A 1.59Gpixel/s Motion Estimation Processor with -211-to-211 Search Range for UHDTV Video Encoder

Dajiang Zhou, Member, IEEE,
Jinjia Zhou, Member, IEEE,
Gang He,
and Satoshi Goto, Life Fellow, IEEE

Abstract

3840×2160 and 7680×4320 UHDTV formats deliver remarkably enhanced visual experience relative to high definition but in the meanwhile involve huge complexity and memory bandwidth requirements in video encoding. Especially, enlarged motion distances of UHDTV lead to additional difficulties in the implementation of motion estimation, which is originally the most critical bottleneck of an encoder. This paper presents a motion estimation processor design for H.264/AVC. A test chip is implemented in 40nm CMOS. With algorithm and architecture co-optimization, the processor delivers a maximum throughput of 1.59Gpixel/s for 7680×4320 48fps video, at least 7.5 times faster than previous designs. The corresponding core power dissipation is 622mW at 210MHz, with energy efficiency improved by at least 23%. The chip’s DRAM bandwidth requirement is also 68% lower than previous chips. With a maximum search range of ±211 (horizontal) by ±106 (vertical) around a predictive search center, the proposed motion estimation processor well accommodates the high motion of UHDTV.

Index Terms

Ultra High Definition Television, UHDTV, Super Hi-Vision, SHV, H.264/AVC, video encoder, VLSI, ASIC, motion estimation
I. INTRODUCTION

Ultra high definition television (UHDTV) includes 4K (3840×2160) and 8K (7680×4320) formats. By delivering 4× to 16× the number of pixels per frame compared to today’s high definition, UHDTV achieves remarkably enhanced visual experience and is currently being promoted as the next-generation standard of digital television. To store and transmit the huge volume of UHDTV data, efficient and real-time compression is essential. While the latest video coding standards such as H.264/AVC [1] and H.265/HEVC [2] provide excellent compression ratio, they also involve high computational complexity and memory bandwidth requirements. This leads to critical challenges to the implementation of video codecs in meeting the constraints of throughput (for real-time capability) and power dissipation (for extended battery life).

Motion estimation (ME) is a key encoding component of almost all modern video coding standards. By reducing the temporal redundancy between frames, motion estimation significantly improves compression ratio and video quality relative to intra-only coding. However, the algorithms of ME also involve intensive computation and heavy data traffic, which makes it the major bottleneck of a video encoder. In existing video encoder chips (e.g. [3], [4]), ME processors usually compose over half of chip area and the vast majority of memory bandwidth utilization. As a result, the design of ME processor is also a determining factor of the power efficiency of video encoder VLSI. In transition from high definition to UHDTV, not only pixel rate, but also motion distances in pixels grow extensively [5]. Accordingly, larger search range (SR) is required by ME to capture fast moving objects, relative to the SRs (typically 64 to 128) adopted by high definition ME processors. As a result, the overall complexity of ME rises more than proportionally to the number of pixels.

This work focuses on algorithm and VLSI architecture co-optimization of motion estimation for UHDTV applications. Key optimizations for reducing circuit area include: a) for integer motion estimation (IME): a rhombus search window based full-search architecture; b) for
fractional motion estimation (FME): a directional 5T12S (5 transform candidates among 12 search candidates) search pattern and an LPT-DCT (low pass truncated discrete cosine transform) based cost generation architecture. An efficient reference pixel fetching mechanism is also designed to minimize memory traffic while giving support to a predictive search center, by integrating features including sliding cache, block reordering and frame interleaving. A test chip supporting H.264/AVC has been implemented in 40nm CMOS. It demonstrates the effectiveness of the proposed optimizations with 7.5× higher throughput, 23% better core energy efficiency, and 68% lower memory bandwidth requirement, in comparison with previous designs. The chip delivers a maximum pixel rate of 1.59Gpixel/s, which means real-time processing capability for 8K video at 48 fps or 4K at over 120 fps. With a maximum search range of ±211 (horizontal) by ±106 (vertical) around a predictive search center, the processor also well accommodates the high motion of UHDTV.

The rest of this paper is organized as follows. Section II introduces motion estimation algorithms and related work. Section III gives an overview of the proposed algorithm and architecture. Section IV and V describe details of the IME and FME architectures, respectively, highlighting the proposed optimizations. Section VI explains the reference pixel fetching mechanism. Section V presents the chip implementation results.

II. MOTION ESTIMATION AND RELATED WORK

In modern video coding standards, motion estimation is a process used to find in previously encoded frames the blocks that well match those in the current frame. Only the blocks’ differences along with a set of displacements called motion vectors need to be encoded so that bit rate can be significantly saved. Fig. 1 illustrates the basic idea of block-matching motion estimation. For a block currently being encoded, a search window is defined in each of the reference frames. Size of the typically rectangle-shaped search window is specified by the search range. A search center specifies the search window’s location, which can either be fixed relative
to the current block’s coordinates (fixed search center), or predicted from the motion vectors of spatial or temporal neighboring blocks (predictive search center). A larger search range usually improves coding efficiency by providing stronger capability of capturing fast moving objects, but is also expensive due to the quadric growth of the search window’s area. A predictive search center is statistically closer to the best motion vector than a fixed search center, which means the former requires a smaller search range than the latter to achieve the same coding efficiency level.

For classic full-search motion estimation, all candidate blocks inside the search window are compared with the current block until the best one is selected based on the costs of both the differences between blocks and the motion vector. Such a process involves high computational complexity. Moreover, since the search window of a current block is usually much larger than the block itself, each pixel in the reference frames needs to be accessed many times for the motion estimation of one current frame. This leads to extensive memory traffic. Modern video coding standards allow advanced features such as fractional motion estimation, variable block sizes, bidirectional prediction and multiple reference prediction. All of them further improve coding efficiency at the expense of even higher complexity and memory bandwidth requirement.

Many techniques have been developed to optimize ME, especially IME. One category of them is to apply pattern based search and early termination to reduce the number of candidates evaluated [6]–[9]. In spite of proving to be very useful in speeding up software based encoders, these approaches usually involve an irregular data processing flow not easy to be pipelined or parallelized in hardware implementation. Therefore most of the hardware IME designs, including those implemented in recently published video encoder chips [3], [4], [10]–[12], are still based on full-search or modified versions of full-search algorithms. Ding, et al. [3] applied a candidate based search center derivation method to improve the performance of a full-search IME processor with ±16 search range. Kao, et al. [13] proposed Macroblock-level parallel IME processing to optimize on-chip memory traffic. Bao, et al. [14], Wu, et al. [15], and Lin, et al.
[16] developed various forms of hierarchical full-search IME architectures to extend the search range with reasonable hardware cost. Fractional motion estimation (FME) is mostly implemented as an additional component after IME to provide half- and quarter-pixel refinement. Several VLSI designs for FME have been discussed in [10], [16], [17].

III. ALGORITHM AND ARCHITECTURE OVERVIEW

A. Overview of the Proposed Algorithm

The IME part of the processor is separated into two steps of coarse search (IMEC) and refinement (IMER) as shown in Fig. 2. IMEC is performed on subsampled and bit-truncated images of the current and reference frames to save computational complexity. Every 4×4 luminance block in the original full resolution is averaged to be a single pixel in the subsampled domain. Furthermore, least significant bits of the results are truncated so that each of them is represented by 6 bits. For encoding a Macroblock (MB) of a P frame, IMEC conducts full-search 4×4-block (subsampled from 16×16 pixels) matching inside a rhombus-shaped search window with a ±52 horizontal and a ±26 vertical search ranges, in the subsampled domain. This corresponds to ±208 and ±104 in full resolution. The best two motion vectors from coarse search, along with the predicted motion vector (MVP), are sent to IMER, which then conducts a variable-block-size (VBS) full search in a ±3×±2 rectangle-shaped window in full resolution around each of the three candidate motion vectors. Block sizes of 16×16, 16×8, 8×16, and 8×8 are evaluated in IMER. For each MB of a B frame, IMEC is performed in two ±106×±54 rectangle-shaped search windows located in two reference frames, before IMER refines a total of four candidate motion vectors. Finally, FME evaluates interpolated half- and quarter-pixel candidates around the refined integer motion vectors.

B. Top-level Architecture

Fig. 3 shows the top-level block diagram of the proposed motion estimation processor. The IMEC, IMER and FME components undertake major computational tasks. They work in parallel
in an MB-level pipeline and share a common processing time budget of 28 clock cycles per MB.

Two independent DDR3 interfaces are employed to provide connectivity to the external DRAM chips. DDR3 (A) is a 64-bit interface for buffering reference frames, while the 32-bit DDR3 (B) is for source frames and motion vectors. By working at a 2× clock domain (420MHz) relative to the core computational components, the two interfaces deliver a maximum bandwidth of 6.72GBps and 3.36GBps, respectively. To reduce the required traffic of DDR3 (A), lossless compression is performed on the reference frame data before they are stored into the off-chip memory. A corresponding decompression component is used to restore the data once they are fetched back. Algorithms, memory organization and VLSI architectures for frame compression and decompression have been discussed in [18].

Two caches implemented with on-chip SRAMs are employed for serving reference frame data to IMEC and IMER. IMER cache can store up to 512KB of full-resolution pixels, while IMEC cache stores a subsampled and bit-truncated version of the same pixels with 24KB SRAM. Data from the decompression component are always distributed simultaneously to the two caches. Instead of having its own cache, the FME component receives reference pixels directly from IMER, so that no additional off-chip memory traffic or dedicated on-chip SRAM is needed.

A memory scheduler component issues data requests to DDR3 controllers, schedule MB-level tasks, and oversees the operations of the whole processor based on the status and progress of the computational components and the data FIFOs between them.

IV. IME ARCHITECTURE

A. Rhombus Window Full Search

Full-search block matching performed inside a rectangular search window is the most widely applied algorithm in VLSI motion estimation processors [3], [4], [11]–[15]. In addition to providing global optimum results, this algorithm benefits from a regular and low-data-dependency processing flow which makes it suitable to be pipelined and parallelized. Efficient
search orders such as snake scan [4] are also available which effectively improves its data reusability and hardware utilization. However, rectangle may not be the best form for a search window from the point of view of coding efficiency-complexity ratio. As shown in Fig. 4 (a), by applying a rectangular search window, it is implicitly assumed that motions in diagonal directions are emphasized more than horizontal and vertical motions. This does not match the characteristics of most natural video sequences. Search candidates close to the four corners are much less probable to be selected as best motion vectors. Consequently the complexity spent in these areas cannot be regarded as well assigned.

In this work we adopt a rhombus-shaped search window in IME coarse search for P frames. As shown in Fig. 4 (b), the four corners are removed from the rectangle and the computational complexity in terms of number of candidate blocks is thereby halved. In the meanwhile, the rhombus window still covers the same horizontal and vertical distances as a rectangular window, as well as a reasonable range in the diagonal directions. As a result, the former can deliver similar coding efficiency as the latter for most video sequences.

B. Parallel Diagonal Scan

The main difficulty in applying a rhombus search window in VLSI design is from the loss of the rectangle’s natural regularity. Classic processing orders for block matching such as raster scan or snake scan no longer work efficiently due to the variable number of candidates in each horizontal row or vertical column of the search window. To address this issue while supporting a high parallelism required for UHDTV motion estimation, we propose a parallel diagonal scan scheme. As illustrated in Fig. 5, its basic idea is to simultaneously process a whole line or several lines of candidates which are parallel to a pair of sides of the rhombus. The search window can be fully explored by being processed one line after another following a diagonal direction that is parallel to the other pair of sides of the rhombus.

In a frame, pixels are always arranged discretely along the horizontal and vertical directions,
so the diagonal line of candidates and the diagonal scan direction are both mapped to zig-zag segments. Fig. 6 shows the detailed schedules for block matching and reference pixel loading. Note that each block-matching candidate in the search window as in Fig. 6 (b) corresponds to a 4×4-block of reference pixels in the subsampled domain, while the reference blocks of neighboring candidates overlap with each other. This results in the shape of the reference pixel window as in Fig. 6 (a). Every clock cycle reference pixels in a stair-shaped region are loaded from the IMEC cache into an array of shift registers. The region is composed of 14 partitions with 4×2 pixels in each of them. The 4×2-partition is also the basic unit of memory access in IMEC cache. The region to be loaded shifts rightward or downward alternatively starting from the upper-left position, so that the whole window of reference pixels can be covered in 28 clock cycles. The 14 partitions inside the stair-shaped region always have consecutive horizontal coordinates. This enables the loading operation to be realized with the parallel accesses to 14 independent memory banks. Fig. 6 (b) shows the schedule of 4×4-block matching. Every clock cycle a maximum number of 105 candidates inside a stair-shaped set can be processed. The maximum set is composed of 13 4×2-blocks and one additional candidate. The set being processed also follows a zig-zag route to cover the ±52×±26 search window in 27 clock cycles. 101 candidates are processed in each even clock cycle while 105 are processed in each odd cycle, except that 53 and 27 are processed in the first and the last cycles, respectively.

Fig. 7 shows the relation between a candidate set and the reference pixels it involves. The reference pixels required by the first cycle of block matching (53 candidates) become available after the 3rd loading cycle is completed. After that, a new set of candidates can be processed once another loading cycle is completed. This results in the 2-cycle pipeline latency between loading and block matching as illustrated in Fig. 8.

Consider a classic rectangular window with the same ±52×±26 spans and the same processing time budget. It requires at least 212 candidates to be evaluated and 28 4×2-partitions to be loaded
in each clock cycle. In comparison, the proposed parallel diagonal scan schedule can process a rhombus window with both computational complexity and memory traffic halved.

Obviously, the proposed schedule can also be extended for different combinations of search ranges and parallelism degrees.

C. IME Coarse Search (IMEC) Architecture

Fig. 9 shows the architecture of IMEC cache and computational units. The cache consists of 16 data memory banks with independent read addresses. Each bank is implemented with a 1R1W SRAM of 256×48 bits. The port size corresponds to a 4×2-partition of 6-bit pixels. Each partition of reference pixels is stored into one of the 16 memory banks according to the mod-16 result of its horizontal coordinate (divided by partition width). Since the 14 partitions have consecutive horizontal coordinates, they are always distributed in different banks and therefore possible to be loaded at the same time. A boundary processing unit supplies values to pixels which are out of the frame boundary. After that, a cyclic shifter unit sorts the partitions according to their spatial locations. The pixels are then shifted leftward or upward into a register array. The array holds the reference pixels necessary for block matching, and shifts out the pixels no longer needed. Parallel processing elements are used to calculate the cost of each candidate by evaluating the 4×4-SAD (sum of absolute differences) and the MVCost (motion vector cost). A comparator tree is used to find the candidates with lowest and 2nd lowest costs of each processing clock cycle, and thereby concludes the best and 2nd best candidates of the whole search window.

D. IME Refinement (IMER) Architecture

In designing the IMER and FME components, the data traffic volume of on-chip SRAM becomes a critical challenge in addition to computational complexity and off-chip memory bandwidth. We take two measures to alleviate this issue. Firstly, variable-block-size (VBS) search is only enabled after IMEC. IMER searches for all block sizes are performed around the same search center for each coarse result from IMEC. This contributes to up to 75% reduction of
SRAM traffic and complexity for IMER from the reuse of reference pixels and SAD trees. Secondly, loading operations of reference pixels are integrated for the IMER and FME components. IMER now loads an extended region of reference that includes additional pixels for FME. Once IMER is finished, the loaded pixels are directly passed to FME, which removes the necessity of data traffic between FME and the cache memory or external memory.

IMER performs VBS (16×16, 16×8, 8×16 and 8×8) search inside a ±3×±2 rectangular window in full resolution. This requires a 22×20 region of reference pixels. Further taking FME into consideration, a ±3×±3 extension of the region is necessary to supply additional pixels in generating interpolated fractional pixels. The resulting reference region size is 28×26, as shown in Fig. 10. Such an area, regardless of its location, can always be covered by 8×8 partitions with 4×4 pixels in each partition. This feature is used to build the cache memory organization and reference loading schedule of IMER. Note that each partition is aligned to its own size. We use it as the basic unit of data storage in the IMER cache.

The data memory of IMER cache is composed of 16 banks. Each bank is implemented with a 1RW SRAM with 2048×128 bits, capable of reading or writing one 4×4-partition per clock cycle. A partition is stored into one of the 16 banks according to the combination of the mod-4 results of both its horizontal and vertical coordinates, as shown in Fig. 11 (a). Every clock cycle 16 partitions composing a quarter (Fig. 10) of the target region can be loaded. A 2-D cyclic shift process as shown in Fig. 11 (b) and (c) is used to sort the partitions in their natural spatial order as in Fig. 11 (d).

Fig. 12 shows the IMER block matching schedule. Every clock cycle 5 candidates in a column of the search window are evaluated. For each candidate, 9 costs for 16×16 down to 8×8 modes are generated by using a VBS SAD tree architecture [3]. The whole search window can be explored in 7 clock cycles. Since the maximum number of IMER searches is 4 for an MB, it can still be processed in the common time budget of 28 clock cycles.
V. FME ARCHITECTURE

For each MB, fractional motion estimation (FME) receives 12 or 16 modes (4 block sizes combined with 3 or 4 IMER search windows) from IME. Mode filtering [16] is applied to select 4 most likely modes from them to be processed by the FME architecture as shown in Fig. 13. Integer reference pixels from IMER are first used to interpolate half pixels with 2-D low-pass filtering [19]. Transforms are then performed on differences between the current blocks and the reference blocks corresponding to integer- and half-pixel candidate locations. In H.264, quarter pixels are generated from integer and half pixels with bi-linear filtering. Therefore transform coefficients of quarter-pixel candidates can also be generated by averaging the coefficients of integer- and half-pixel candidates. Fractional motion vectors as results are decided by comparing SATD (sum of absolute transformed differences) and MVCost among the candidates.

Two optimizations are proposed for the FME architecture. A directional 5T12S search scheme reduces the number of fractional candidates, especially those that require transforms. Low-pass truncated DCT further reduces the complexity of each replica of cost generation hardware. As a result, FME hardware cost is saved by 52.3% with similar coding efficiency delivered.

A. Directional 5T12S Search Scheme

In FME design, number of candidates is a major trade-off between coding efficiency and complexity. In hardware, nevertheless, candidates at different locations do not correspond to the same complexity since transform coefficients of a quarter-pixel block may be generated directly from the coefficients of integer- and half-pixel blocks, rather than from a full transform. Based on this feature, an FME algorithm as shown in Fig. 14 (a) that explores a 5×5 fractional-pixel window is applied in several previous designs including [10]. We name it as 9T25S since it searches a total of 25 candidates including 9 integer- and half-pixel candidates that require transforms. All the rest 16 quarter-pixel candidates are generated directly without transform.

To further reduce complexity, we propose a directional 5T12S scheme as shown in Fig. 14 (b)
by utilizing the available information of neighboring IME costs. A cost is calculated for each corner of the square formed by the surrounding integer-pixel candidates, by averaging IME costs of the candidate at the corner and its two closest integer-pixel candidates. Using these costs, the best corner is first decided and then the 2nd best is chosen from its two neighbors. Directions to the two bests are used to contour a trapezoid-shaped region that contains a total of 12 candidates including 5 requiring transforms. Compared with 9T25S, directional 5T12S saves hardware for cost generation by near half with only a coding efficiency degradation of 0.02 dB in BD-PSNR.

B. Low-pass Truncated DCT

Most conventional FME designs use Hadamard transform in cost generation. To calculate a Hadamard based 8×8-SATD, differences between current and reference blocks should be processed by two stages of 1-D transforms before a third stage calculates the SAD of the transform coefficients to get SATD. In hardware, register arrays with the same size as the transform block are usually needed to pipeline the computation.

In this work, we propose a low-pass truncated DCT (LPT-DCT) scheme that calculates SATD by accumulating only the low-frequency components or the upper-left 4×4-area of the 8×8-block of transform coefficients, as shown in Fig. 15. While storage is no longer needed for the discarded coefficients, 50% and 75% of the registers after the 1st and 2nd 1-D DCT stages can be saved, respectively. Complexity for calculating SAD and generating quarter-pixel coefficients can also be reduced by 75%. Although the butterfly arithmetic for DCT is more complex than that of Hadamard transform, the 2nd stage of LPT-DCT only need to process half the inputs, which corresponds to 50% complexity reduction.

As for efficiency, DCT is originally the transform used in coding the coefficients. It is thereby naturally better than Hadamard transform as a cost function. Although discarding high-frequency components lessens this advantage, LPT-DCT still shows slightly better (0.03 dB BD-PSNR) coding efficiency than Hadamard due to DCT’s energy compaction property.
VI. REDUCING MEMORY BANDWIDTH REQUIREMENT

Real-time UHDTV motion estimation involves huge memory bandwidth requirements. Even if the proposed algorithms have reduced the data volume of reference pixel loading, IMEC and IMER for each MB in a P frame still require 2.352KB and 3.072KB of memory traffic, respectively. This corresponds to overall traffic of 33.74GBps for 4320p 48fps UHDTV. If all these data are directly loaded from DRAM, at least a 64GBps interface is needed with DDR overhead taken into account. This is already far beyond the maximum bandwidth of DDR4 [20]. Therefore a significant data traffic reduction further on the architecture level is expected.

The memory architecture is also expected to support using a predictive search center. Many previous works on memory traffic reduction assume a fixed search center [21], [22]. Compared to a predictive search center, motion estimation on a fixed center requires a much larger search range to achieve the same coding efficiency. While being acceptable at lower resolutions, the additional complexity may be too expensive for a UHDTV encoder.

A. Constraint for Fetching Reference Pixels

A group is defined to be \( M \) consecutive rows of MBs inside the current frame that is thereby divided into several groups not overlapped with each other. For each group, a belt containing \( N \) rows of pixels is assigned in each reference frame. For P frames and B frames, \( M/N \) is set as 8/512 and 4/256, respectively. The group-belt relationship is illustrated in Fig. 16. MBs inside a group are allowed to fetch reference pixels from the corresponding belt. This makes it possible for the search center of IMEC to be determined according to the motion vector predictor as long as at least half of the resulting reference window is contained in the belt. Since a belt already covers the whole width of the reference frame, constraint on the search center is mainly placed on its vertical component, which has to be pulled back into the belt when the motion vector predictor points out of it. To compensate for this relatively low degree of freedom, the vertical location of each belt relative to the corresponding group is determined by the average of vertical
motion vectors in the previous group. This improves the capability of capturing global vertical motion by allowing a group-by-group updating of belt locations.

B. Sliding Cache

Reference pixels inside a belt are reused by MBs in the corresponding group with a sliding cache as shown in Fig. 17. The cache always holds a segment of the belt, with the width of the segment equal to the quotient of cache size and belt height. For P frames, the segment size is 1024×512. When an upcoming MB requests for a reference window not fully cached, the cache is updated by sliding the segment rightward to contain the whole reference window. Reference pixels fetched from DRAM are stored into the SRAMs of IMER cache to replace the discarded data. A subsampled and bit-truncated version of the same segment is stored for IMEC. The sliding cache does not need a tag RAM. Instead, the memory scheduler maintains the cache’s status by recording the leftmost and rightmost position of the segment.

For each group, its belt is designed to be fetched from DRAM only once by disallowing the cached segment to slide back leftward unless in group switching. Under level C+ data reuse [22], the average volume of data fetched for an MB is reduced by 81% to 1.024KB. With lossless frame compression [18], the required data volume is further reduced by 50% in average to be only 2 times of the MB size.

C. MB Reordering and Frame Interleaving

The MB processing order inside a group is changed as shown in Fig. 18. The alternative order still produces standard compatible outputs for each MB by keeping its access to all neighboring information [22]. In the meanwhile, it achieves data reuse between different rows of MBs in the same group. The data dependencies between the memory scheduler, IMEC, IMER and FME components are meanwhile alleviated, enabling pipelining of the motion estimation processor.

For B frames, MB reordering is combined with frame interleaving as shown in Fig. 19. For every two or three B frames with the same reference frames, the same belts are shared by
collocated groups to reduce memory bandwidth requirement further by 50% or 67%.

VII. RESULTS

A. Coding Efficiency

The proposed motion estimation algorithm is compared with JM 17.2 reference software. The reference motion estimation algorithm is configured to perform full search in full-resolution rectangular search windows for IME, and the default two-iteration search for FME. Its search ranges and number of reference frames are set to be the same as the proposed algorithm. TABLE I summarizes the key non-ME coding parameters and the test video sequences with resolutions ranging from 1080p to 4320p. Experiments show that the proposed algorithm’s coding efficiency is only 0.08 dB lower than the reference in BD-PSNR. The coding efficiency loss is mainly from the optimizations that trade off search precision for a large search range, including subsampling in coarse search and performing variable-block-size search only in refinement search. With limited computation power relative to the high throughput requirement of UHDTV, such a tradeoff effectively reduces the possibility of drastic coding efficiency degradation from a real motion vector located outside of the search range. In the meanwhile, the tradeoff does reduce the algorithm’s sensitivity to slight changes of motion vectors, but the corresponding coding efficiency loss is usually insignificant after the refinement search is performed. Fig. 20 shows the coding performance comparison with a previous UHDTV motion estimation design [3], which performs only full-resolution search inside a small search window. In comparison, the proposed one performs significantly better (0.30-0.56 dB) for high-motion sequences (Nebuta and Locomotive) and only slightly worse (0.01-0.06 dB) for the rest.

B. Test Chip Implementation

A test chip of the motion estimation processor is fabricated in 40nm CMOS [23]. TABLE II and Fig. 22 show the chip specification and micrograph, respectively. The chip size is 4.85×3.2 mm². I/O, PLL and peripherals for two independent DDR3 interfaces are placed around the chip.
2458K equivalent gates of standard cells and 552KB of SRAMs for the processor compose the rest area. The memory for IMEC and IMER caches forms the majority of SRAMs. Clocks for the DDR3 and the processor domains follow a fixed frequency ratio of 2:1.

PCB-based test shows the chip works successfully when the processor and DDR3 work at 210MHz and 420MHz, respectively, with 10.08GBps bandwidth delivered by the two DRAM interfaces. Maximum throughput achieved is 1.59Gpixel/s for 7680×4320 48fps video. The corresponding core power consumption is 622mW.

C. Chip Comparison

Fig. 21 shows the comparison on DRAM bandwidth requirements. With the architecture-level optimizations including sliding cache and frame compression, DRAM traffic is reduced by 91% to be 3.19GBps for 7680×4320 48fps video. Relative to previous designs [3], [4], [16], the proposed processor requires at least 68% less bandwidth for delivering the same throughput.

TABLE III summarizes the comparison between the proposed and previous designs. For complete video encoder chips [3], [4], the area of only their ME components are listed. Their power consumption figures for ME are estimated from overall power and area proportion. The proposed chip delivers 7.5× or higher throughput with a larger search range. Core energy efficiency (0.39nJ/pixel) is also improved by at least 23%. It should be noted that, due to its target application of UHDTV, a significant proportion of our chip’s core power, such as that for the large SRAMs and the reference frame decompression component, is dissipated in exchange for lowering the DRAM traffic. Therefore, the architecture efficiency of our design is actually improved more than that reflected by the chip’s energy efficiency.

VIII. SUMMARY

This paper presents several techniques for designing efficient motion estimation architecture for UHDTV applications. Rhombus window full search is developed based on a hierarchical IME to reduce both complexity and memory traffic. 5T12S directional search and LPT-DCT
reduce the complexity of FME. DRAM bandwidth requirement is also significantly reduced with a sliding cache design. Effectiveness of these techniques has been proved in silicon.

The implemented chip was designed for H.264/AVC. In the latest HEVC standard published in 2013, basic framework of motion estimation isn’t changed. Therefore the proposed techniques should also be useful for HEVC motion estimation.
REFERENCES


LIST OF FIGURES

Fig. 1 Block-matching motion estimation.
Fig. 2 Proposed IME flow for an MB in P-frame coding.
Fig. 3 Top-level block diagram of the motion estimation processor.
Fig. 4 Rhombus vs. rectangular search windows.
Fig. 5 Basic idea of parallel diagonal scan.
Fig. 6 Reference pixel loading and block matching schedules for rhombus window full search.
Fig. 7 Relative positions of candidates and reference pixels.
Fig. 8 Timing relation between pixel loading and block matching in IMEC.
Fig. 9 Architecture of IMEC cache and IMEC computational units.
Fig. 10 Organization of IMER reference pixels.
Fig. 11 IMER cache architecture and 2-D cyclic shift to sort the partitions in a natural order.
Fig. 12 Block matching schedule for IME refinement search inside a ±3×±2 window.
Fig. 13 Block diagram of the FME architecture.
Fig. 14 Conventional 9T25S vs. proposed directional 5T12S schemes for FME.
Fig. 15 Low-pass truncated DCT as FME cost function.
Fig. 16 Mapping from a “group” to a “belt”; MBs in one group share reference data from the same belt.
Fig. 17 Sliding cache updating inside a belt.
Fig. 18 MB reordering inside a group.
Fig. 19 MB reordering combined with frame interleaving for B frames.
Fig. 20 Coding efficiency comparison on 4320p video sequences. “Nebuta” and “Locomotive” are sequences with high motion.
Fig. 21 Comparison on DRAM bandwidth requirement.
Fig. 22 Chip photo.
Fig. 1 Block-matching motion estimation.

[Diagram showing block-matching motion estimation]

**Step 1: Coarse search**
- 16:1 subsampled
- truncated to 6bit/sample
- 2 best cand. + 1 mvp

**Step 2: Refinement**
- ±3(H)/±2(V) around 3 coarse results
- cand. 1
- cand. 0
- mvp

4x4: 1 subsampling

Fig. 2 Proposed IME flow for an MB in P-frame coding.

[Diagram showing proposed IME flow]

Fig. 3 Top-level block diagram of the motion estimation processor.
(a) Rectangle shaped search window.

(b) Rhombus shaped search window.

Fig. 4 Rhombus vs. rectangular search windows.

Fig. 5 Basic idea of parallel diagonal scan.
Fig. 6 Reference pixel loading and block matching schedules for rhombus window full search.
Search candidates processed in a single clock cycle = 105.

(b) Corresponding reference pixels needed.

Fig. 7 Relative positions of candidates and reference pixels.

Fig. 8 Timing relation between pixel loading and block matching in IMEC.
Fig. 9 Architecture of IMEC cache and IMEC computational units.

Fig. 10 Organization of IMER reference pixels.
Fig. 11 IMER cache architecture and 2-D cyclic shift to sort the partitions in a natural order.

Fig. 12 Block matching schedule for IME refinement search inside a ±3×±2 window.
Fig. 13 Block diagram of the FME architecture.

Fig. 14 Conventional 9T25S vs. proposed directional 5T12S schemes for FME.

Fig. 15 Low-pass truncated DCT as FME cost function.
Fig. 16 Mapping from a “group” to a “belt”; MBs in one group share reference data from the same belt.

Fig. 17 Sliding cache updating inside a belt.

Fig. 18 MB reordering inside a group.

Fig. 19 MB reordering combined with frame interleaving for B frames.
Fig. 20 Coding efficiency comparison on 4320p video sequences. “Nebuta” and “Locomotive” are sequences with high motion.

Fig. 21 Comparison on DRAM bandwidth requirement.
Fig. 22 Chip photo.
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE I</td>
<td>Test Conditions.</td>
</tr>
<tr>
<td>TABLE II</td>
<td>Chip specification.</td>
</tr>
<tr>
<td>TABLE III</td>
<td>Chip comparison.</td>
</tr>
</tbody>
</table>
TABLE I  TEST CONDITIONS.

<table>
<thead>
<tr>
<th>Reference software</th>
<th>JM 17.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantization parameters</td>
<td>24, 28, 32, 36</td>
</tr>
<tr>
<td>Frame structure</td>
<td>IBBBP</td>
</tr>
<tr>
<td>Entropy coding tool</td>
<td>CABAC</td>
</tr>
<tr>
<td>Test sequences (1080p)</td>
<td>Tractor, BlueSky, RushHour, Station, Pedestrian, SunFlower, BasketballDrive, BQTerrace, Cactus, Kimono, ParkScene</td>
</tr>
<tr>
<td>Test sequences (2160p)</td>
<td>ParkJoy, IntoTree, CrowdRun, DucksTakeOff</td>
</tr>
<tr>
<td>Test sequences (4320p)</td>
<td>8 commercial native 8K sequences</td>
</tr>
</tbody>
</table>

TABLE II  CHIP SPECIFICATION.

<table>
<thead>
<tr>
<th>Process</th>
<th>SMIC 40nm CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply voltages</td>
<td>1.1V core, 2.5V/1.5V I/O</td>
</tr>
<tr>
<td>Chip size</td>
<td>4.85×3.2mm² (incl. DDR3 PHY, DLL and PLLs)</td>
</tr>
<tr>
<td>Package</td>
<td>280-pin BGA</td>
</tr>
<tr>
<td>Logic gates</td>
<td>2458K (equivalent 2-input NAND)</td>
</tr>
<tr>
<td>On-chip memory</td>
<td>552KB (SRAMs and register files)</td>
</tr>
<tr>
<td>External memory</td>
<td>64-bit DDR3 (A) + 32-bit DDR3 (B) SDRAM</td>
</tr>
<tr>
<td>Maximum pixel throughput</td>
<td>1.59Gpixels/s</td>
</tr>
<tr>
<td>Maximum resolution</td>
<td>7680×4320 48fps</td>
</tr>
<tr>
<td>Core power @1.1V/25°C</td>
<td>622mW @210MHz/420MHz (core/DDR3)</td>
</tr>
</tbody>
</table>
### TABLE III  CHIP COMPARISON.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. resolution</td>
<td>4320p48</td>
<td>2160p24</td>
<td>1080p30</td>
<td>1080p60</td>
</tr>
<tr>
<td>Max. throughput</td>
<td>1593Mpixel/s</td>
<td>212Mpixel/s</td>
<td>62Mpixel/s</td>
<td>124Mpixel/s</td>
</tr>
<tr>
<td>Inter frame types</td>
<td>P, B</td>
<td>P, B</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Search range H/V (P)</td>
<td>±211/±106</td>
<td>±16/±16</td>
<td>±96/±64</td>
<td>±128/±128</td>
</tr>
<tr>
<td>Search range H/V (B)</td>
<td>±107/±56</td>
<td>±16/±16</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Technology/ supply</td>
<td>40nm/1.1V</td>
<td>90nm/1.2V</td>
<td>0.18µm/1.8V</td>
<td>0.13µm/1.2V</td>
</tr>
<tr>
<td>Logic gates/ SRAM</td>
<td>2458K/552KB</td>
<td>909K 1)</td>
<td>689K/41KB</td>
<td>283K/9KB</td>
</tr>
<tr>
<td>ME power</td>
<td>622mW</td>
<td>274mW 2)</td>
<td>737mW 2)</td>
<td>-</td>
</tr>
<tr>
<td>Norm. ME power 3)</td>
<td>0.39nJ/pixel</td>
<td>0.48nJ/pixel</td>
<td>0.98nJ/pixel</td>
<td>-</td>
</tr>
</tbody>
</table>

1) ME gate count calculated from total gate count (1732K) and area proportion of the ME components (52.5%) in the die photo.
2) ME power calculated from encoder core power and ME area proportion.
3) ME power normalized to 40nm/1.1V. (P_{40nm/1.1V} = P_{90nm/1.2V} /2.678 = P_{0.18µm/1.8V} /12.05)