G. N. U. T. and Yukhin, K. Intel Advanced Vector Extensions Support in GNU Compiler Collection Legal Disclaimer & Optimization Notice. 2015. 2014(July 2014).

[6] Colfax Research. Capabilities of Intel AVX-512 in Intel Xeon Scalable Processors (Skylake), 2017. URL https://colfaxresearch.com/skl-avx512/

[7] Silveira, E., Diniz, C., Beck Fonseca, M. and Costa, E. SATD hardware architecture based on 88 Hadamard Transform for HEVC encoder. Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems, 2016. 2016-March: 576–579.

[8] Swaroop Krishna Rao, Nikita Thakur, S. K. A. HEVC Intra Prediction. 2016. (1001256012): 1–30.

[9] Chi, C. C., Alvarez-Mesa, M., Bross, B., Juurlink, B. and Schierl, T. SIMD acceleration for HEVC decoding. IEEE Transactions on Circuits and Systems for Video Technology, 2015. 25(5): 841–855. ISSN 10518215.

[10] Chen, K., Duan, Y., Yan, L., Sun, J. and Guo, Z. Efficient SIMD optimization of HEVC encoder over X86 processors. Apsipa Asc, 2012. (61071082): 1–4.

[11] Ahn, Y.-J., Hwang, T.-J., Sim, D.-G. and Han, W.-J. Implementation of fast HEVC encoder based on SIMD and data-level parallelism. EURASIP Journal on Image and Video Processing, 2014. 2014(1): 16. ISSN 1687-5281.

[12] Van Dien, N. and Ryu, E. S. Performance comparison of SIMD-based HEVC decoders on mobile processor. Proceedings of KICS-IEEE International Conference on Information and Communications with Samsung LTE and 5G Special Workshop, ICIC 2017, 2017: 298–303.

[13] Bariani, M., Lambruschini, P., Raggio, M., Architectures, N. and Pezzoni, L. An Optimized Software Implementation of the HEVC / H.265 Video Decoder. Consumer Communications and Networking Conf. (CCNC), 2014:83–88.

[14] HEVC Test Model. URL https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/trunk/

[15] Jeffers, J., Reinders, J., Sodani, A., Jeffers, J., Reinders, J. and Sodani, A. Chapter 12 Vectorization with AVX-512 intrinsics. 2nd ed. Elsevier Inc. 2016. ISBN 9780128091944.

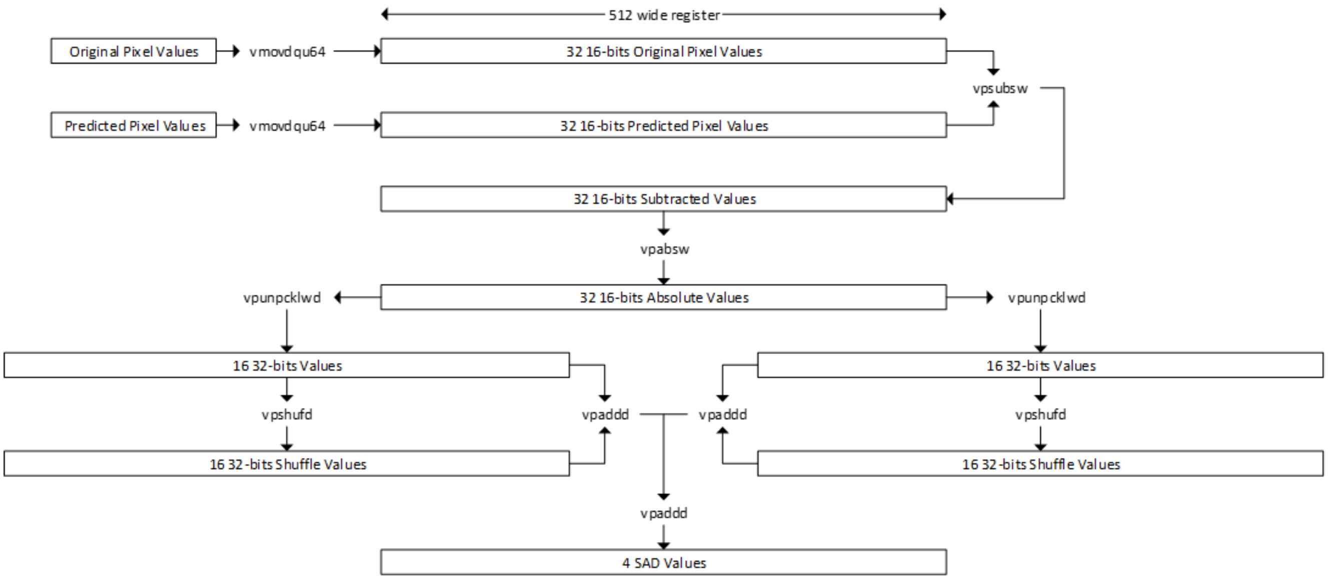


Fig.2. Flowchart for SATD implementation

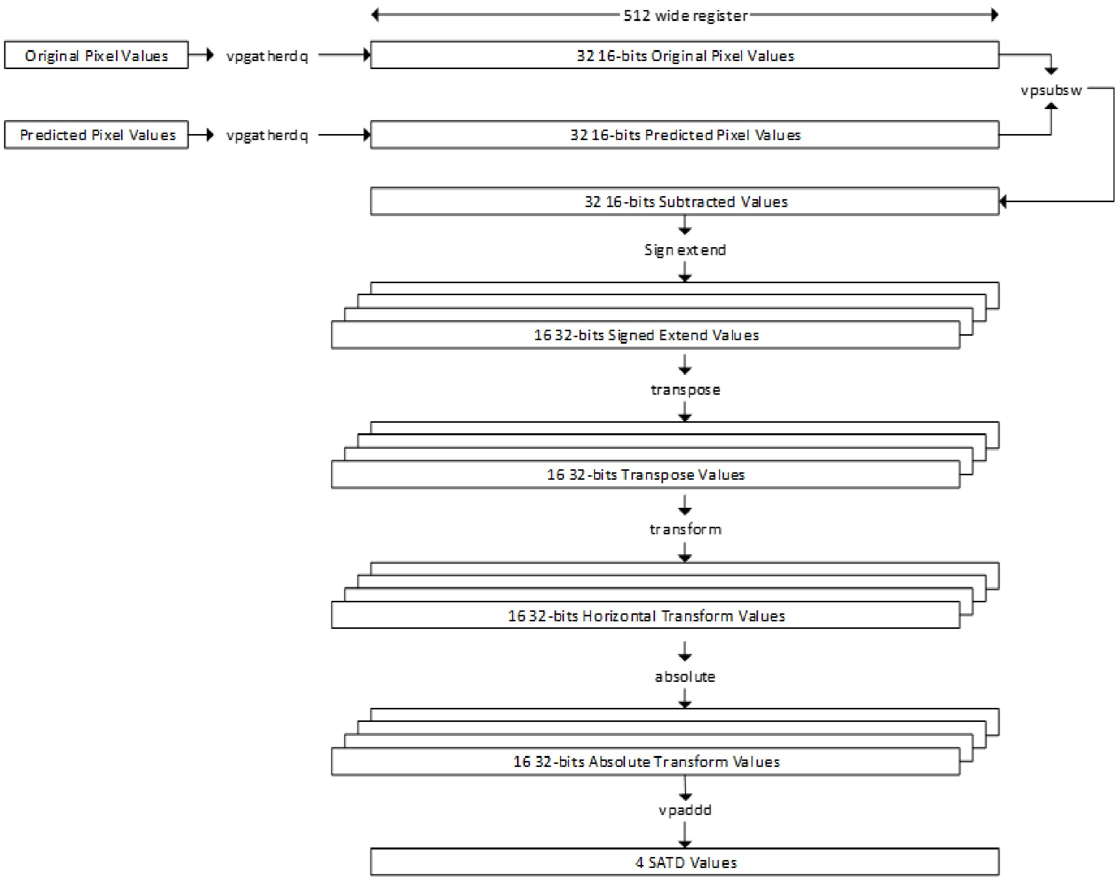

512 wide register
Original Pixel Values
vpgatherdq
32 16-bits Original Pixel Values
vpsubsw
Predicted Pixel Values
vpgatherdq
32 16-bits Predicted Pixel Values
32 16-bits Subtracted Values
Sign extend
16 32-bits Signed Extend Values
transpose
16 32-bits Transpose Values
transform
16 32-bits Horizontal Transform Values
absolute
16 32-bits Absolute Transform Values
vpaddd
4 SATD Values

Fig.3. Flowchart for SAD implementation.