HARDWARE-AWARE MOTION ESTIMATION SEARCH ALGORITHM DEVELOPMENT FOR HIGH-EFFICIENCY VIDEO CODING (HEVC) STANDARD

Mahmut E. Sinangil[*]
Anantha P. Chandrakasan
Massachusetts Institute of Technology
Cambridge MA

Vivienne Sze
Minhua Zhou
Texas Instruments Inc.
Dallas TX

ABSTRACT

This work presents a hardware-aware search algorithm for HEVC motion estimation. Implications of several decisions in search algorithm are considered with respect to their hardware implementation costs (in terms of area and bandwidth). Proposed algorithm provides 3X logic area in integer motion estimation, 16% on-chip reference buffer area and 47X maximum off-chip bandwidth savings when compared to HM-3.0 fast search algorithm.

Index Terms— HEVC, motion estimation, hardware-aware, motion search algorithm

1. INTRODUCTION

High-Efficiency Video Coding (HEVC) is a new video compression standard being standardized by the JCT-VC (joint collaborative team on video coding) established by ISO/IEO MPEG and ITU-T [1]. HEVC has a design goal of achieving 50% coding gain over AVC/H.264 High Profile. For this purpose, several coding efficiency enhancement tools have been proposed to this new standard.

Motion estimation is one of the most critical blocks in video encoding. Search algorithm performed in motion estimation determines coding efficiency as well as power consumption through number of computations and on-chip and off-chip bandwidths. Searching among a larger number of candidates leads to finding better motion vectors and improves coding efficiency. However, larger number of candidates often results in more switching activity and data bandwidth and hence more active power consumption. Moreover, for a fixed cycle budget, searching through a larger number of candidates require a higher level of parallelism and results in larger hardware area. It is also important to note that searching strategy can have multiple stages where the output of the previous stage can affect where and how the next stages will be performed. This can also limit the throughput for a fixed cycle budget and is not desirable for hardware implementation. Hence, search algorithm development should consider its coding efficiency impact as well as its implications on hardware implementation.

Among many new tools in HEVC, coding quad-tree structure is designed for providing a modular coding structure. In HEVC, a frame is divided into LCUs (largest coding units) and an LCU is further divided into CUs (coding units) in a quad-tree structure. Currently, LCU size can be as large as 64x64 pixels and SCU (smallest coding unit) size can be as small as 8x8. If a CU is not divided into smaller CUs, it is predicted with one of several PU (prediction unit) types.

For inter prediction, PU types can be 2Nx2N, 2NxN, Nx2N or NxN where 2Nx2N corresponds to the size of the CU. If asymmetric motion partitions (AMP) are used, non-square PUs for inter also include 2NxnU, 2NxnD, nLx2N and nRx2N. NxN is only allowed at the SCU level not to present redundancy.

This paper presents a HEVC motion estimation search algorithm suitable for hardware implementation. First, a search pattern consisting of two parallel stages is implemented to enable parallel evaluation of candidates. Secondly, a scheme allowing a single shared search range across parallel engines is proposed to reduce on- and off-chip bandwidth. Lastly, a pre-fetch scheme is implemented to offset the latency of the off-chip memory holding reference frame data. The rest of this paper is organized as follows: Section 2 provides an overview of the motion estimation architecture suitable for HEVC and its quad-tree structure. Section 3 presents the parallelized search strategy and its implementation. Section 4 focuses on the search range sharing between engines and section 5 talks about the pre-fetching strategy to offset latency of the off-chip memory. Finally, section 6 summarizes the benefits of the proposed search algorithm in terms of hardware area and data bandwidth and the cost in terms of coding loss.

2. OVERVIEW OF HEVC MOTION ESTIMATION ARCHITECTURE

For maximum coding efficiency in motion estimation, all the block sizes (CUs and PUs in CUs) should undergo independent motion searches sequentially as implemented in the HEVC test model (HM) [2]. In hardware implementation, however, various simplifications have been discussed

[*]Funding provided by Texas Instruments Inc.

for previous standards to allow search range and cost calculations to be shared between different block sizes [3, 4, 5]. These changes cause motion vector predictor calculations to be based on estimations rather than correct motion vectors and cost coding efficiency degradation.

In this work, an architecture suitable for HEVC motion estimation allowing sequential processing of blocks is considered to provide highest coding efficiency. Search algorithm development in this work focuses on reducing number of computations and data bandwidth. However, proposed modifications do not require a change in sequential processing of blocks and AMVP (advanced motion vector predictor) calculations continue to be exact.

Since cost calculations and motion search for different blocks are separate and independent, separate and independent engines for different block sizes are necessary in implementation. Then, to find the best combination of CU sizes and PU types, output of each engine can be combined to make size and type decisions starting at smallest CU and then going to larger CU sizes. Fig. 1 shows the architecture of the HEVC motion estimation module considered in this work. It consists of four parallel CU engines supporting only 2Nx2N PU type. Hence, supported block sizes are 64x64, 32x32, 16x16 and 8x8. This architecture can be generalized to support different PU types and consequently different block sizes as well.

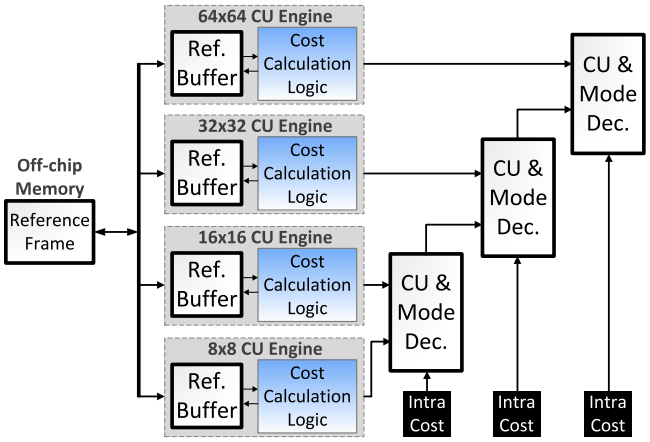


Fig. 1. Architecture of the HEVC motion estimation module considered in this work. Four parallel CU engines from 64x64 down to 8x8 are included with only 2Nx2N PU type.

3. SEARCH STRATEGY

Fast search strategy used in HM-3.0 starts the search around the best AMVP and involves many inter-dependent stages in integer motion estimation (IME). For example, the result of the initial diamond search determines if a sub-sampled raster search is performed or not. In hardware implementation, this dependency increases complexity and often results in extra cycles or extra hardware to account for the worst-case conditions. For, fractional motion estimation (FME) refinement, best integer motion vector is taken as search center and then half and quarter pixel refinements are performed respectively.

Recent work focused on search algorithms that can be parallelized in hardware implementation [6, 7]. In this work, a two-stage search strategy is used for IME where each stage can be independently performed in parallel. Fig. 2 shows IME search patterns used in each of the stages. First, search center is decided by comparing AMVP list entries (up to three entries) and [0,0]. During this comparison, SAD (sum of absolute differences) cost is used. After search center is determined, two stage search is started.

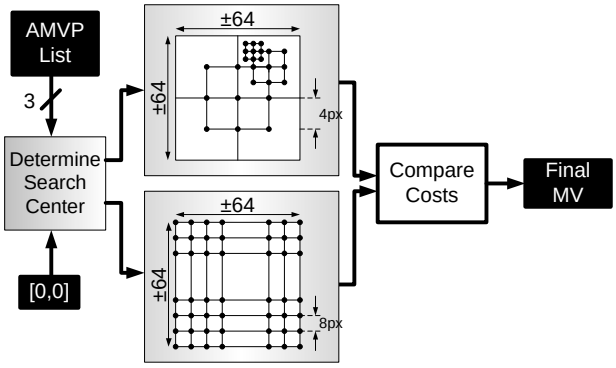


Fig. 2. Two stage search approach used in this work. Stages are independent of each other and can be performed in parallel in hardware.

The first stage consists of a coarse search covering ± 64 by checking every 8th candidate in each direction. This stage can capture a change in motion or irregular motion patterns that cannot be tracked by AMVP. The second stage performs a more localized three step search around the ± 7 window of the search center. This stage can capture regular motion. It is important to note that the AMVP calculation for all blocks uses exact motion vectors of the neighbors and AMVP is accurate and hence can track motion well in most cases.

The proposed IME search strategy checks a total of 285 candidates for each block as opposed to up to 850 candidates that are checked in fast search strategy in HM-3.0. This results in roughly 3X hardware area reduction in IME for the same throughput constraint. Actual savings might be larger in implementation because of the additional complexity due to inter-dependent stages of HM-3.0 algorithm.

Lastly, for FME, search strategy of HM is used where refinement is performed around the best integer motion vector.

	Separate Buf. (4 x 1R1W)	Shared Buf. (1 x 4R1W)
Memory Size	109.2KB	39KB
Est. Array Area	$0.75mm^2$	$0.61mm^2$
Est. Perip. Area	$0.5mm^2$	$0.44mm^2$
Est. Total Area	$1.25mm^2$	$1.05mm^2$

Table 1. Area comparison of shared and separate reference buffers. Estimates are based on a generic 65nm CMOS technology.

Block Size	Max. Off-Chip BW	Avg. Off-Chip BW
64x64	2.2GB/s	1.49GB/s
32x32	6.4GB/s	3.64GB/s
16x16	20.9GB/s	7.62GB/s
8x8	75.6GB/s	17.47GB/s
Total	105.1GB/s	30.22GB/s

Table 2. Maximum and average off-chip bandwidth requirement for different block sizes (search range is ± 64).

4. SHARED SEARCH RANGE ACROSS PARALLEL ENGINES

As explained in Section 2, for maximum coding efficiency, each engine in Fig. 1 is running independently and has separate reference buffers holding pixels for independent search windows. This approach is expensive in terms of area and external memory bandwidth. Sharing the on-chip reference buffer across parallel engines can be significantly more efficient for practical implementations. This section will focus on benefits and cost of sharing search range across parallel engines.

4.1. Area Impact of Reference Buffers

In the case of separate reference buffers with ± 64 search range for each engine, the implementation in Fig. 1 requires four 1R1W port memories with 39KB, 27.5KB, 22.5KB and 20.2KB sizes for 64x64, 32x32, 16x16 and 8x8 engines respectively. The sizes of these memories are determined by the block size, search range and the extra pixels at the edges of the search range for FME. In practical implementations, a larger size for each buffer might be used to allow data sharing between consecutive blocks. Total area consumed by these four memories will be roughly $1.25mm^2$ in a 65nm CMOS technology as shown in Table 1. It should be noted that this area is for storing the pixels on the chip for a single direction and single reference frame.

In the case of a shared reference buffer with ± 64 search range, the size is determined by the largest block size. In this case a single 39KB memory is needed with 4R1W ports. Although the bit-cell area and some peripheral components need to be expanded to support multiple read ports, the overall area can be smaller as shown in Table 1. Hence, shared search range across parallel engines result in 16% area savings for the implementation considered in Fig. 1.

4.2. Data Bandwidth Impact of Reference Buffers

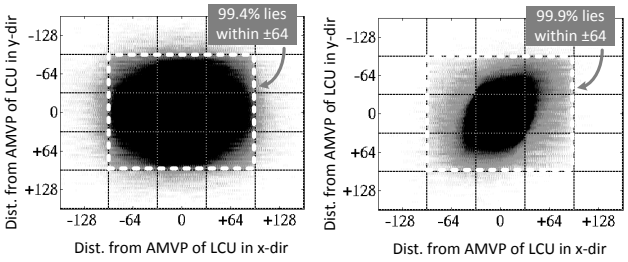
With independent motion searches, each engine might have different search centers and consequently access different parts of the reference frame as the search window. Table 2 shows maximum and average off-chip bandwidth for 64x64, 32x32, 16x16 and 8x8 engines. The upper limit on the bandwidth is calculated by assuming that the entire on-chip reference buffer needs to be updated between consecutive blocks and hence no data re-use is possible. The total maximum off-chip bandwidth is 112 GB/s for supporting 4Kx2K resolution at 30fps assuming a search range of ± 64. Average bandwidth number assumes close to 100% data re-use between consecutive LCUs. However, it should be noted that 100% data re-use is not possible due to sequential processing of the blocks.

In the case of a shared reference window across engines, the maximum bandwidth is equal to the maximum bandwidth of the 64x64 block since the size of the shared search window is determined by the largest block size. Hence, sharing the search window provides 47X and 20X savings in terms of the maximum and average bandwidth requirements.

4.3. Strategy for Sharing Search Window

In order to minimize the coding efficiency impact of sharing search window across engines, a good representative should be selected for the motion of the LCU and CUs within the LCU. AMVP of the LCU is observed to provide a good center point for the shared search window. Fig. 3 shows the density map for the relative location of the pixels from best matching blocks with respect to the AMVP of the LCU for two different sequences. Best matching blocks are calculated with the original HM-3.0 search algorithm and the search range is ± 64 pixels in each direction. For both sequences, more than 99% of the best matching pixels lie in the ± 64 vicinity of the AMVP of the LCU. This indicates that AMVP of the LCU can be used as the search window center without introducing significant coding efficiency loss.

For smaller blocks that have different AMVPs and consequently different search centers, the search window is modified to fit in the shared window. It is important to note that although the search window is modified, original AMVP of the block is used in cost calculations. Moreover, total number of candidates for smaller blocks stay the same regardless of the search window being modified or not. This provides simplicity in hardware implementation.



(a) PeopleOnStreet (b) Traffic

Fig. 3. Density maps for the relative location of pixels from best-matching blocks with respect to the AMVP of the LCU. More than 99% of the pixels lie within ± 64 of the AMVP of the LCU (2560x1600 sequences with QP=22 in random-access configuration).

5. PRE-FETCHING STRATEGY

For a practical hardware implementation, it is necessary to request the data from off-chip memories in advance as the latency of these memories can be on the order of thousands of cycles. To address this, a pre-fetching strategy is developed for the search algorithm proposed in this work.

AMVP of the LCU is used to open the shared search range. However, AMVP calculation for current LCU depends on its left neighbor's motion data and cannot start until left neighbor's motion search is finalized. In this work, top (T), top-right (TR) and top-left (TL) neighbors of the current LCU are used to predict current LCU's AMVP and pre-fetch corresponding data from the off-chip memory. The procedure to predict current LCU's AMVP is as follows:

- If none of the neighbors is available, data is pre-fetched from [0,0] location.
- If only one of the neighbors is available, AMVP of the available neighboring LCU is used to pre-fetch data.
- If two of the neighbors are available, AMVP of one of the available neighboring LCUs is used to pre-fetch data in the following precedence order: $T > TR > TL$
- If all neighbors are available, median of the AMVPs of three neighbors is calculated and used to pre-fetch data.

With this strategy, request can be sent to an off-chip memory as soon as the motion search of *TR* neighbor is completed.

6. SIMULATION RESULTS

Proposed changes in search algorithm are implemented in HM-3.0 and simulations are performed under common test conditions defined by JCT-VC [1]. Table 3 shows coding efficiency change with respect to the HM-3.0 fast search algorithm after each of the algorithm modifications proposed in this work. Columns *LD*, *LDP* and *RA* stands for low-delay, low-delay with P and random-access test conditions as defined by JCT-VC [1]. *Avg* column is the average of *LD*, *LDP* and *RA*. Lastly, *Max* and *Min* columns are the maximum and minimum rate change for all tested sequences respectively.

	LD	LDP	RA	Avg	Max	Min
HM-3.0 Anchor	0	0	0	0	0	0
Prop. Search	0.6	0.8	1.6	1.0	3.1	0.1
Prop. Search & Shared Window	0.6	1.0	2.9	1.5	7.4	0.2
Prop. Search & Shared Window & Pre-fetch	0.9	1.0	2.9	1.6	7.3	0.2

Table 3. Simulation results of the coding efficiency change with the proposed changes with respect to HM-3.0.

The average rate increase due to the proposed changes is 1.6%. Random-access test conditions result in the largest coding efficiency degradation as the distance between the reference frame and the coded frame is longer.

7. SUMMARY AND CONCLUSIONS

This work presents a search algorithm for HEVC motion estimation. Search patterns, shared search window and pre-fetch schemes are designed to minimize hardware costs. Proposed changes provide 3X IME logic area, 16% on-chip memory area and 47X maximum off-chip bandwidth savings at the expense of 1.6% average bit-rate increase when compared to HM-3.0 fast search algorithm.

8. REFERENCES

[1] "Joint Call for Proposals on Video Compression Technology," ITU-T SG16/Q6, 39th VCEG Meeting: Kyoto, 17-22 Jan. 2010, Doc. VCEG-AM91.

[2] "JCT-VC Reference Software HM-3.0," ISO/IEO MPEG and ITU-T.

[3] Yu-Wen Huang et al., "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264," in *ISCAS*, May 2003, vol. 2, pp. II–796 – II–799.

[4] Swee Yeow Yap et al., "A VLSI Architecture for Variable Block Size Video Motion Estimation," *Circuits and Systems II: Express Briefs, IEEE Tran. on*, vol. 51, no. 7, pp. 384 – 389, July 2004.

[5] M. Haller et al., "Robust global motion estimation using motion vectors of variable size blocks and automatic motion model selection," in *IEEE ICIP*, Sept. 2010, pp. 737 –740.

[6] Chia-Chun Lin et al., "PMRME: A Parallel Multi-Resolution Motion Estimation Algorithm and Architecture for HDTV Sized H.264 Video Coding," in *IEEE ICASSP*, April 2007, vol. 2, pp. II–385 –II–388.

[7] Hsiu-Cheng Chang et al., "A 7mW-to-183mW Dynamic Quality-Scalable H.264 Video Encoder Chip," in *IEEE ISSCC*, Feb. 2007, pp. 280 –603.