Analysis of the complexity of the HEVC motion estimation

Jakub Stankowski [1], Damian Karwowski, Krzysztof Klimaszewski,
Krzysztof Wegner, Olgierd Stankiewicz, Tomasz Grajek
Poznan University of Technology, Polanka 3, 60-965 Poznań, POLAND
[1] *jstankowski@multimedia.edu.pl*

***Abstract* – The paper presents analysis and comparison of complexity of selected algorithms of motion estimation used in video compression. With the use of own, highly optimized software implementation the authors explored complexity of the methods applied in the framework of the new High Efficiency Video Compression (HEVC) technology. The influence of many different factors on motion estimation complexity has been deeply studied, including the implementation technique, type of the algorithm, number of processor threads used, kind of metric of blocks similarity. The results obtained allowed to formulate guidelines and conclusions that may be useful for future implementation of motion estimation algorithms in context of HEVC encoders.**

Keywords – **Video encoder, HEVC, motion estimation, complexity**

I. INTRODUCTION

The technique of hybrid video compression has revolutionized the way in which we store and transmit digital images. In this technique, the blocks of the currently encoded image are effectively predicted and coded on the basis of other data (derived from the same image - i.e. intra-frame predictive coding, or derived from other neighbouring images – i.e. inter-frame predictive coding) resulting in a very high efficient video compression. Just for this reason, the technique in recent years has been the subject of international standardization and gained a great interest from the market.

The highest achievement in the field of hybrid video compression is recently standardized High Efficiency Video Coding (HEVC) technology[2]. HEVC high efficiency of compression comes for the price of high complexity of a video encoder. It is commonly known that high complexity of an encoder is mainly caused by motion compensated inter-frame prediction, which requires computationally intensive motion estimation process. For every block size of every picture, an encoder has to find optimal motion vector against the number of previously encoded frames in order to choose the best compression mode for a given fragment of an image. Unfortunately, this highly increases the complexity of an encoder[3]. Therefore, necessary works on complexity reduction of an encoder must take place before any market adaptation (of a new coding technology).

Although a number of different algorithms of motion estimation have been developed and tested since the beginning of hybrid video compression development (e.g. Two Dimensional Logarithmic Search[7], Hexagon-based Search[8], Diamond Search[9], Uneven Multi-Hexagon Search[10], Enhanced Predictive Zonal Search[11] or TZ Search[12]) there are very little details about real complexity of those methods applied in the employed to the market encoders. It is especially true when consider those methods in the new HEVC framework.

First of all, relatively poor implementations of the known methods were used in the previously conducted studies i.e. [11][12] and the improvements made to those methods i.e. [13][14]. The implementation (the source code) of the considered methods was usually non-optimized. It did not use the available for over the decade fast SIMD (Single Instruction Multiple Data) vector instructions such as x86 SSE and AVX. Moreover, very often an implementation did not exploit the computational power of modern computers as it has been single-threaded only. Besides, the implementations flaws of methods studied in the literature, there is very little about the impact of the other techniques that accelerate motion estimation process, such as image pre-interpolation technique for calculating motion vectors performed with ½-pel and ¼-pel accuracy.

For the reasons mentioned above, there is a strong need to came back to complexity analysis of the known motion estimation methods. This paper tries to analyse the cause of high complexly of motion estimation algorithms currently known from the literature.

The goal of this paper is to measure and compare the complexity of selected motion estimation algorithms. Because the authors are interested in showing the results from the practical (production) point of view, the selected algorithms were first implemented from scratch with the use of advanced programming techniques such as SIMD or multithread processing, and then compared against each other and optimized version known from reference encoders. All of the work have been done in the context of HEVC.

II. EFFICIENT IMPLEMENTATION OF THE SELECTED METHODS

In order to properly meet the assumed objectives of the works, three different algorithms of motion estimation have been fully implemented from scratch. These are:

- Brute force ‘Full Search’ algorithm – the method which is a reference to other algorithms. This algorithm is particularly easy to optimize.
- ‘Two Dimensional Logarithmic Search’ (hereafter referred to as ‘Log2 search’) – as example of basic “fast” motion estimation algorithm.
- ‘TZ Search’ – as example of commonly recognized state-of-the-art algorithm. The ‘TZ Search’ is used as default algorithm in reference implementations of many video coding

technologies like Scalable and Multiview AVC reference software (JSVM) [4], HEVC reference software (HM) [5] and Joint Exploration Test Model Software for beyond HEVC development (JEM) [6].

Moreover, each of the selected algorithms has been implemented in three different ways:

- First: using plain C++ programming language. The goal was to prepare the algorithmically optimized implementation including efficient memory and resources usage. This implementation makes the starting point for the two next.
- Second: using SSE instructions. In the C++ program code all critical functions (distortion metric calculation, reference picture interpolation, prediction signal calculation, etc.) has been implemented using SSE intrinsic. In this way data level parallel processing (SIMD) offered by most x86 CPUs is exploited.
- Third: using AVX instructions. All SSE instructions that were introduced in the second way of implementation have been substituted by appropriated AVX instructions, which are available in modern x86 CPUs.

In the case of implementations in which SSE and AVX vector instructions were used, some level of acceleration was achieved due to processing of vectors of data within one processor instruction. Additionally, all three versions have been implemented with multithreaded parallel processing in order to examine the impact of the number of active processor cores on duration of motion estimation process. In total, six versions of each of the algorithm have been prepared.

III. Methodology of Experiments

All the implementations that were presented in Section II have been prepared as standalone modules that can be easily put in and out from any encoder implementation. In order to test complexity of the algorithms from the point of view of their implementation in the HEVC video encoder, the software has been specially adjusted in order to take into account the actual conditions in which the motion estimation process is carried out in the encoder. In particular, the following conditions have been included:

- Calculations are carried out for a numerous of block sizes that were defined in the HEVC encoder. These are 64x64, 32x32, 16x16, 8x8, 4x4.
- Starting point for estimation process is calculated on the basis of prediction of a motion vector. Prediction of a motion vector is performed as defined in the HEVC standard.
- Full pel, half-pel and quarter-pel accuracy of motion estimation is considered. In the case of half- and quarter-pel accuracy interpolation of reference pictures is implemented using HEVC interpolation algorithm.
- Both uni- and bi-directional inter-frame prediction is considered.
- In the multithreaded parallel implementation there have been retained limitations of parallel processing of blocks, known from the HEVC ‘wavefront’ scanning scheme.

The study of the complexity have been made on the basis of execution time of the authors implementation on the PC platform equipped with Intel core i7 – 5820K (3.6 GHz – 1 core, 3,4 GHZ – 2 cores, 3.3 GHz – more than 2 cores) and 64 GB of RAM (68 GB/s memory bandwidth).

Base on the collected data, the following research works were done:

- Comparison of complexity of selected motion estimation algorithms (‘Full Search’, ‘Log2 Search’, and ‘TZ Search’).
- Analysis of the impact of implementation technique on complexity of each selected motion estimation algorithms (plain C++, SSE, or AVX).
- Analysis of the impact of the number of processor threads.
- Comparison of complexity of the algorithm when operating with various block similarity metrics (like SSD, SAD).
- Motion estimation complexity reduction with help of image pre-interpolation.

The execution times of the algorithms have been obtained by performing motion estimation for a set of 16 natural video sequences recorded in full HD spatial resolution with 25, 30 and 50 fps. The sequences are characterized by diverse motion with both static and moving background. The partial motion estimation time results received during the experiments were averaged over entire set of sequences.

IV. Complexity of Motion Estimation in the HEVC-Experimental Results and Analysis

A. *Impact of implementation technique on the complexity of motion estimation.*

The experiments were performed in order to evaluate implementation type influence. During the experiments, the TZ search algorithm has been used, and SSD has been set as a distortion metric. In the presented work we focus on x86 processors, therefore, set of 3 optimized implementations has been investigated: plain C++, SSE based and AVX based, with 6 threads working in parallel. The optimized C++ is approximately 10% faster when compared to HM [5] reference software implementation. The results of experiment are shown on Figure 1.

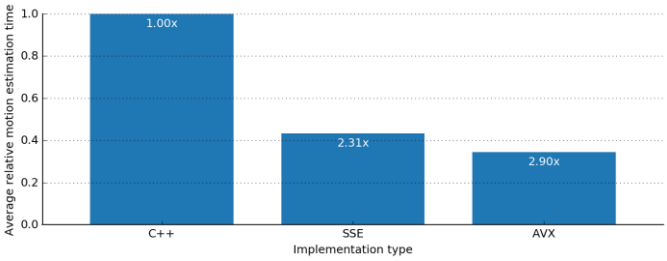

Figure 1. Comparison of the performance of different implementation types (plain C++, SSE and AVX) with speedup over optimized plain C++ implementation. All results gathered for TZ search algorithm and 6 working threads.

The comparison between different implementations clearly shows that the effort spent on optimization of calculations to exploit SIMD instructions is worthwhile. The prepared implementations perform significantly better than the optimized C++ implementation that does not use SIMD instructions. The time reduction is about 57% and 66% (2.31x and 2.90x speedup) for SSE and AVX implementation, respectively.

B. *Motion estimation algorithm and its complexity*

It is obvious that Full Search (FS) algorithm has the highest computational complexity measured as a time of motion estimation. The next experiment was to check the improvement in time performance offered by fast algorithms (Log2 and TZ search) for a fully optimized production quality encoder. The performance

ratios between Full Search and fast algorithms may be influenced by using the best performing (according to the previously described experiment) AVX instruction set. The test scenario assumes the use of multithreaded, AVX optimized implementation and 6 working threads. The results of experiment are shown in Figure 2.

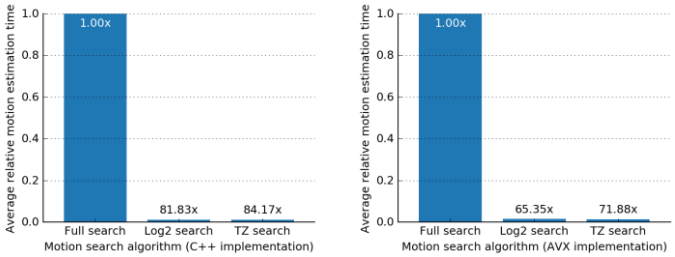

Figure 2. Comparison of search algorithms complexity (measured as relative motion estimation time) with speedup over Full search. All results gathered for SSD distortion metric and 6 working threads.

It is obvious from the results that even for the SIMD optimized methods there is a significant difference in performance between full search and fast search algorithms, although the difference is smaller than for a C++ implementation without SIMD instructions (TZ search is 84 times faster than full search when not using SIMD and 72 times faster than full search when both are using SIMD implementation).

C. *The number of processor threads and its impact on the complexity of motion estimation.*

The next experiment concerned the influence of the number of threads onto the performance of motion estimation. Similarly as before, the best preforming method, namely, an AVX optimized TZ search motion estimation, was chosen for the test. The number of parallel working threads was adjusted in the range from 1 to 12 threads. The results are shown on Figure 3. It is important to note, that all the tests were performed on a computer with a 6 core (included HyperThreading mechanism – 12 threads) processor. The results clearly show that, as long as the number of threads does not exceed the number of cores multiplied by 2, the performance increase stays approximately proportional to the number of threads used (4 threads perform almost 4 times better than a single thread). The increase is not perfectly proportional, most probably due to a reduced CPU frequency when using more than 1 core.

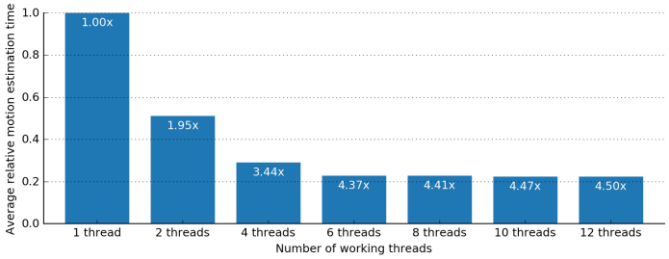

Figure 3. Comparison of different number of working threads on 6 core (with HyperThreading) CPU with speedup over singlethreaded mode. All results gatheres for AVX optimized implementation, TZ search algorithm and SSD distortion metric.

Increasing the number of working threads over the number of actual cores in the processor does not improve the performance. In fact, the performance for threads number exceeding the number of cores leads to a slight (but, nonetheless, noticeable) decrease of the performance of the coder. Also – the increased number of threads increases the required transfer from memory, and this one stays the same, independently of the number of cores used. This forms the most important bottleneck for a multithreaded coder.

It is therefore advised to set the number of threads to match twice the number of cores used in the actual machine that performs the compression.

D. *Metric of similarity of image blocks and its influence on complexity of motion estimation.*

Another important factor influencing the complexity of an encoder is the type of the metric used during motion estimation. The metric is used to express the similarity between two blocks of images – from a current and from a reference image. The most commonly used is SAD – Sum of Absolute Differences, since the SSD – the Sum of Squared Differences can be, due to the presence of the squaring phase, perceived as more computationally expensive. For more precise estimation, like ½- and ¼-pel accuracy motion estimation, the SATD (Sum of Absolute Transform Differences) is used [4][5][6]. This is much more complex, since it involves computation of the transform of a block. The obtained results are shown in Figure 4. The numbers represent the performance ratio when compared to the SAD full pel + SAD fractional pel metric scenario. The lower the bars, the better the performance. It is clearly shown, that with the use of AVX instructions, the SAD and SSD metrics are identical in terms of computational complexity. The same is not true for a simple C++ implementation without SIMD, where the SSD requires approximately 53% more time to calculate. What is more, the use of AVX instructions flattens the performance graph – the most complex combination of metrics, the SSTD full pel + SSTD fractional pel, needs 345% of time needed for SAD + SAD, while for simple C++ implementation it requires 435% of time. Take into account the fact that AVX performs overall approximately 3 times better for SAD itself (as shown in part IV A) and the differences become even more appreciable.

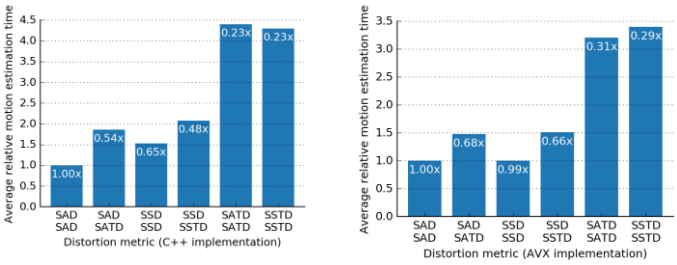

Figure 4. Comparison of different distortion metrics for Full pel / Half- and Quarter-pel block position. Speddup calculated with SAD/SAD as reference. All results gathered for TZ search algorithm and and 6 working threads.

E. *Pre-interpolation of reference pictures and influence of the technique on complexity of motion estimation.*

The motion estimation with ½- and ¼-pel accuracy requires additional effort caused by performing the reference picture interpolation. When compared to AVC [1], HEVC interpolation filters are much more complex and almost double the number of required operations (memory bandwidth, additions and multiplications) to calculate interpolated value [15]. Therefore, the reference picture pre-interpolation and buffering technique described in [16] has been evaluated. In case of AVC the best

tradeoff between complexity and memory usage was to store ½-pel interpolated images and perform ¼-pel interpolation (simple bilinear filter, based on full-pel and ½-pel samples) on-the-fly. In case of HEVC both ½- and ¼-pel blocks are interpolated based on full-pel samples. Moreover, 9 of 15 possible sample positions require two interpolation passes – first horizontal pass and second vertical pass. Due to the increased complexity of interpolation stage, the decision was made to evaluate 4 different scenarios: no pre-interpolation (all calculations are done on-the-fly), pre-interpolation for ½-pel samples, pre-interpolation for ¼-pel samples and pre-interpolation for horizontal samples (to avoid the need of 2 interpolation passes). Only the best performing AVX scenario is considered here. All result have been shown in Figure 5.

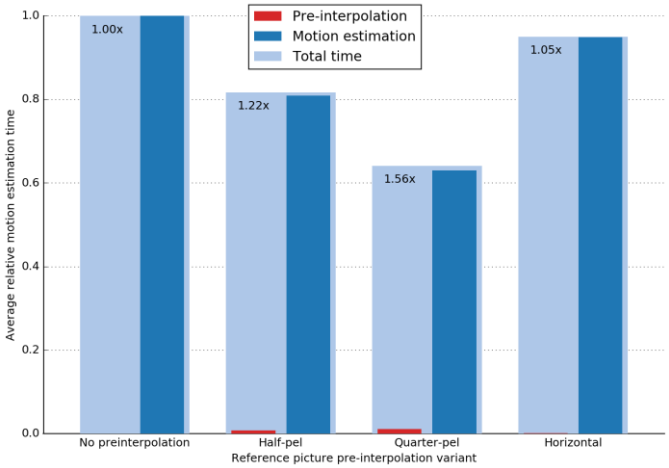

Figure 5. Comparison of pre-interpolation approaches with speedup calculated over on-the-tly (no pre-interpolation) sheme. All results gathered for AVX optimized implementation, SSD distortion metric, TZ search algorithm and and 6 working threads.

The experiments clearly show that buffering of pre-interpolated reference images allows a substantial speedup in the HEVC motion estimation stage. The pre-interpolation for all positions (Quarter-pel) offers best performance (1.56 times faster than on-the-fly) but requires to buffer additional 15 images (thus, requiring 16 time more memory). The ½-pel only and horizontal only pre-interpolation cases require to store only 3 additional pictures but offer lower speedup. Especially the horizontal pre-interpolation case is not worth of use when compared to ½-pel pre-interpolation.

V. Conclusions and Final Remarks

The results shown in this paper support clearly the following conclusions. Firstly, the performance of the encoder can be significantly increased by speeding up the motion estimation phase (the most time consuming one) with the use of SIMD (Single Instruction Multiple Data) instructions. The best performing implementation obtained in the research was making use of the AVX instruction set. The expected increase of the performance for the motion estimation phase for the most frequently used scenario is 3 times. More importantly, the AVX implementation equalizes the performance of SSD and SAD metrics used in motion compensation, so any of them can be selected, without any performance penalty.

Acknowledgment

Research project was supported by The National Centre for Research and Development, POLAND, Grant no. LIDER/023/541/L-4/12/NCBR/2013.

References

[1] ISO/IEC 14496-10, Int. Standard “Generic coding of audio-visual objects – Part 10: Advanced Video Coding” 7th Ed., 2012, also: ITU-T Rec. H.264, Edition 7.0, 2012.

[2] ISO/IEC 23008-2 (MPEG-H Part 2) / ITU-T Rec. H.265: High Eciency Video Coding (HEVC), Apr. 2013.

[3] F. Bossen, B. Bross, K. Suhring, D. Flyn, HEVC Complexity and Implementation Analysis, IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, December 2012.

[4] H. Schwarz , M. Wien and J. Vieron, JSVM software manual, Doc. JVT-S070, 2006

[5] F. Bossen, D. Flynn, K. Sharman, K. Sühring, “HM Software Manual”, Joint Collaborative Team on Video Coding of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Document: JCTVC-Software Manual

[6] Joint Exploration Test Model Software, Joint Video Exploration Team, of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, https://vceg.hhi.fraunhofer.de/svn/svn_HMJEMSoftware/tags/HM-16.6-JEM-1.0/

[7] J. Jain and A. Jain, "Displacement Measurement and Its Application in Interframe Image Coding," in IEEE Transactions on Communications, vol. 29, no. 12, pp. 1799-1808, Dec. 1981.

[8] Ce Zhu, Xiao Lin, and Lap-Pui Chau, Hexagon-based search patten for fast block motion estimation, IEEE Trans. Circuits Syst. Video Technol., 12(5) (2002) 349–355.

[9] S. Zhu, K.-K. Ma, A new diamond search algorithm for fast block matching motion estimation, in: Proceedings of the International Conference on Information and Communication and Signal Processing, Singapore, Sept., 1997, pp. 292–296.

[10] Peng Yin, Hye-Yeon Cheong Tourapis, Alexis Michael Tourapis, Jill Boyce, Fast mode decision and motion estimation for JVT/H.264, in: Proceedings of the IEEE International Conference on Image Processing 2003, September 2003, vol. 2, pp. 853–856.

[11] A.M. Tourapis, Enhanced predictive zonal search for single and multiple frame motion estimation, in: SPIE Proceedings of the Visual Comm. Image Proc., 2002.

[12] Xiu-li T., Univ H., Sheng-kui D., An analysis of TZSearch algorithm in JMVC, International Conference on Green Circuits and Systems (ICGCS), Shanghai, 21-23 June 2010, s. 516 – 520.

[13] Xufeng Li, Ronggang Wang, Wenmin Wang, Zhenyu Wang and Shengfu Dong, "Fast motion estimation methods for HEVC," Broadband Multimedia Systems and Broadcasting (BMSB), 2014 IEEE International Symposium on, Beijing, 2014, pp. 1-4.

[14] N. Purnachand, L. N. Alves and A. Navarro, "Fast Motion Estimation Algorithm for HEVC," Consumer Electronics - Berlin (ICCE-Berlin), 2012 IEEE International Conference on, Berlin, 2012, pp. 34-37.

[15] Hao Lv, Ronggang Wang, Xiaodong Xie, Huizhu Jia and Wen Gao, "A comparison of fractional-pel interpolation filters in HEVC and H.264/AVC," *Visual Communications and Image Processing (VCIP), 2012 IEEE*, San Diego, CA, 2012, pp. 1-6.

[16] T. Grajek, D. Karwowski, A. Łuczak, S. Maćkowiak, M. Domański: „Architecture of Algorithmically Optimized MPEG-4 AVC/H.264 Video Encoder”, LNCS 7594, pp. 79-86, Springer, Heidelberg (2012)