Waseda University Doctoral Dissertation

# Decoding Motion Vector based on Block Merging and Motion Compensation with Distance-biased Cache for Energy-efficient VLSI Architectures of HEVC

Shihao WANG

Graduate School of Information, Production and Systems

Waseda University

July 2017

# Abstract

Recently, 4K/8K ultra-high definition television (UHDTV) has been regarded as the next frontier of video formats. The increased resolutions have stimulated a new video coding standard in 2013 called HEVC (High Efficiency Video Coding). Compared to H.264 standard, HEVC has announced its twice compression capacity to achieve the same video quality at the expense of increased complexity, which is challenging the VLSI implementation of real-time HEVC decoder system. For VLSI implementations, the increases of computational complexity, external memory bandwidth and on-chip memory requirement caused by HEVC are challenging us to explore energy efficient VLSI a /8K UHDTV video decoding hardware. In detail, to pursue better energy efficiency, it is critical to eliminate the redundant memory accesses and computations by designing customized architectures for the complicated HEVC decoding algorithms.

HEVC inherits the basic idea of previous video coding standards where each sample can be intra or inter predicted. Especially, decoding inter prediction samples contributes the major coding efficiency at the expense of more than 49% of decoding complexity and 60% of memory bandwidth requirement. Therefore, it is indispensable to design VLSI architectures for real-time decoding processing, which consists of decoding motion vectors (called PDec: Parameter Decoder) a

. For example, the increased number and maximum size of coding units require more design efforts because architectures should handle all 24 possible cases, which are three times more than that of H.264. Another example is the longer taps of MC interpolator, which directly introduces more memory bandwidth requirements and computations. In total, it is desirable for new VLSI a

.

To remedy this situation, this dissertation presents the proposed energy efficient

VLSI a . These proposals are motivated from a viewpoint of data reuse to reduce both memory accesses and computations by new hardware architectures. The unexploited potentials for data reuse are discovered and utilized to form the main idea in each design. Specifically, in PDec, the potential is that the same result of motion vectors is computed repeatedly for several blocks. Therefore, these blocks are merged together (called “Block Merging”) to reuse this result among blocks for saving computing energy. In MC, I discovered the potential that possibility of reuse is highly related to the distance between two frames. Thus, I applied this relationship to the cache design (called “Distance-biased Cache”) to improve cache memory energy efficiency. Besides the above proposals, the related memory hierarchy and hardware optimization techniques are also carefully designed so as to achieve better energy efficiency for both computing and memory parts of their VLSI a .

The following presents the summaries of four chapters and related evaluation results:

Chapter 1 [Introduction] gives the background and the new features of PDec and MC in HEVC decoder, followed by the discussions on the VLSI design challenges caused by these new features. The perspective viewpoint for solving the design challenges and the research target are then discussed. Meanwhile, the state-of-the-a

.

Chapter 2 [Block Merging based Unified HEVC Parameter Decoder Design] presents an energy efficient VLSI PDec architecture by block merging. The proposed idea of block merging bases on the discovery of unexploited reuse potentials between blocks in previous works. All these blocks share the same MV result while they are calculated for multiple times, once per block. This work merges these blocks together to calculate this MV result only once and reuse this result for all blocks to save computing energy. By doing this, CU-adaptive pipeline is proposed for block merging to save up to 95.5% redundant computations and memory accesses compared to previous works without block merging. Meanwhile, the idea of block merging can also reduce the number of block shapes from 24 to 4 types to reduce the hardware costs.

Besides block merging, a unified architecture decoding not only motion vector but also boundary strength is proposed for memory sharing as well as enhancing data reuse between these two decoding processes. Additionally, the memory architecture and organization are proposed, and PU based coding scheme for co-located storage is introduced for further reducing 30-90% DRAM bandwidth requirement. Finally, the proposed area optimization techniques like index-mapping scheme save area costs by 43.2k logic gates. In total, the proposed PDec design supports real-time 7680x4320@60fps video decoding at 249MHz and it's the world's first design for the new HEVC standard. Besides the reduced DRAM memory accesses, we also achieve 36% logic gate reduction by the proposed block merging and index-mapping scheme compared to the state-of-the-a

-biased Cache based HEVC Motion Compensation Architecture] presents a new cache memory design based on the discovery of the relationship between distance and the data reuse potentials for each reference frame. Instead of fixing all the cache sizes, this work can on-the-fly program the cache into multiple cache sets and determine each cache size based on distances. Based on this idea, a novel cache design with distance biased direct mapping scheme is first proposed. The size of each cache set is inversely proportional to the proposed distance, which is defined as the time interval between current and reference frames. This means a reference frame with small distance is given a large cache set for its good reuse possibility and vice versa. This can achieve a near-optimum hit rate while it involves significantly lower complexity by being direct mapping. Besides, eight-bank cache is organized differently at reading and writing interfaces to double the data delivery efficiency by removing the unused data fetching as many as possible for energy and area saving. Additionally, row based miss information compression is applied and mask-based block conflict check scheme also efficiently solve the potential pipeline hazards as well as reducing the design costs. The proposed architecture achieves 8x throughput enhancement to support 7680x4320@60fps video applications. Compared with the state-of-the-a

improvement in terms of normalized logic gate, memory and power metrics. The demerit is the increased logic gate costs for supporting the reconfigurability of cache.

Chapter 4 [Conclusion] summaries the contributions of this dissertation from the view of data reuse for energy efficiency. The solved and remaining problems are discussed with an expectation for future works.

# Acknowledgement

First of all, I would like to show my deepest gratitude to my parents and my wife for the understanding and supporting over the years.

Secondly, I would like to show my great respect and thanks to my supervisor, Prof. Takeshi Yoshimura, who leads my doctoral research and helps me accomplish this dissertation. His patient guidance and meaningful advices in both research and daily life will always be helpful for my future life. I would also like to thank Prof. Takahiro Watanabe, Prof. Shinji Kimura and Prof. Takeshi Ikenaga for providing me precious comments and advices for this dissertation.

Thirdly, I also want to express my sincere gratitude to my supervisor during the master period, Prof. Satoshi Goto, who gave me continuous encouragement and support throughput the life in Waseda University, and also provides me the opportunity to be enrolled in the study of my doctoral program.

Fourthly, I would like to thank all members in Prof. Takeshi Yoshimura's lab and Prof. Satoshi Goto's lab for sharing the wonderful time with me. Especially, I would like to thank Mr. Dajiang Zhou for his guidance and help during my study in Waseda University.

Finally, I want to show my great respects and thanks to all the professors and students in the Waseda University Program for Leading Graduate Schools (Graduate Program for Embodiment Informatics). Without the financial support and the interdisciplinary exchange opportunities, I could never have such a meaningful period.

# Contents

# Index of Figures

# Index of Tables

# 1. Introduction

## 1.1 Background

V

1 . In the past decades, we have all witnessed the rapid development of videos. One of the direct feeling is the increasing pixel resolutions of videos. Although the High Definition (HD) with 1920×1080 pixels per frame is still the mainstream format for the television broadcasting, we have already started to embrace the new era called Ultra-High Definition Television (UHDTV) [2] for the next-generation format in a near future. Generally, UHDTV may refer to 4K (3840×2160) or 8K (7680×4320) format, which is at most sixteen times more pixels than the conventional HD applications. This can bring consumers much better visual experiences and has been attempted for broadcasting the coming 2020 Tokyo Olympic Games.

However, the increasing demands for higher pixel resolutions from the consumers not only bring a better visual experience, but also lead the video storage and transmission a big challenge for researchers. For example, the real-time transmission of raw 8K applications with 60 frames per second (fps) requires a bandwidth of around 4GB/s. This bandwidth is only for one channel and has already been beyond the existing broadcasting peak bandwidth. Therefore, video coding is the fundamental need for compressing those massive video data. The technology of video coding has been developed for four decades by both researchers and industries. We have witnessed several commercial standards like MPEG1/2/4 and H.264 in 2003. The newest video coding standard, High Efficiency Video Coding (HEVC), has been standardized in 2013 and announced its twice compression capacity to achieve the same video quality compared to the previous H.264 standard. The better compression capacity means that HEVC is a promising solution for up to 8K UHDTV. Therefore, HEVC video decoder design is required by both technology development and market demands.

Inside a video decoder, decoding motion vectors and inter prediction samples (which are called PDec and MC, respectively) are one of the dispensable function modules because inter prediction samples contribute the major coding efficiency as shown in Table 1. It is also the distinguished feature to be different with the image coding [3] by exploiting the temporal correlations for removing the redundancy between frames. However, the merits are obtained at the expense of consuming more than 49% of the decoding complexity [4] and 60% of memory bandwidths, as shown in Fig. 1. Therefore, the implementation of PDec and MC is important because it is not only time-consuming but also energy consuming module inside the whole HEVC decoder. VSLI approaches for PDec and MC are the promising solutions due to the high integration density for high energy efficiency, high speed and low fabrication costs.

Table 1 Better coding efficiency contributed by inter prediction

| Example:600-frame Vidyo4 | Data Size | Coding Efficiency |
|---|---|---|
| Original Video | 829,440KB | 1x |
| Coded with In | | |
| Coded with In In | | |

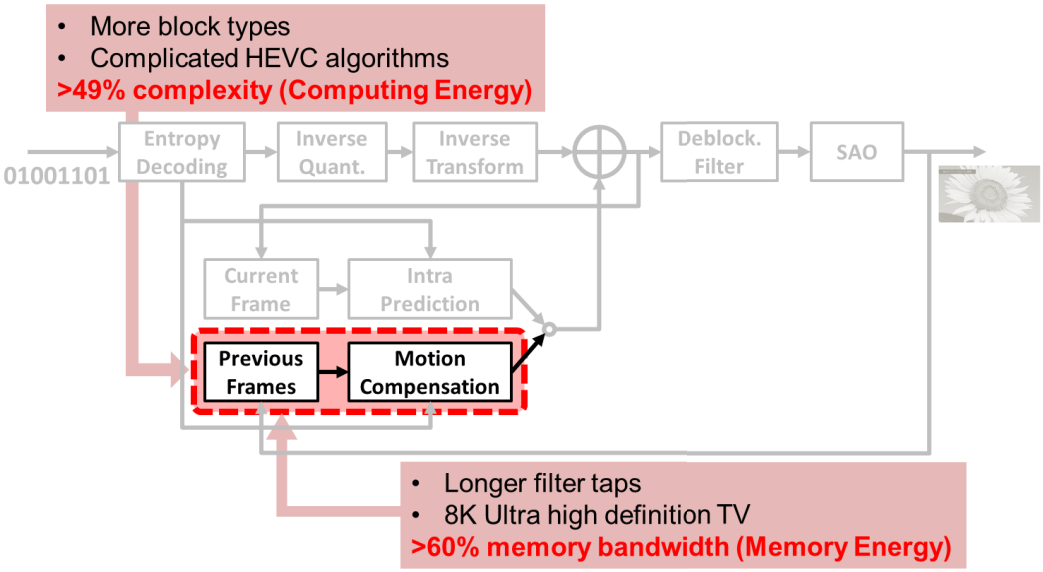


Fig. 1 Significance of PDec and MC in HEVC decoder system

In the VLSI implementations of HEVC decoder system, two modules, PDec and MC are related to the decoding process of motion vectors and inter prediction samples. These two modules are cascaded inside the video decoder. PDec is in charge of decoding motion data like motion vector and reference index from the implicit syntax elements. Then, these motion data are utilized by MC to locate and fetch the reference block data and do the interpolation to generate the compensated blocks as the decoded results of current block. These two modules are researched together not only due to the tight relationships of their functions, but also because they both need to access off-chip memories for storing and fetching data.

Both the parameter decoder and motion compensation in HEVC has been evolved throughout the development of standards for better coding efficiency. This high coding efficiency is good at handling applications with a huge data volume like 4K/8K UHDTV. For example, 8K@60fps containing around 4G pixels/s can be coded into only 30KB/s bandwidth requirement. This can significantly reduce the data for storage and transmission and accelerate the popularization for 8K UHDTV.

In the following discussions, the new coding tools involved in HEVC for improving the parameter decoder and motion compensation are briefly introduced.

### 1.1.1 Coding Tree Units and Coding Tree Block Structure in Both PDec and MC

Block based video coding is the classical procedure throughout the history of video coding including HEVC. HEVC tries to process each video frame block by block, instead of sample by sample. Generally, motion vector is used to represent the movement of objects between frames. However, the problem is that different objects have different movements as shown in Fig. 2. Two objects, sun and cloud are moving vertically and horizontally, respectively. Therefore, it is difficult to depict the movements for the entire frame.

HEVC [5] solves this problem by dividing a frame into coding tree unit. Each

coding tree unit can be further divided into small square blocks (called coding unit in HEVC). Each coding unit can further be split symmetrically or asymmetrically into two rectangular blocks (called prediction unit in HEVC). The target is to make each block belong one specific object. By doing this, each block can have its own motion vector. This kind of block division is called coding tree block structure in HEVC.

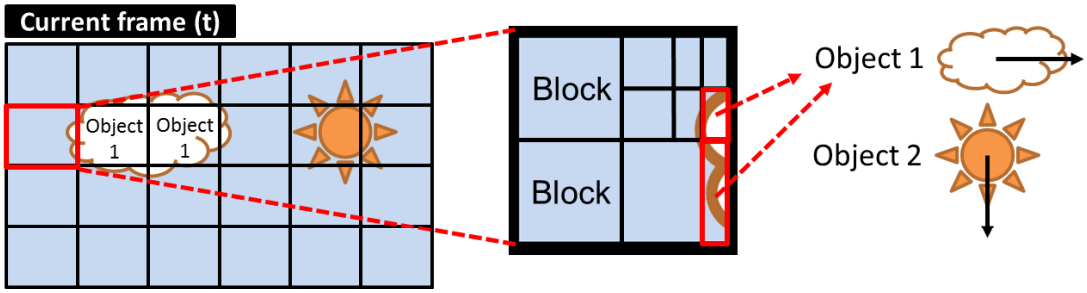


Fig. 2 Block based inter prediction for objects with different movements

Compared to the H.264/AVC [6] in 2003, this coding block tree structure in HEVC is much more complicated and flexible. In H.264, the basic coding block is called macroblock. The macroblock can be divided into small blocks. The smallest blocks in H.264 is 4×4. Meanwhile, each block can be further symmetrically divided into two rectangular blocks. Totally, there are 8 different block shapes supported in H.264 as shown in Fig. 3.

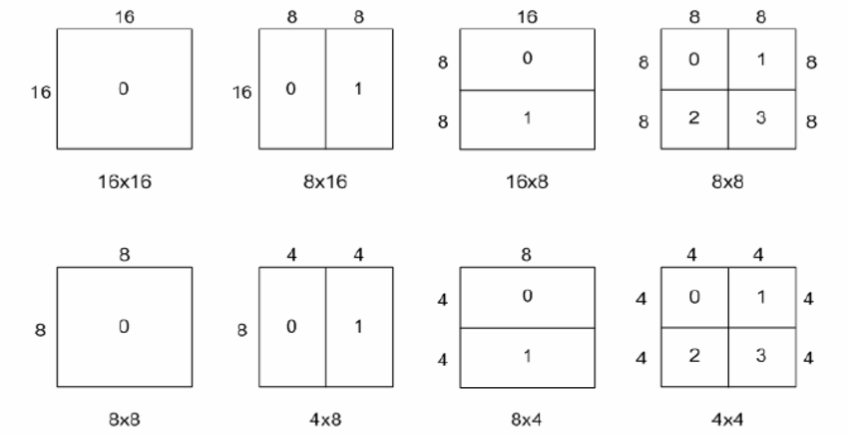


Fig. 3 Possible block sizes in H.264/AVC.

On the other hand, HEVC contains up to 24 block shapes. The largest supporting size is 64×64 in default, which is called a coding tree unit. This coding tree unit can be

hierarchically divided into small coding blocks ranging from 8×8 to 64×64. Meanwhile, the block can also be asymmetrically partitioned. This results an increased complexity for both parameter decoding and motion compensation. To simplify the problem, HEVC limits the asymmetrically partitions of 8×8 blocks to reduce the total number of block shapes. An example of partition for coding tree unit is shown in Fig. 4.

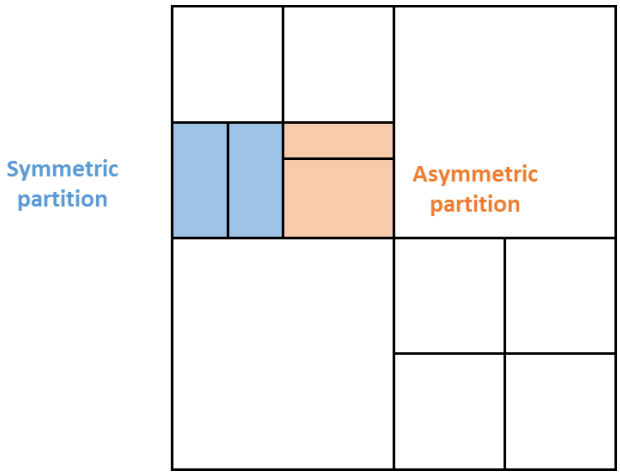


Fig. 4 An example of 64×64 HEVC coding tree block structure.

## 1.1.2 New Features in Parameter Decoder

In block-based video coding, each block has a specific motion data as a vector of (Δx, Δy, Δt). They represent the displacement in horizontal and vertical directions and the distance with reference frames. This motion vector is calculated in the encoder while it is not directly sent from encoder to decoder. Encoder expects this motion vector to be similar to the ones of surrounding blocks. Therefore, encoder will find the most similar one as the motion vector predictor and only the difference between actual motion vector and the predictor is sent to the decoder for coding efficiency.

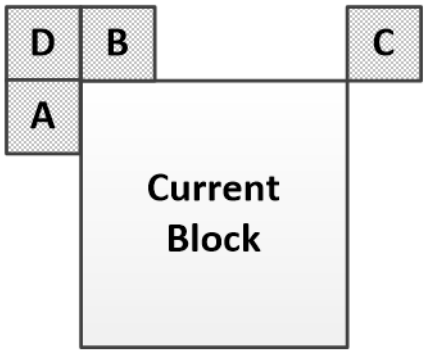


Fig. 5 Spatial neighbours for motion data prediction in H.264

In H.264, four spatial neighbours and one temporal co-located neighbour will be used as candidates of the predictor. Generally, only A, B and C in Fig. 5 will be considered. Block D will be used only when C is not valid. In H.264, if a reference block is intra-coded, then the motion data will be regarded with zero motion vectors. The motion data of current blocks will be a copy of neighbours or the median value among A, B and C in the skip mode.

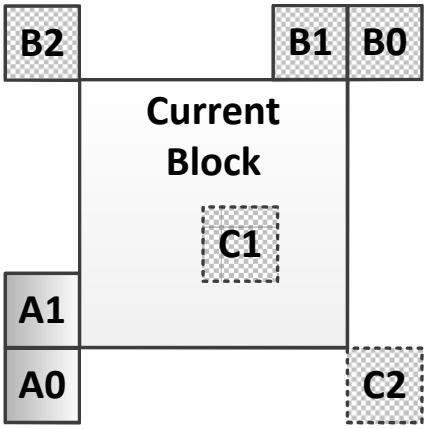


Fig. 6 Spatial and temporal neighbors for motion data prediction in HEVC

In terms of HEVC, the algorithm for motion data coding becomes much more complicated. Firstly, the involved neighbours are more than H.264, as shown in Fig. 6. Five spatial neighbours (A0, A1, B0, B1, B2) a -located neighbours (C1, C2) support two prediction modes in HEVC, advanced motion vector prediction (AMVP) mode and merge mode.

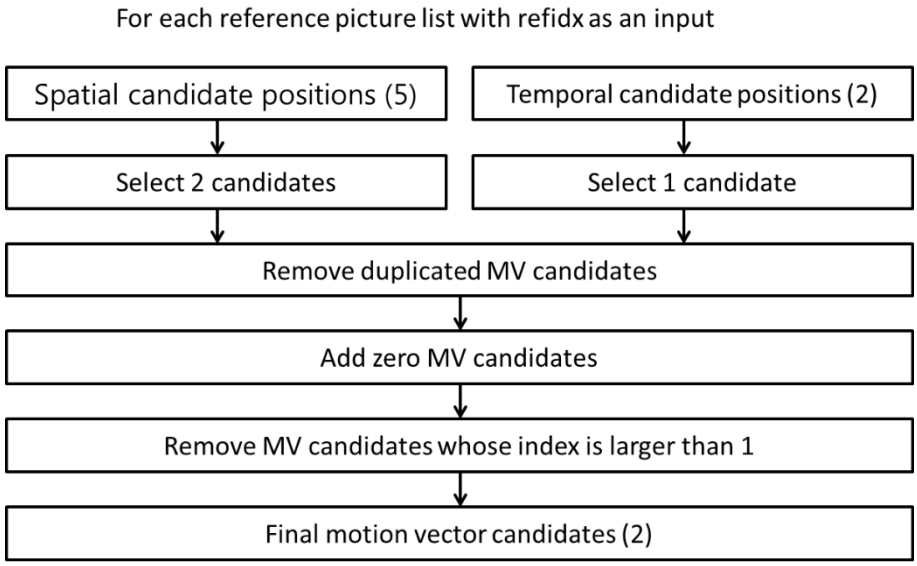


Fig. 7 Derivation of motion vector prediction candidates in AMVP in [12]

AMVP mode is depicted in Fig. **7**. Up to two spatial candidates and one temporal candidate will be first selected. In HEVC, if a neighbour is intra coded, this neighbour is treated as no qualification to attend the competition. Then the algorithm will remove the duplicated candidates and add zero candidates if the number of candidates is not enough. Then, another selection will be conducted to exclude candidates whose $\Delta t$ is larger than 1. Finally, only two candidates will be left and the current block will choose one of them as the predictor according to the received indexes from the bit stream.

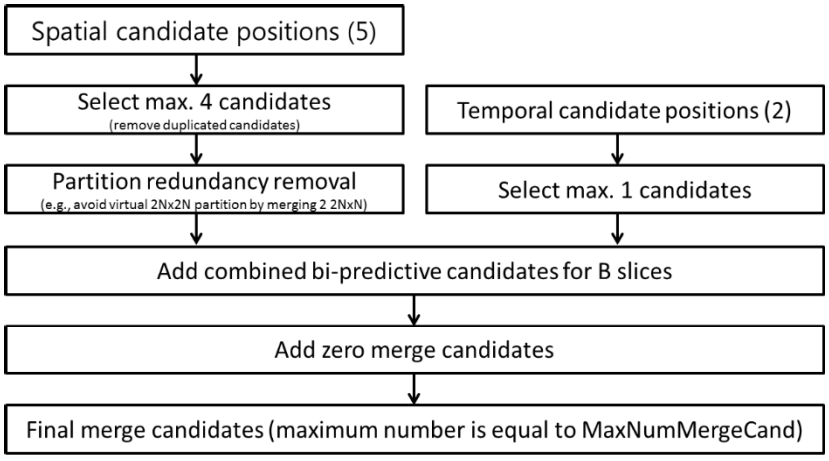


Fig. 8 Derivation of merge candidates in merge mode [12]

Merge mode is similar with the skip or direct mode in H.264. However, the algorithm is complicated instead of copying or selecting the median of neighbors. As

shown in Fig. 8, the first step is also the motion data competition. Up to four and one candidates will be selected from the spatial and temporal neighbors, correspondingly. Then, HEVC involves a new approach to generate combined bi-predictive candidates based on the valid candidates in the motion data competition. Finally, zero candidates with different Δt are selected if the amount of valid candidates is less than five in default. Finally, current block will choose one of them as the motion data predictor according to the received merge index.

### 1.1.3 New Features in Motion Compensation

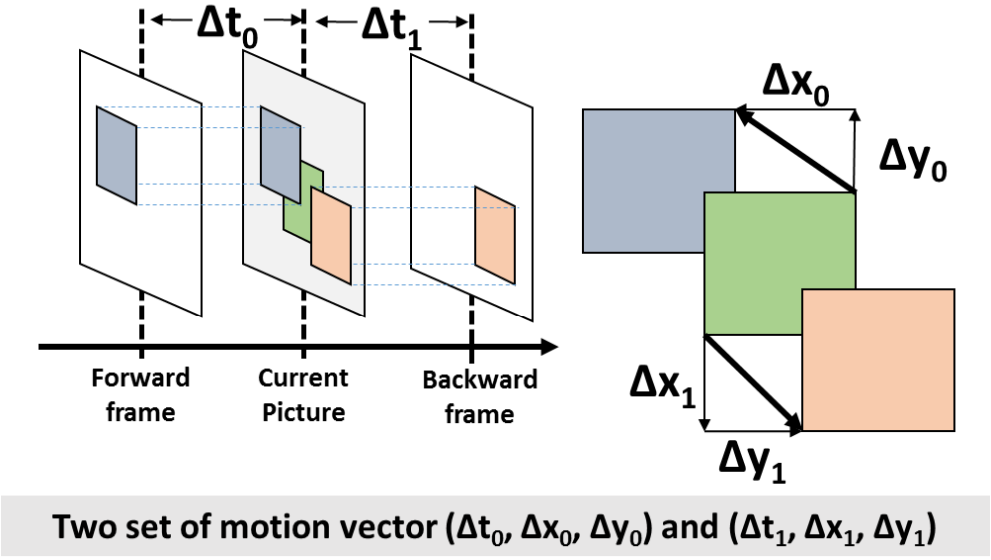


Fig. 9 Bi-directional prediction in HEVC.

Motion compensation in HEVC inherits most of the features in H.264. For example, each block may have one or two motion vectors for prediction. In Fig. 9, an example is given to show a bi-directional prediction where two reference frames come from the prior and past of the current frame. In HEVC, bi-directional prediction can also have two reference frames which are both from the prior frames. This case is called lowdelay configuration. Therefore, HEVC supports a more flexible reference structure compared to H.264.

The second important new feature in HEVC is that the precision of motion vector

(Δx, Δy) is enhanced to support 1/4 pixel with a more sophisticated interpolation filter. For half-pixel interpolation in H.264, a 6-tap two-dimensional interpolation filter is employed. For quarter-pixel, the adjacent integer pixel and half-pixel will be used by a simple linear interpolation. However, HEVC has introduced a new 7/8-tap interpolation filter for either half-pixel samples or quarter-pixel samples, which can provide a better prediction to reduce the data amount of residuals.

## 1.2 Design Challenges – Energy Efficiency

Before discussing the design challenges, I want to first clarify the research target. In this dissertation, I want to design architectures for decoding motion vector and inter prediction samples for real-time decoding HEVC 8K UHDTV with 60 frame per second fresh rate for bi-directional prediction. If I consider a reasonable clock frequency of 250 MHz, approximately 8 pixels per cycle should be achieved for both pixel and motion data decoding. Besides the throughput requirement, I want to achieve high energy efficiency as possible as I can so that to support the demands of battery-limited mobile applications.

In terms of the throughput requirement, the architecture usually require higher parallelism to support more concurrent processing. With the development of Moore's Law [7], it is not the bottleneck to support the higher throughput by parallelism as I can now integrate more logic gates within a unit silicon area.

Instead, the energy efficiency becomes the main target for VLSI designs. As mentioned in [8], computing power cannot be scaled down proportionally like the length of the gates as it is difficult to decreasing the supply voltage with the thinner and shorter gate. On the other hand, memory energy has been consuming more than half of the die energy. Careful designs for hardware architectures are urgently required under the demands for mobility and In 9

### 1.2.1 Computing Energy

Computing power is regarded as the power consumed by the logic gates instead of memories. The computing power is concluded to be produced by two aspects.

The first one is the actual computing for arithmetic calculations. In HEVC, two modes in parameter decoder and interpolation in motion compensation are the main sources of the calculations. The energy costs for these part is fixed by the algorithms and you have few space for optimizing them. However, I think there are potentials to optimize the computing power for data motion prediction. For example, the merge mode introduced in Section 1.1.2 involves many possible intermediate candidates like bi-directional combined candidates. In the hardware design, I have to consider the worst cases that all possible intermediate have to be generated due to the strict requirement for throughput. Totally 24 candidates are produced and five of them are chosen. This direct implementation involves a huge amount of logic gates and optimization is required for saving the computing energy.

The second one is the computing energy for controller. The design of controller will affect the efficiency of arithmetic calculations. An efficient controller can remove the redundant calculations for computing energy saving. In HEVC, the flexible quad-tree block partition dramatically increases the design complexity for the hardware controller. In HEVC, the possible block sizes are listed in Table 2. Compared to H.264, 7 possible cases are increased to 24 cases with two times more complicated.

Table 2 Possible block sizes in HEVC

| Level | Possible sizes |
|---|---|
| 8×8 | 8×8, 8×4, 4×8 |
| 16×16 | 16×16, 16×8, 8×16, 4×16, 12×16, 16×4, 16×12 |
| 32×32 | 32×32, 32×16, 16×32, 8×32, 24×32, 32×8, 32×24 |
| 64×64 | 64×64, 64×32, 32×64, 16×64, 48×64, 64×16, 64×48 |

In the hardware design, it is usual to exploit pipeline technique to improve the

throughput. Generally, designers prefer the basic pipeline granularity to be unique for simple controller designs as hardware is usually good at doing repeated works. However, the 24 possible block sizes in HEVC challenge the design of pipeline controller. It seems that in many previous works they choose a minimum possible block as the pipeline granularity (4×4 for example) for their pipeline solutions, which will be discussed in Section 2.1.4. However, these solutions cannot be adopted to my research for 8K@60fps specification due to their low computing efficiency.

The related complexity for computing power from H.264 to HEVC is listed in Table 3.

Table 3 Computing energy related coding tools from H.264 to HEVC

| | H.264 | HEVC | Complexity |
|---|---|---|---|
| Possible block sizes | 8 types | 24 types | 3× |
| Motion data prediction | Motion vector prediction | AMVP mode + Merge mode | More than 2× |
| Pixel prediction | 6-tap | 7/8-tap | 1.78× |

## 1.2.2 Memory Energy

The memory energy has dominated the main power of the decoder system. Compared to previous works, this research faces more challenges according to Table 4.

Table 4 Memory energy related coding tools from H.264 to HEVC

| | H.264 | HEVC | Complexity |
|---|---|---|---|
| Research target | 4K@60fps | 8K@60fps | 4× |
| Motion data prediction | 4 spatial + 1 temporal | 5 spatial + 2 temporal | 1.4× |
| Pixel prediction | 6-tap | 7/8-tap | 1.78× |

The total amount of memory accesses is increased in both pixel prediction and

motion data prediction. This will proportionally consume the memory energy. Moreover, as the reference data (temporal neighbours in motion data prediction and reference frames in pixel prediction) is massive, off-chip memory like DRAM with high energy costs have to be employed for the storage. This intensifies the design challenges for energy-efficient designs.

Especially, motion compensation consumes the major communication bandwidth with off-chip memories. In HEVC, if I consider the worst case that all the pixels are bi-directional predicted, totally 1020GB/s memory accesses are required. This bandwidth exceeds almost all the existing DRAM technologies.

Currently, cache based motion compensation for pixel prediction is the base for relieving this memory intensive issue. The cache designs are still based on the ideas of that on computers. Existing caches have to decide whether to pursue a high cache performance or low design complexities. There is still a lack of discussion on an efficient cache based architecture that can achieve both metrics. Moreover, cache systems require many peripheral circuits to ensure its functionality. One of the most important is the detect of conflicts due to the pipeline designs in Fig. 10.

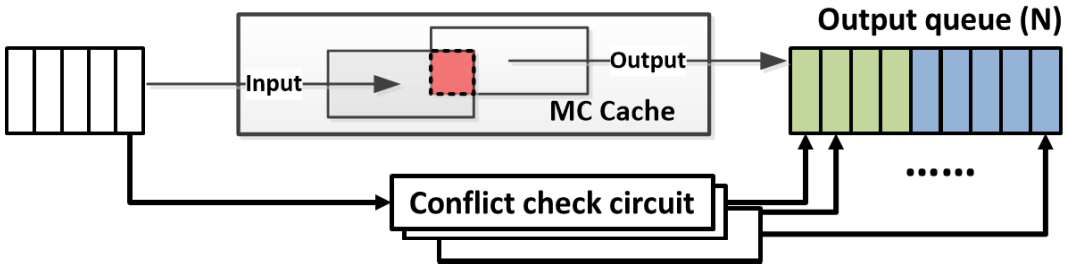


Fig. 10 Conflict check circuits in motion compensation

The pipeline will make the cache be simultaneously written or read. We define an output queue for reading data from cache and another queue for input. For the first write request in the input queue, it should not be written into the regions which will be used by any request in the output queue. Otherwise, read request will fetch the wrong data from cache and I call it a conflict. Therefore, a copy of conflict circuit should be maintained for each request in the output queue. This consumes a big number of logic gates. Moreover, the increased memory bandwidth requirement in Table 4 may lead to

a longer output queue, which will further challenge the hardware design of the cache based motion compensation for pixel prediction.

## 1.3 Data Reuse based Approach

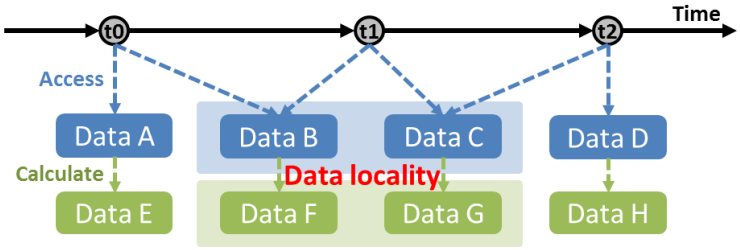


Fig. 11 Data reuse exists in memory accesses and calculations

Data reuse based VLSI design approach is proposed in this dissertation to guide the design of HEVC PDec and MC architectures inside video decoder. Data locality, which provides the potentials for data reuse in HEVC, generally means to use something near the thing you just used as shown in Fig. 11. It defines the cases that the processes are similar for neighboring data. Therefore, it has high possibility that data which are accessed or calculated once are likely to be accessed or calculated again.

Data reuse generally have two types according to the data locality, spatial locality and temporal locality. Spatial locality refers to the use of neighboring data in a short future while temporal locality means the current data will be reused in the future. Temporal locality is the most common type for data reuses. Usually, we will allocate a pitch of on-chip memory for storing these data for the reuse in the near future.

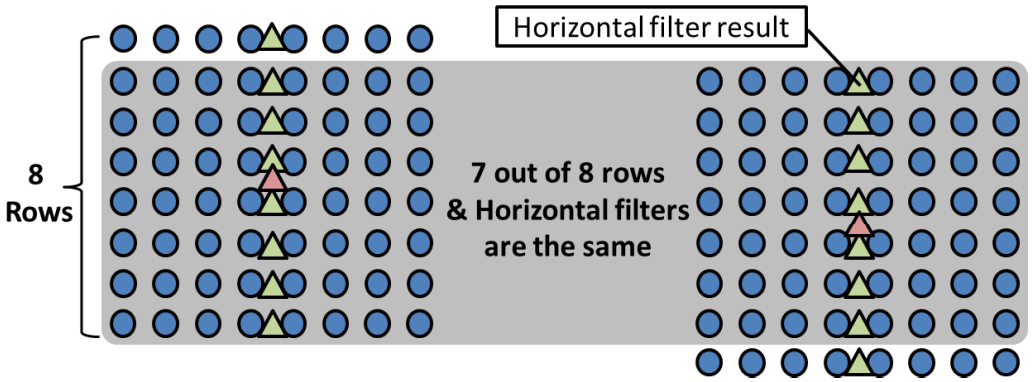


Fig. 12 Example of one possible data reuse inside the motion compensation

Data reuse has the potentials for saving memory energy and computing energy with an efficient architecture. In Fig. 12, an example shows one possible example inside the motion compensation, whose algorithm will be given in Section 1.1.3. To generate each pixel inside the current frame, up to 8×8 reference pixels are required. Moreover, if the vertically adjacent pixel shares the same motion vector with the current pixel and is processed sequentially, there will be 7/8 overlapped rows of reference pixels and horizontal filter results. This provides the potential to reuse these overlapped data for energy efficiency. On the other hand, the spatial locality can be found when off-chip memories like DRAM are accessed. According to DRAM's specification [10], burst access technology is used to eliminate throughput gap. This means you will fetch a bulk of data even if only parts of them are useful for the current calculation. However, the spatial locality implies the rest data of this bulk will possibly be used in the near future. This is a feature for motion compensation.

Based on the data reuse potentials in PDec and MC, the design methodology in this dissertation is proposed in Fig. 13.

The data reuse based approach contains two steps. The first step tries to maximum data reuse in the algorithms of PDec and MC. This is achieved by proposing efficient dataflow. This dataflow decides the order of the processing and the parallelism selection. An efficient process order can significantly reduce the design complexities for the system controller for saving the computing energy. This issue is becoming more and more important for the complicated HEVC standard. Besides, it can also exploit the

data correlations between two adjacent processing units of pixels so that data reuses can be enhanced. The parallelism is also designed to maximize the reference data overlaps between paralleled pixels. These ideas for data sharing can significantly reduce the necessary memory accesses for data fetching so that memory energy can also be reduced.

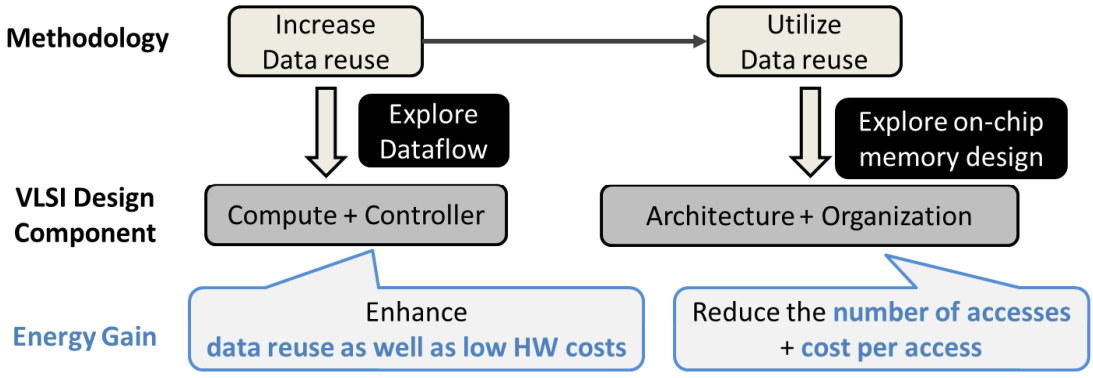


Fig. 13 Data reuse based research approach

The second step is to utilize these data reuse to design the on-chip memory hierarchy and computation core. As we have known the characteristic of the data reuse in the algorithms, we will know how often data are reused since they are first fetched from off-chip memory or from its generation. According to this information, we can carefully design the architecture and organization of memory hierarchy. Two considerations exist here. The first one is to allocate the memory accesses to the low-cost memory level for energy saving by carefully designing memory architectures. The second one is to enhance the data delivery efficiency so that the useful information per access can be maximized as possible as we can by efficient memory organizations. Both can improve the memory energy efficiency.

To apply the proposed data reuse based approach, I have designed the hardware architectures of both PDec and MC for HEVC decoder, which contribute the major part of the total off-chip memory accesses and involve a huge amount of computations. All the works are implemented and evaluated with real hardware by fabricating a real chip to verify its performance and correctness.

### 1.3.1 Conceptual Difference with Works from My Group

Three works [19][22][26] which will be discussed in Section 2.1.4 and Section 3.1.3 are accomplished from my research group. This section clarifies the conceptual difference of the approaches between this dissertation and related works.

Table 5 Conceptual difference between this work and works from my group

| | HEVC | H.264 | Conceptual Difference |
|---|---|---|---|
| **Parameter Decoder** | Chapter 3 of this work | J. Zhou [28] | ***Merge vs. Divide***<br>• **This work *merges*** various block shapes into the same square block shapes<br>• **[28] *divides*** various block shapes into the same small block shapes |
| **Motion Compensation** | Chapter 4 of this work | J. Zhou [34]<br>D. Zhou [20] | ***Unfixed Cache vs. Fixed Cache***<br>• **This work** has ***unfixed*** cache sizes by on-the-fly determining sizes based on reuse possibility<br>• **[20][34]** have ***fixed*** cache sizes by using conventional cache organization |

The conceptual differences are summarized in Table 5. This dissertation focuses on the newest HEVC standard while [22][19][26] are accomplished for H.264 before HEVC is standardized. HEVC involves many new coding tools for more design challenges as are introduced in Section 1.1. Therefore, it is not fair to directly compared the hardware costs like area or energy because the increased complexity of HEVC is requiring more hardware resources. Therefore, we choose to compare the idea differences by assuming their ideas are applied to HEVC, although it is difficult or impossible to apply their ideas, which will be discussion I the following chapters in detail.

For parameter decoder in Chapter 2, I conclude the idea difference as merge and divide. The problem in HEVC is the increased number and maximum size of block shapes as introduced in 1.1.1. [19] tried to solve this problem by dividing them into the same small block shapes and calculating these small blocks sequentially. This work chooses another idea to merge difference block shapes to the same square blocks.

Although both can reduce the number of block shapes, this conceptual difference between divide and merge can significantly reduce the redundant computations, which will be further discussed in Section 2.2.

For motion compensation in Chapter 3, I conclude the difference is the cache design. The contribution of [22][26] is to design the 2D cache using the conventional cache mapping (direct mapping). All the cache sizes are fixed in the hardware. The idea of this work is to achieve unfixed cache sizes by on-the-fly determining sizes based on the distance. This conceptual difference can improve cache reuse possibility based on distance to achieve better balance between cache performance and hardware costs.

## 1.4 Scope of the Dissertation

In this dissertation, decoding motion vector based on block merging and motion compensation with distance-biased cache for energy-efficient VLSI architectures of HEVC are proposed as shown in Fig. 14. Besides the two modules, I also made some contributions to this group work [11] by designing an FPGA-based demonstration system as well as verifying chip function and performance.

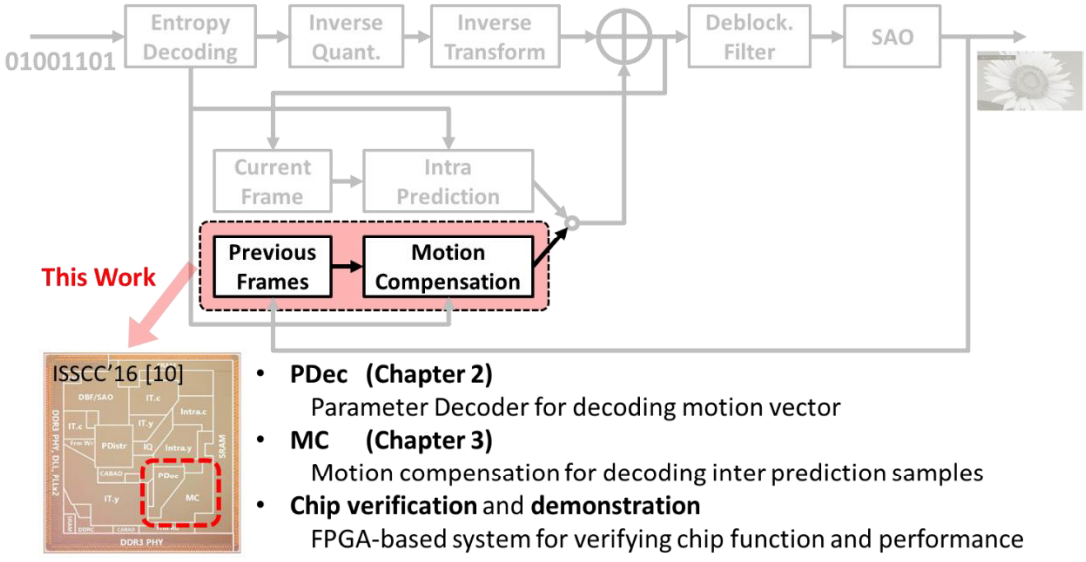


Fig. 14 Contributions inside the HEVC decoder chip [11]

The scenario of this dissertation is shown in Fig. 15.

Before the discussion of the design details, the preliminary knowledge has been

given in Chapter 1. The new features of PDec and MC employed by HEVC are first discussed. Then, we will figure out the design challenges for VLSI in terms of complexity, parallelism, hardware under current silicon technologies. Finally, the frontier of the related works from both my research group and other groups in this research field are concluded at the end of this chapter.

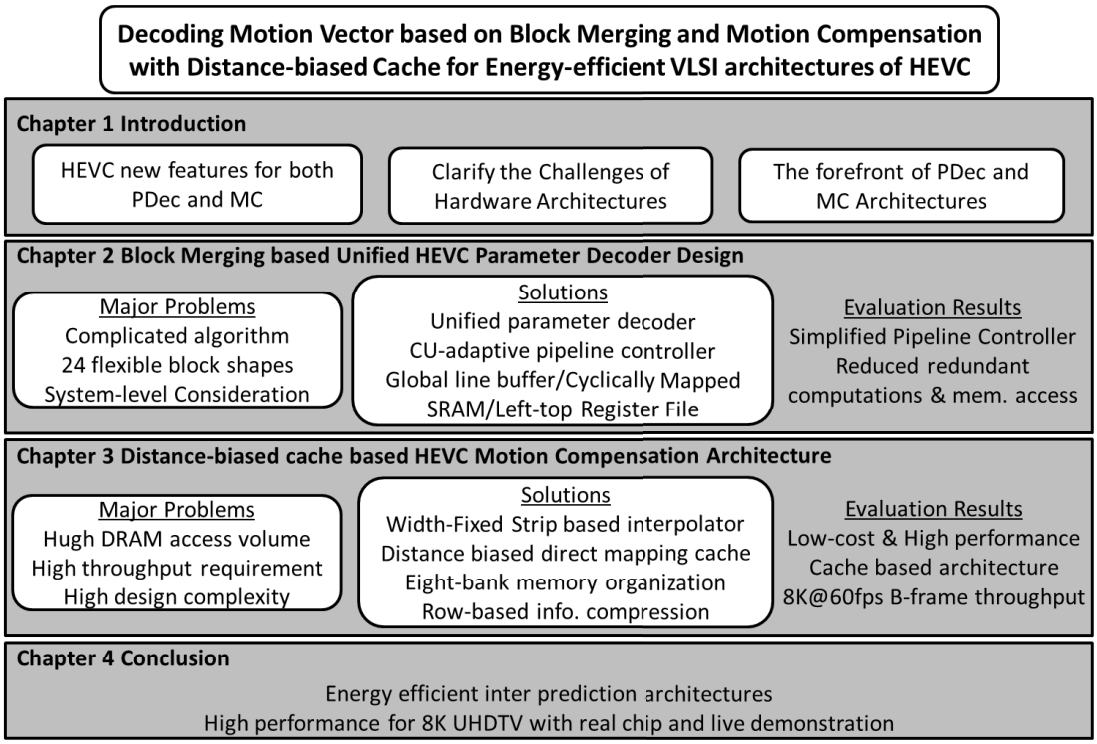


Fig. 15 Scenario of this dissertation.

In Chapter 2, a unified parameter decoder architecture for 8K UHDTV applications is designed with the proposed block merging idea for data reuse. Block merging can reduce block shapes to reduce hardware costs and maximize data reuse to reduce energies for computations by at least 50% (50% is the case where 8x8 CU contains two PUs). This unified architecture for MV and BS parameter decoder is proposed for memory sharing. Secondly, memory architecture and organization are proposed based on temporal data locality, and PU based coding scheme for co-located storage is proposed for further reducing at least 30% DRAM bandwidth requirement. Thirdly, CU-adaptive pipeline is proposed to simplify the design complexity caused by HEVC's complicated algorithm. Finally, the proposed area optimization techniques like index-mapping scheme save area costs by 43.2k logic gates. In total, the proposed PDec

design supports real-time 7680x4320@60fps video decoding at 249MHz and it's the world's first design for the new HEVC standard. Besides the reduced DRAM memory accesses, we also achieve 36% logic gate reduction compared to the state-of-the-a

-biased cache. This cache can improve cache reuse possibility based on the proposed idea of distance. In detail, a novel cache design with distance biased direct mapping scheme is proposed that can achieve a near-optimum hit rate while it involves significantly lower complexity by being direct mapping. Secondly, eight-bank cache is organized differently at reading and writing interfaces to double the data delivery efficiency for energy and area saving. Thirdly, row based miss information compression is applied and mask-based block conflict check scheme also efficiently solve the potential pipeline hazard to reduce the design costs. The proposed architecture achieves 8x throughput enhancement to support 7680x4320@60fps video applications. Compared with the state-of-the-art works, this work shows 76%, 81% and 62% performance improvement in terms of logic gate, memory and power. The demerit is the increased logic gate costs for supporting the reconfigurability of cache.

In Chapter 4, the dissertation is concluded with solved and remaining problems and the future works are also given.

# 2. Block Merging based Unified HEVC Parameter Decoder Design

In this chapter, a VLSI a (PDec) for motion data prediction is proposed. I call it unified parameter decoder because it merges another parameter decoding process (boundary strength decoding) into the motion data decoding, considering the potential data reuse in HEVC. As a result, the proposed parameter decoder in HEVC video decoder is in charge of decoding motion data like motion vector (MV) a (BS) parameters. It is an important module in VLSI decoder design.

Energy efficiency is the main design challenge for this unified parameter decoder. The energy consumption consists of both computing energy and memory energy for this particular module.

In terms of the computing energy, compared to the previous H.264, HEVC involves high complexities including the flexible quad-tree coding unit (CU) structure, variable sizes up to 64, asymmetric partitions and more complicated prediction algorithms for motion vector generation. Moreover, efficient HEVC parameter decoder design is challenging for real-time designs, especially for 8K UHD applications. Both the design complexity and the high throughput requirement increase the design costs like required logic gates. In modern silicon technology, the number of logic gate significantly affect the leakage energy power. Therefore, an efficient design is desired to reduce the computing energy in this part.

Memory energy comes from the writing and reading temporal neighbors in off-chip DRAM memory. With the increasing throughput requirement and complicated algorithm, the burst memory bandwidth and the total memory bandwidth are both increased. Therefore, a proper on-chip memory hierarchy should support a higher burst memory bandwidth as well as reducing the total memory bandwidth requirement as DRAM memory accesses are quite expensive compared to the on-chip memory

accesses and the computing energy.

In this chapter, I introduce the block merging based design approach for parameter decoder design. Generally, the process order at the coding tree unit level is pre-determined by the HEVC decoding algorithm. Therefore, the pre-known order can help us save energy consumption from two aspects. For computing energy, I can design a proper pipeline scheme so as to maximum the reuse for data or computing resources. For memory energy, on-chip memory hierarchy can be optimized according to the data reuse. For those data that will be reused in a relatively long time since it was first fetched, I may prefer to store them in the global buffer instead of registers for energy saving. Such kinds of ideas to exploit the potential data reuse occur in many proposals of this parameter decoder.

## 2.1 Introduction

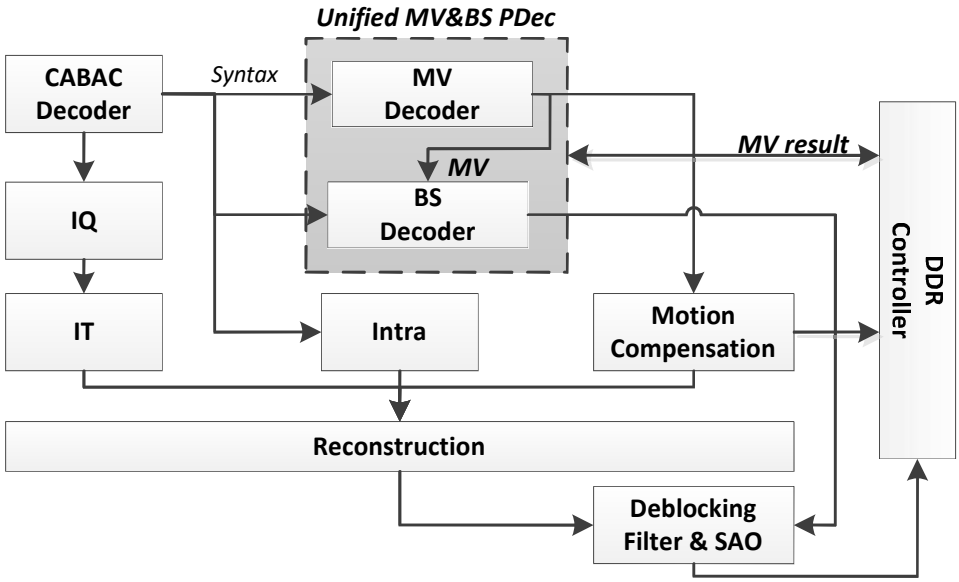


Fig. 16 Unified MV and BS parameter decoder

In this section the overview of parameter decoder is discussed in detail. My proposed unified parameter decoder contains two major parts, motion vector calculation and boundary strength calculation, as shown in Fig. 16. The calculation for intra prediction mode is excluded from the proposed PDec for following reasons. Firstly,

in HEVC standard, decoding syntaxes of transform units in CABAC relies on the result of intra prediction mode for current coding unit. If intra part is encapsulated in PDec, a long feedback path inside the whole decoder is generated, leading to performance degradation. Secondly, algorithm for intra prediction is simpler than MV's and it doesn't need the reference data which is locates in different coding tree unit (CTU) rows. Therefore, the overhead for adding intra part into CABAC is negligible compared to MV and BS calculation. The rest of this chapter gives a brief overview for MV and BS calculation in HEVC. Then the design consideration is given followed by the previous works.

### 2.1.1 Motion Vector Decoding

The process of calculating motion vector is to decode syntax elements into motion parameters, which can be directly used by the following motion compensation module. A block's motion parameters have high possibility to be similar to spatial or temporal neighboring. In HEVC, irregular coding algorithm is employed to eliminate such kinds of redundancy for compression efficiency.

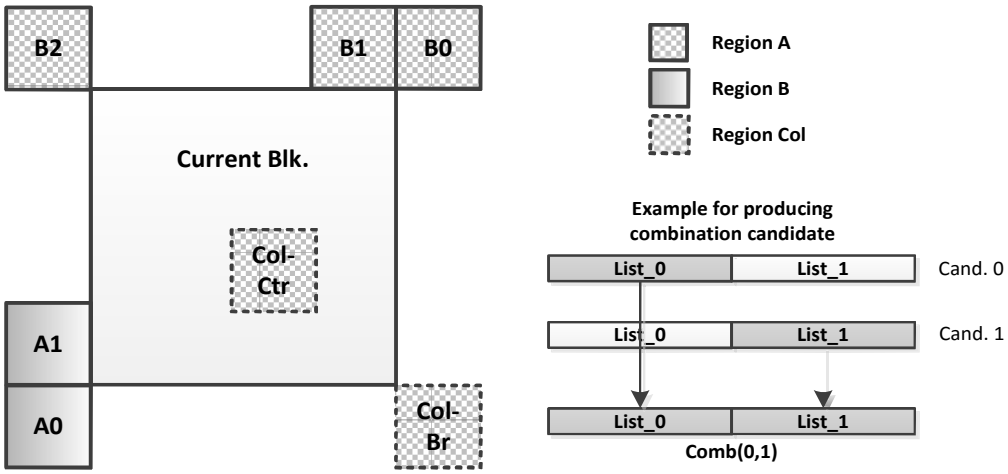


Fig. 17 Candidate selection in MV decoding

As is introduced in Section 1.1.2, Advanced Motion Vector Prediction (AMVP) mode and merge prediction mode are employed by HEVC for coding MV parameters. Both of them require prediction parameters of five spatial neighboring blocks and two temporal co-located blocks as input, as depicted in Fig. 17. Besides, each of the two

prediction modes owns its unique algorithm and resource sharing between them is limited.

In AMVP, all seven reference blocks are categorized into three regions, A, B and Col regions. Each region will produce at most one candidate so that a list of at most three candidates can be constructed in general. If the neighbors in the left are not available, two motion vector candidates are derived both from the above side, which is the region B. After removing the identical ones in the list, I will add zero MV candidates into the list. In the last step, I will also try to remove the MV candidates whose index is larger than 1. Then, the final MV will be selected by the syntax mvp_lx_flag.

In merge mode, motion parameter decision starts with constructing merge candidate list. Firstly, valid blocks are pushed into the list in the order of B0, A0, A1, B1, B2 and Col. Up to four spatial neighbours and one temporal neighbour will be pushed into the candidate lists if they are available. For the spatial candidates, I also have to check partition redundancy removal so that I will never merging 2 2N×N blocks into one 2N×2N block. If the candidates are fewer than five in default, the combined candidates will be assembled with the content of candidate list and added into list with the valid candidates in the first step if the current slice supports bi-prediction. Finally, zero candidates with different reference frames are produced if the list is not full. After the list is constructed, merge MV result is chosen by the merge_idx from the list. Note that in each mode, the scaling operations will be processed when there is a difference between reference frame of current block and that of reference block [12].

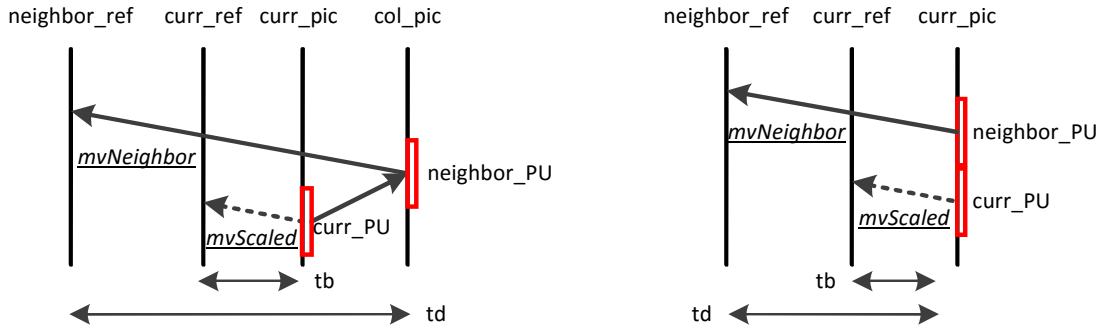


Fig. 18 Scaling calculation when different reference frames are chosen

Scaling calculator is an operation frequently used in AMVP mode. It is also used

for the temporal candidates in the merge mode. Fig. 18 shows two cases where scaling calculator is utilized when tb!=td. (tb & td represent POC differences, Left: neighbor_PU is temporal; Right: neighbor_PU is spatial). In the figure tb (td) indicates the POC difference between current (neighbor) picture and its reference picture. When tb and td are not the same, mvNeighbor can't be directly used as the prediction of motion vector and it must be scaled. Therefore, mvScaled should be deduced by following equations defined in HEVC specification [12].

```
tb = Clip3( -128, 127, DiffPicOrderCnt( curr_pic, curr_ref ) )
td = Clip3( -128, 127, DiffPicOrderCnt( neighbor_pic, neighbor_ref ) )
tx = ( 16384 + ( Abs(td) >>1 ) ) / td
distScaleFactor = Clip3( -4096, 4095, ( tb * tx + 32 ) >> 6 )
mvScaled = Clip3( -32768, 32767, Sign( distScaleFactor * mvNeighbor ) *
                ( ( Abs( distScaleFactor * mvNeighbor ) + 127 ) >> 8 ) )
```

Fig. 19 Scaling algorithm in HEVC when reference frames are different [12]

## 2.1.2 Boundary Strength Decoding

Boundary strength is used in de-blocking filter for filter selection. The calculation of BS can be divided into two steps. The first step is to prepare necessary data. Generally, I have to fetch the MV parameters of current block and all the adjacent neighboring blocks in the left and top. Then in the second step, specific algorithm is used to produce the BS result by comparing MV parameters between current block and neighboring blocks.

BS calculation will be executed on all the prediction unit (PU) and transform unit (TU) edges at 8×8 block grid. Although the BS values are calculated at 4x4 block basis, but the final result is re-mapped to an 8×8 grid by choosing the larger BS value from the two 4×4 grid edges. Let P and Q be the two blocks beside a certain edge. The full algorithm for BS calculation is shown in Fig. 20. If P or Q is intra prediction, BS is equal to 2. Otherwise, PU and TU edges have different algorithms to produce BS value. The algorithm for TU edges is quite simple, which just check whether P or Q has non-zero coefficients. In contrast, the algorithm for PU edges are much more complex.

Almost all the motion information has to be compared between P and Q blocks, such as the number of reference frames, the number of MV and the MV difference [12].

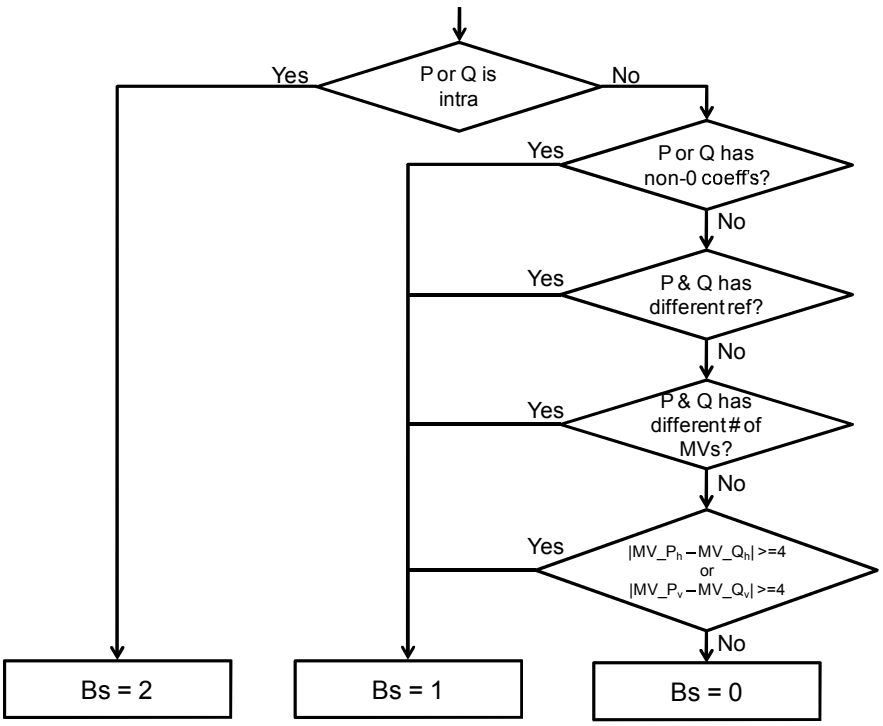


Fig. 20 Flow diagram for BS calculation [12]

## 2.1.3 Block Merging for Data Reuse of Motion Vector

To design a parameter decoder for HEVC and 8K UHD applications, there exist several challenges. Firstly, since high video resolution directly increases the burden on system throughput because of the huge data volume, higher throughput requirement is needed. Assuming the clock frequency is 250MHz, for 7680x4320@60fps, a throughput of at least 8 pixel/cycle has to be achieved. On the other hand, new HEVC standard introduces complicated coding tools in MV and BS calculation to achieve better compression performance, such as Advanced Motion Vector Prediction (AMVP) mode and merge mode. Flexible quad-tree CU structure is also introduced in HEVC. All these new coding tools lead to challenges on memory bandwidth requirement, data dependency and throughput requirement for high-performance VLSI implementation [13].

I conclude above design challenges as energy efficiency issues, including both

compute energy and memory access energy. Compute energy in PDec means the design costs to support the complicated algorithm caused by HEVC. Memory access energy is mainly introduced by the high resolution of 8K UHD applications where more pixels per second have to be decoded.

To solve the energy issue, block merging is proposed to consider the data reuse inside the parameter decoder. HEVC processes each video frame based on block in Section 1.1.1, which means that the same MV is used for all the samples inside each block as shown in Fig. 21. This is called the reuse of MV. To optimally use this reuse, the idea is that the calculation for MV only needs to be done once per block. By doing this, the number of calculation times and memory accesses can be minimized. This is ignored by previous works so that the block with the same MV are divided into small blocks. Block merging is to merge these small blocks for energy efficiency. To achieve this goal, the dataflow is first given in Section 2.2 and the related memory hierarchy is given in Section 2.3 to further exploit the temporal data reuse inside the dataflow.

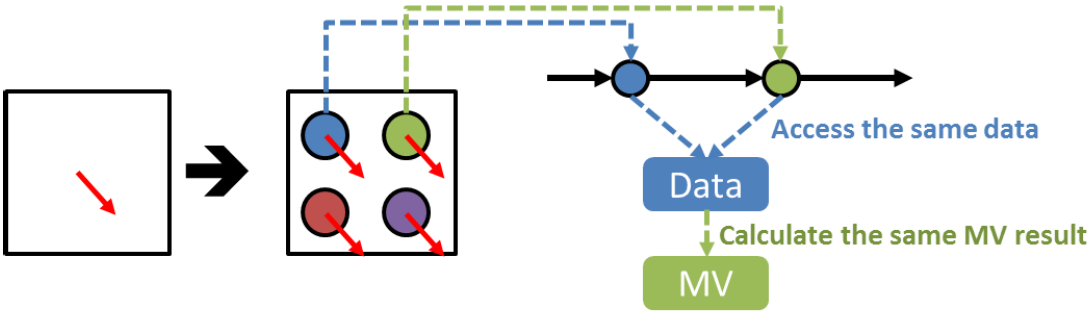


Fig. 21 Reuse of motion vector for block based process in HEVC

Therefore, this dissertation discusses the design consideration of VLSI a

-process is needed. Data and control flow in pre-process is characterized an irregularity because these processes are highly input-dependent and evolve with the algorithm itself. 3) The pre-process for decoding syntaxes into parameters has less dependency with core calculation in MC and de-

blocking filter (DBF). Further, BS calculation relies on the result of current block's MV, which inspires us to build unified PDec architecture to share on-chip memory without much extra control overhead.

### 2.1.4 Literature Review

Parameter decoder is named as many previous works unified other parameter decoding with the motion vector decoding. Two representative previous works by K. Yoo [17] and J. Zhou [19] are introduced. These two works are both for H.264 while there is no research result for HEVC designs.

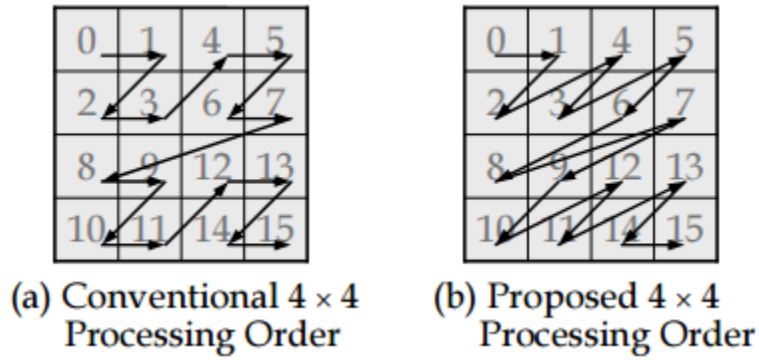


Fig. 22 The proposed novel processing order in [17]

K. Yoo [17] introduced a VLSI architecture design for decoding motion vectors in H.264/AVC. The design challenges in this work are the complex derivation process of motion data. In detail, the algorithm for motion vector prediction copes with the various macroblock partition and spatial/temporal neighbours. To address these design challenges. The macroblock is first divided into small 4x4 constant blocks and they are processed sequentially. Three main proposal were mentioned. Firstly, it classified the complex derivation process of motion data into three basic modes, regular, spatial direct and temporal direct. They area efficiency was claimed by this design. Secondly, a novel processing order was presented as shown in Fig. 22. The motivation is to reduce the dependency for enhancing the throughput. For example, block with index 4 only relies on the motion data of block 1, which means block 4 can start decoding after block 1 instead of waiting for the finish of block 3. By doing this, up to 36 and 72 cycles can be reduced for P frames and B frames. Thirdly, the complicated pipeline design was

achieved for the various macroblock partitions. Three deterministic processing loop control scheme can handle all possible cases for simplify the designs.

J. Zhou [19] proposed a unified parameter decoder for H.264, including motion vector, intra prediction and boundary strength. Unified architectures for multiple parameter decoding can share the on-chip memory by careful designs to reduce hardware costs. Targeting at 4K@60fps throughput, the main challenge was the support of high throughput and low memory bandwidth requirement. To further simplify the control logic compared to [17], a 64 cycle per macroblock pipeline was proposed to increase the throughput and reduce the design costs. The implementation is to support two basic pipeline block sizes, 8×8 and 4×4, with fixed cycle counts. Moreover, a partition based storage format is applied to condense the macroblock level data. The main idea is to use a simple entropy coding scheme, variable length coding, for compressing the motion data.

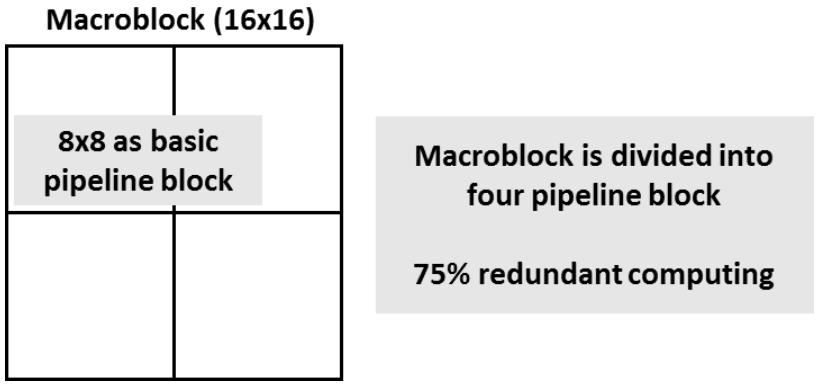


Fig. 23 An example of proposed solution for various macroblock partition [19]

However, both works are for H.264 and their designs cannot be employed for HEVC with minor revision. Firstly, both of these two works tried to solve the various macroblock partitions due to its high complexity and design costs. In HEVC, this problem becomes more and more complicated as discussed in Section 1.1.1. The conventional methods chose the minimum block size as the pipeline granularity. When a larger block is given, it will be divided into small blocks, resulting in redundant computing because all the motion data inside the large block is the same. An example is shown in Fig. 23. Secondly, neither of the previous works has a deep discussion on

the memory hierarchy designs. The reason is that the throughput requirement for them is not intensive. In [19], 4 cycles are allocated for a 4×4 block, which only require three spatial neighbours and one temporal neighbour. Therefore, no specific hardware design optimization is needed. However, I mentioned that the throughput requirement of my research target is about 8 pixels/cycle, equivalent to 4 cycles for a minimum 4×8 or 8×4 block in HEVC. For this block size, 5 spatial neighbours and 2 temporal neighbours have to be fetched in 4 cycles. A novel memory hierarchy is required to support the intensive memory accesses. Finally, the algorithm in HEVC is also complicated. I have introduced the AMVP and merge modes in Section 1.1.2. Considering the design costs in previous works, a low-cost and energy efficient architecture for HEVC algorithm should also be explored.

Besides, several parameter decoder approaches are reported previously. In [14] an approach for MPEG2 is achieved. For H.264/AVC, Xu, et al. [15] first shows a solution for QCIF format. Higher throughput is also achieved in [16][17][18]. However, these realizations have much room for optimization. The fastest throughput among them is 260 cycles/MB, which is far slower than the throughput requirement for 8K UHDTV. In Zhou et al.'s work [19], a joint parameter decoder with fixed pipeline granularity for 4K@60fps UHDTV application is proposed for MV, BS and intra prediction mode. Considering this H.264/AVC approach is not applicable for HEVC and the technique can't support high throughput requirement for 8K video, I can't directly inherit [19] for HEVC solution. Except the VLSI works mentioned above, FPGA-based implementation in [20] is also reported. However, FPGA based work is too difficult to be extended for 8K UHDTV application. Many works for video decoder designs does not involve the realization for MV decoding as they are not working on the design of the whole system, while it is a tough work in HEVC for hardware implementation.

## 2.2 Unified CU-adaptive Pipelined Dataflow

This section shows the top level dataflow of the proposed PDec architecture. The unified CU-adaptive pipeline is proposed to support high throughput of up to 8K

application with reasonable implementation complexity. In this section, I also show how data is reused in this dataflow as well as maintaining the data reuse potentials between pipeline granularities.

### 2.2.1 Dataflow Analysis

Dataflow design highly affects the performance of PDec. As there are diverse block types supported in HEVC, the design of dataflow is complicated if we design specific hardware behavior for every possible block types which is around 24 types in HEVC because the algorithm for MV calculation highly related to the block types. In Fig. **24** a simple case is shown for this problem. Case 1 is processing a rectangular block so that A0 is in the left decoded block. In this case, all the spatial neighboring blocks can be used for the calculation. On the other hand, case 2 shows the current block is a square block. The spatial neighboring block A0 is in the region which has not been decoded yet. Therefore, HEVC algorithm will only use another four spatial neighboring blocks for calculation. Therefore, two different hardware processor cores (core 1 and core 2 in Fig. 24) are required for each case. This simple case proves that different block types will highly affect the hardware behavior including memory access pattern and process time of the algorithms. The hardware architecture should handle all 24 block types in HEVC, which may lead to a high hardware costs.

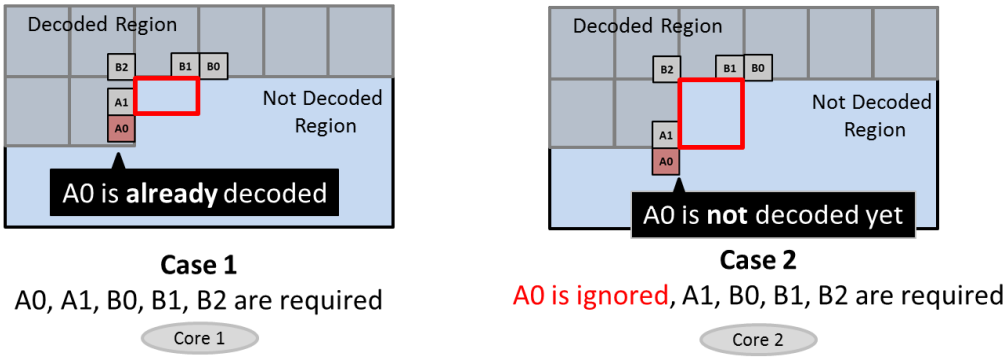


Fig. 24 A case shows the design complexity for 24 block types in HEVC

To handle the diverse block types in HEVC, I conclude there are several approaches like diverse block approach, constant block approach and the proposed CU-adaptive block approach.

#### *2.2.1.1 Variable block approach*

A direct approach is to process the diverse block types as they are. An example is shown in which contains 6 blocks. In this approach, no matter what the size and shape of the current block, the hardware is able to process it optimally by using a dedicated processor cores.

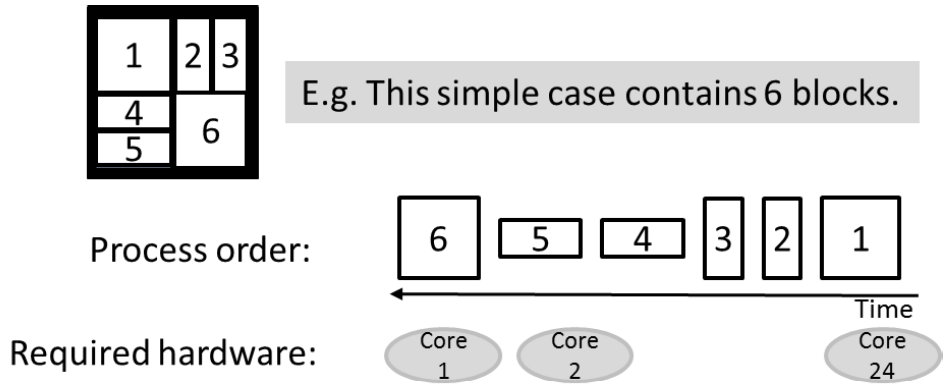


Fig. 25 An example of variable block approach

The merit of this approach is that it ideally maintains the reuse potentials of motion vector. Each block will only be calculated once with its corresponding processor core. Therefore, a high throughput and few memory accesses can be expected by this approach.

The demerit of this idea is also obvious. 24 different processor cores are required to guarantee the functionality with a high complexity.

#### *2.2.1.2 Constant block approach*

Another approach is called constant block approach whose basic idea can be found in many previous works introduced in Section 2.1.4. Given any block type, it is first divided into small constant blocks like 4×4 and the hardware is designed to only support the process of this 4×4 block for simplification. The basic idea is shown in Fig. 26.

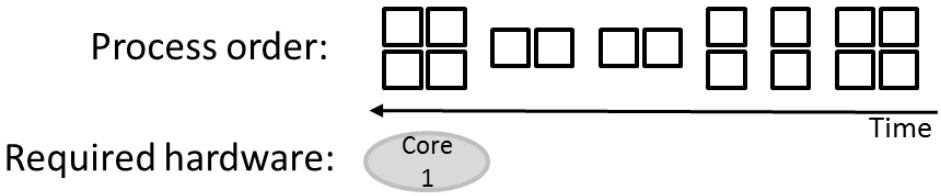


Fig. 26 An example of variable block approach with the same case in Fig. 25

The merit of the constant block approach is that the hardware design is significantly simplified. Only one processor core is sufficient. However, this approach is only suitable for H.264 because the division of blocks and the process based on divided blocks are supported in H.264 [19] while HEVC doesn't have this function.

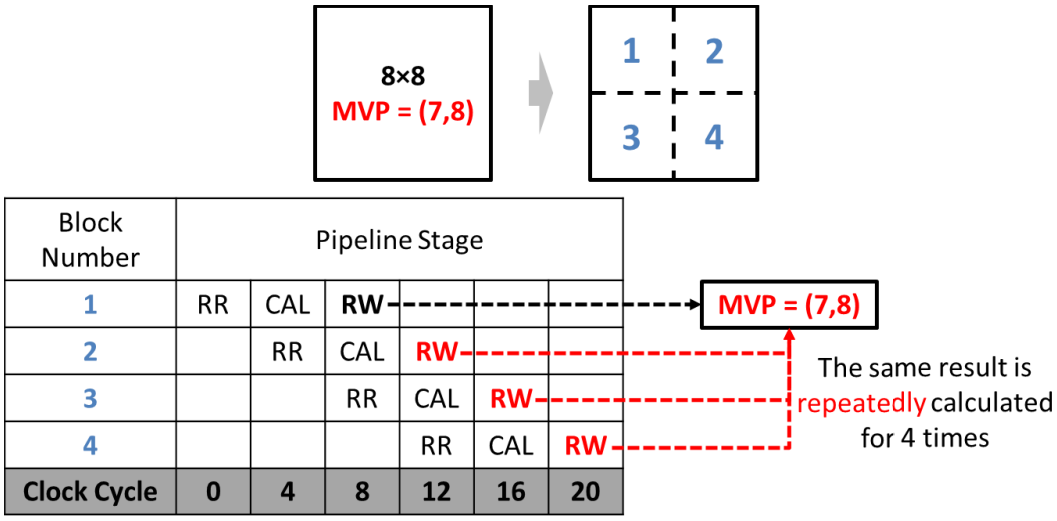


Fig. 27 Constant block pipeline design introduces redundancy.

The demerit of this solution is that it will introduce many redundant computations and cost more processing cycles. An example is given in Fig. 27. A 8×8 block is divided into four constant blocks. Each block is pipelined with three stages reference read (RR), calculation(CAL) and result write (RW). All of the four constant blocks contains the same information so this information is repeatedly calculated four times. This redundancy is accepted by the previous work because the research is only for 4K throughput and H.264 standard. Even if the redundancy exists, the throughput is still enough for their research target. However, the situation becomes worse for our target of 8K and HEVC. Firstly, HEVC supports the maximum block size of 64×64. If the

constant block pipeline is still employed, there will be up to 95.5% redundant computations when processing this 64×64 block. Secondly, the required throughput increases to four times than [19]. The designs in [19] is not capable for the high throughput of 8K, especially considering the high redundancy in the previous work. Therefore, in order to support high throughput requirement and HEVC standard, the pipeline design has to be well designed which involves three aspects, pipeline granularity, pipeline processing stages and cycle resource allocation.

#### *2.2.1.3 Proposed CU-adaptive block approach (Unified CU-adaptive Pipelined Dataflow)*

According to the above discussion, I t

In

HEVC, pixels in one coding unit share the same prediction mode (inter or intra), pixels in each PU share one MV. On the other hand, 4-pixel-length edges on 8x8 grid share one BS for deblocking filter. By combining these two processes, it's better to define the pipeline granularity as 8x8 blocks or larger. In previous works, fixed pipeline granularity is adopted. Though pipeline controller for that is simple, the throughput will be decreased in HEVC. For example, in [19], fixed 4x4 or 8x8 pipeline block is designed. However, HEVC supports maximum 64x64 block size. If fixed 4x4 pipeline granularity is used, large redundant calculation directly degrades the performance (for example, one 64x64 prediction unit needs up to 256 4x4 pipeline blocks, even if pixels in 64x64 block share one motion vector).

To overcome the redundancy problem, I propose the CU-adaptive pipeline granularity. This approach is to adaptively process blocks based on CU sizes as shown in Fig. 28 with the same case used in Fig. 25 and Fig. 26. The basic idea is to merge PUs into the nearest CU and process CU as it is. By doing this, we also keep the reuse potentials of motion vector because no division of blocks occurs in this approach so that better energy efficiency can be expected. Meanwhile, the design overheads is increased compared to the constant block approach because we should handle 4

different CU types in HEVC. However, it still reduces lots of design costs compared to the 24 block types in the variable block approach.

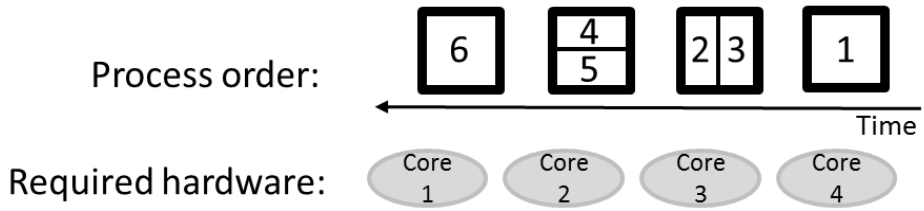


Fig. 28 An example of proposed approach with the same case in Fig 29

In detail, pipeline granularity is coding unit. Though the size and partition mode are various for different coding unit, the CU-adaptive pipeline can still have several advantages. Firstly, compared to the previous fixed pipeline blocks, the CU-adaptive pipeline scheme can efficiently eliminate the redundancy for larger CU cases, which is considered to happen frequently in 8K UHDTV. Secondly, instead of PU based pipeline, the CU-adaptive pipeline scheme can reduce the implementation complexity for pipeline controller. The reason is the proposed scheme can reduce the number of possible cases for pipeline granularity, especially for asymmetrical partitions. Thirdly, in spite of various sizes, the process for each CU follows similar procedure, leading to easier implementation. Finally, even if CU contains two or more PU, neighboring blocks for each PUs have overlap, which means that data reuse can be utilized for memory bandwidth reduction, as shown in Fig. 29.

The left picture in Fig. 29 shows the regular case where a block is not divided into two prediction units. Five spatial neighbors are required. When this block is partitioned into two units like the right picture of Fig. 29, the B1 of left block and the B2 of the right block will be overlapped so that I don't need to fetch them twice. Moreover, I have the preliminary that A0 of the right block is always not available. The A1 of the right block is the result of the left block. In order to prevent 2N×2N partition emulation, this A1 will also be regarded as not available. In conclusion, I originally have to fetch ten spatial neighbors for these two blocks. With the CU-adaptive pipeline design, only seven neighbors are required, with 30% percent of memory reduction for motion data prediction.

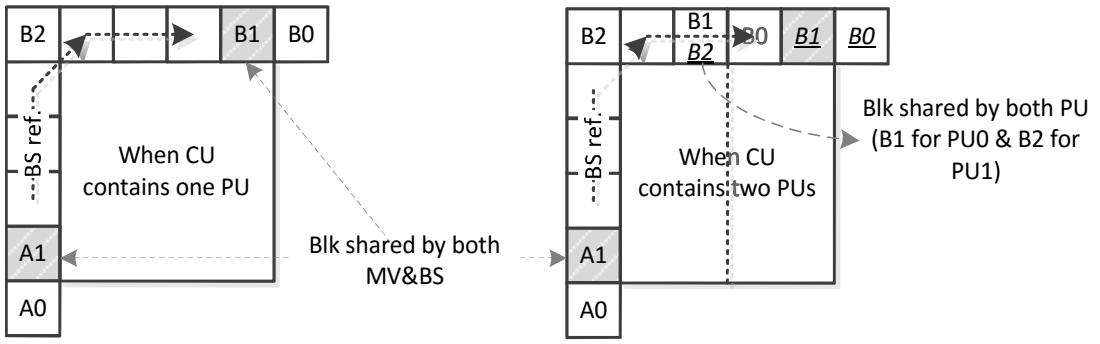


Fig. 29 Data reuse in MV and BS reference fetching order

The CU-adaptive pipeline granularity also brings advantages on data sharing for unified motion vector and boundary strength decoding. As is mentioned in Section 2.1.2, boundary strength is calculated for each block edge. The involved operands for this calculation are the motion data from the two blocks, which contain this edge. In my architecture, I calculate the left and upper edges of each block. With the default Z-scan order, this procedure can finish all the edge calculations for boundary strength. In such a case, the left and upper neighbor blocks have to be fetched from memory. Notice that the boundary strength is calculated on the grid of 4×4 so that the every 4×4 neighbor block should be fetched separately. Compared to the five spatial neighbors involved by motion data prediction, A1 and B1 are always reused by both motion data prediction and boundary strength prediction, as shown in the left side of Fig. 29. This saves memory bandwidth requirement greatly, especially for small block size like 8×8. For an 8×8 case, 25% of memory accesses for boundary strength can be saved. When this block is divided into two prediction blocks, up to 50% of the memory accesses can be saved for both accelerating process throughput and saving memory energy.

A simplified data flow in Fig. 30 illustrates that three pipeline processing stages are designed for unified PDec. Considering BS calculation needs the result of current block's MV, so BS calculation should be scheduled after the correspondent process for MV. Thus, by incorporating the two processes together, I define the unified pipeline as consisting of three main steps: 1) memory reading for reference data fetching, 2) MV calculation for current CU and 3) BS calculation and MV writing back. Notice that only BS calculation for PU edge is in stage 3 while TU edge is done outside pipeline. The

reason is that BS calculation for PU edge require MV information so it ties tightly with the rest of CU-adaptive pipeline. Meanwhile, BS for TU edge is quite simple so that it will not increase the hardware complexity much by separating it from PU edge.

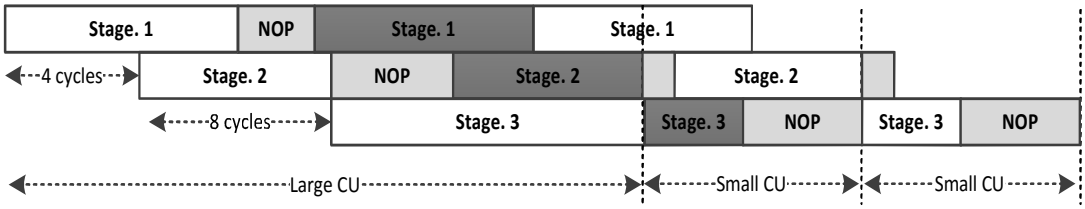


Fig. 30 Trailing NOP realization in the CU-adaptive pipeline

Table 6 shows the cycle resource allocation for each stage and for each pipeline granularity. Generally, a pipeline's throughput is mainly decided by the slowest stage. Hence the performance will not be degraded so much if I decelerate the faster stages. Thus I unify the process speed of each granularity as the slowest stage among all three stages, which is shown in the Max. Cycles column in Table 6. Notice that the data in brackets indicates that a CU contains two PU. It is achieved by adding no operation (NOP) cycles in the trailing of fast stages. The merit is the implementation for pipeline controller is greatly simplified. The throughput for each CU sizes are also listed in the table, which proves that the proposed PDec is competent for at least 8 pixels per cycle.

Table 6 Cycles allocation for different CU cases

| CU Size | Process Cycles for Each Stage (Cycles for CU contains 2 PUs are shown in brackets) | | | Max. Cycles for each CU | Throughput (Pixel/cycle) |
|---|---|---|---|---|---|
| | Stage 1 Memory Read | Stage 2 MV Calculation | Stage 3 BS Cal. Mem Wr. | | |
| 8×8 | 8(8) | 4(8) | 3(4) | 8(8) | 8.0(8.0) |
| 16×16 | 10(12) | 4(8) | 7(10) | 10(12) | 25.6(21.3) |
| 32×32 | 18(20) | 4(8) | 15(22) | 18(22) | 56.9(46.5) |

| | | | | | |
|---|---|---|---|---|---|
| 64×64 | 34(36) | 4(8) | 31(46) | 34(46) | 120.5(89.0) |

If I further analyze the cycle allocation of each pipeline stage in Table 6, I may notice that the complexity of HEVC is much higher than H.264. In the previous works mentioned in Section 2.1.4, the bottleneck of the pipeline design is the calculation stage. They have paid many attentions on simplify the pipeline designs. However, Table 6 shows that the bottleneck stage is varying according to the block sizes. For a small block size like 8×8, the slowest stage is the memory reading and calculation stage (only when the coding unit is divided into two blocks). In such cases, the CU-adaptive pipeline can enhance the data reuse for saving memory access cycles. On the other hand, the bottleneck stage may change to memory write stage because the boundary strength should be written back to memories for the usage of next block. For these cases, the dependency issue may happen due the pipeline designs. Next block may use the current block's result while the result has not been written into memories. Some bypass circuits should be design for this issue.

## 2.2.2 Block Diagram for the CU-adaptive Pipeline

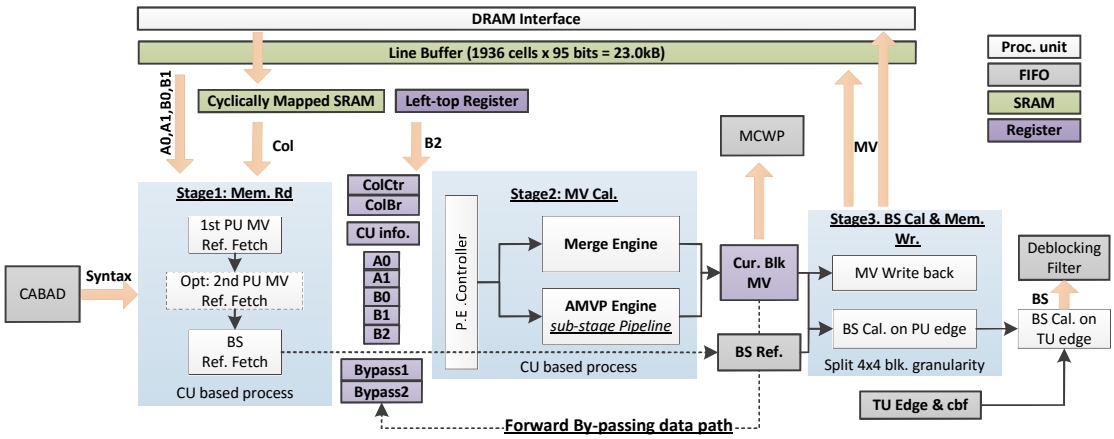


Fig. 31 Block diagram for the CU-adaptive pipeline

Fig. 31 shows the whole framework of unified parameter decoder. The below part illustrates the data flow inside the proposed CU-adaptive pipeline. Inside each pipeline stage, important process units are depicted. The detail for these three stages will be given in the following sections correspondingly. The upper part gives a brief

representation for memory in this design, which will be discussed in Section 2.3. Based on the pipeline analysis, no matter what the current and next CU's partition mode and size are, they can be processed continuously without pipeline pause to guarantee high pipeline performance.

### 2.2.3 Reference Data Fetching

The first stage is in charge of fetching neighboring data from memory, which will be explained in Section 2.3. Generally, each PU in CU needs four cycles for fetching MV reference blocks' information, as shown in Fig. 30. When current CU contains more than one PU, only extra two cycles are needed for second PU as the rest of the neighboring blocks can be reused from first PU. After finishing MV reference fetching, BS reference blocks are accessed and then pushed into FIFO in preparation for stage 3.

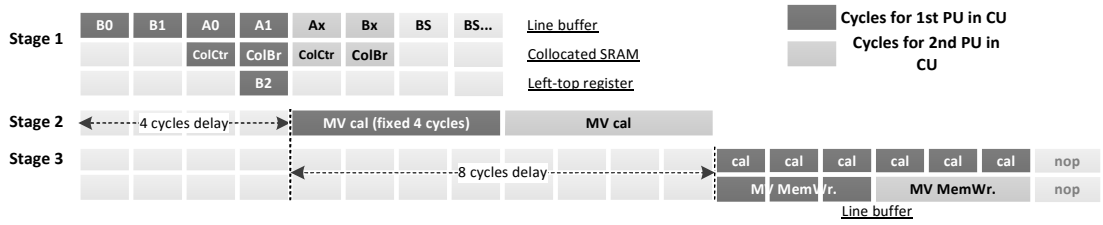


Fig. 32 Cycle resource allocation in the CU-adaptive pipeline

Three memory controllers work in parallel as shown in Fig. 32. Rather than sequential process, these controllers in parallel is to improve the throughput in worst case. The worst case is defined that CU size is 8x8 and partition mode is symmetric partition (two PU in this CU). For each PU, five spatial neighboring in line buffer and 2 temporal neighboring in collocated SRAM need 7 cycles to be fetched. Extra 3 cycles are for fetching BS neighboring blocks. Thus, totally 20 cycles (ten for each PU) are required for this worst case. Under this situation, the performance can't even support an 8K@30fps video application. Therefore, I propose three memory controllers working parallel. Together with the data reuse scheme in Fig. 29, the design can achieve throughput for 8K@60fps.

## 2.2.4 Proposed Index Mapping Optimization

The algorithm-irregular MV calculation (AMVP mode and merge mode) is schedule in Stage 2. The implementation suffers from huge area cost and timing issue. Hence, I propose resource sharing scheme and index-mapping scheme for optimization.

Firstly, MV scaling calculator is optimized and reused. The data path of scaling calculator is depicted in Fig. 33. In this procedure, one division, two multiplications and several adders are employed. Without optimization, the synthesis result reports 7ns critical path and around 10k gates cost. The critical path is marked as read in Fig. 33. Such serious costs in both timing and area are unacceptable for an efficient design. Sub-stage pipeline is proposed inside stage 2 for scaling calculator. Four sub-stages are designed in Fig. 33, which are segmented by dotted line. By doing so, a balanced division for critical path is achieved. Meanwhile, LUT replaces the division implementation in the sub-stage 2 for area and timing saving. The proposed sub-stage pipeline is capable for working under 2.5ns timing constrain. As one scaling calculator is capable to process one motion vector, two calculators are necessary for scaling two motion vectors in a bi-directional prediction case. Thus the optimized architecture for scaling calculator can achieve 400MHz timing constrains.

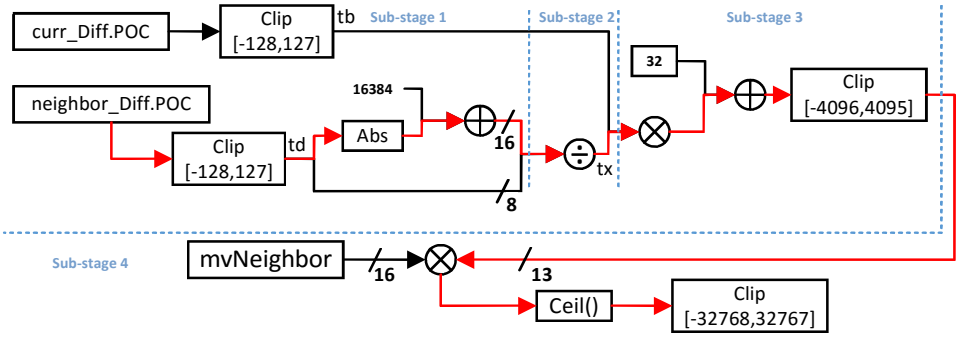


Fig. 33 Data path of scaling calculator and the scheme for sub-stage pipeline

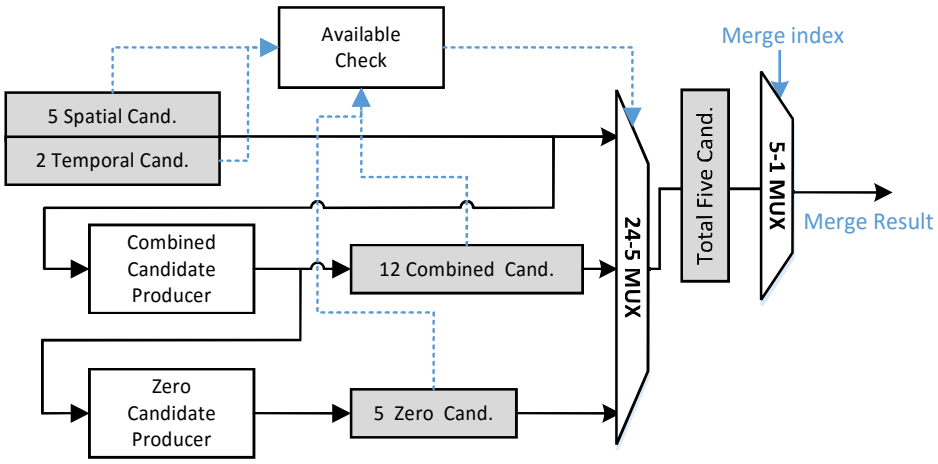


Fig. 34 Direct implementation without optimization

Secondly, index-mapping scheme for merge mode engine is proposed. Candidate Producer in Fig. 34 follows the same procedure and accomplishes the identical function for merge mode in HEVC standard. Taking Combined Candidate Producer as an example, I receive available candidates (OrigCand) from 5 spatial and 2 temporal candidates. If the number of received candidates is less than five (assuming n), then at most (5-n) combined candidates (CombCand) will be produced. The procedure for combined candidates is illustrated in right-bottom in Fig. 17. Two motion vectors for two lists of combined candidate are copied from different OrigCand to construct a combined one. Up to 12 kinds of combinations are employed in HEVC specification, leading to 12 different combined candidates. Zero Candidate Producer produces candidates (ZeroCand) when the number of OrigCand and CombCand is still less than five. Up to five ZeroCand may be used by merge mode.

For hardware implementation, candidate list whose depth is five is first generated from 24 possible candidates mentioned above. Then the final merge result is chosen from this list by merge_index. However, a 24-5 MUX with 95-bit in Fig. 34 for each entry has to be utilized, which leads to huge area cost.

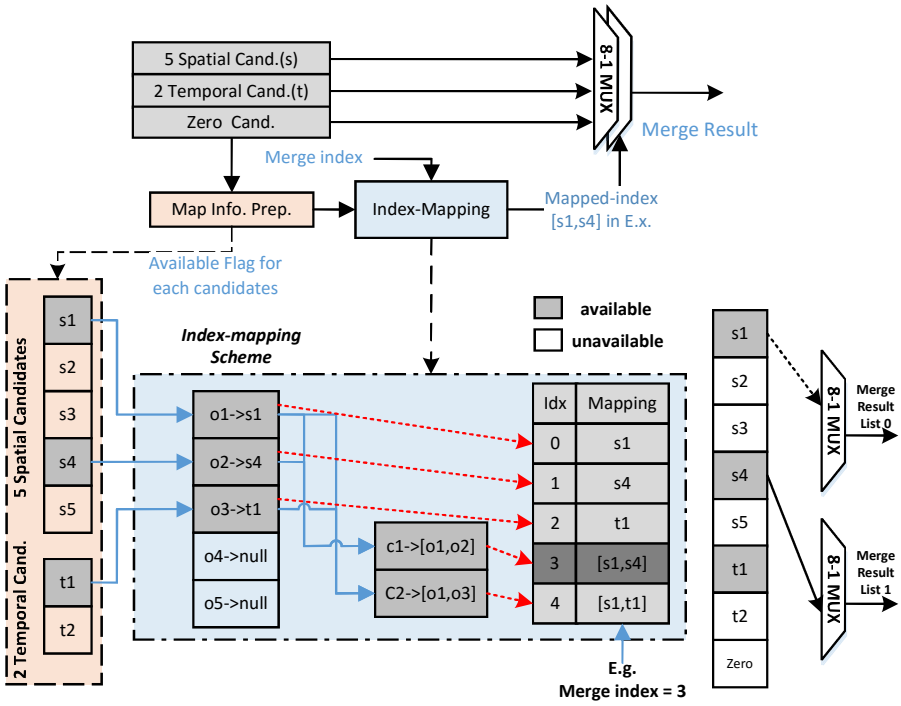


Fig. 35 Index-mapping scheme

Index-mapping is an optimization to reducing area cost. The motivation is I notice the motion vectors of final result can only have two cases: copy from inputs or zero MV. Assuming zero MV as a special input, I can directly copy the input to the final result as long as I construct a mapping relation between them. As is shown in the Fig. 35, Index-mapping Scheme receives little data volume such as available flags as inputs, and five candidates in the final lists are mapped to the original inputs. Based on the merge_index and the mapping results, the merge result can be directly copied from the original inputs.

For hardware implementation of Index-mapping scheme, final motion vector is copied from seven original inputs (5 spatial candidates and 2 temporal ones) or zero MV. Considering bi-direction prediction containing two motion vectors, only two 8-1 multiplexers are required, which saves area cost a lot compared to the original design.

## 2.2.5 Memory Write Back

This stage is in charge of calculating boundary strength parameters and writing motion data result back into memory. BS calculation relies on information of prediction unit and transform unit, as is shown in Fig. 31. Split 4x4 pipeline scheme inside stage

3 is proposed for pipeline buffer reduction. The CU-adaptive pipeline is proposed in the top level of PDec design. However, when CU size is 64x64, totally 32 (64 / 4 × 2) neighboring 4×4 blocks should be buffered before the start of BS calculation. I notice that BS calculation is executed 4×4 by 4×4, thus a further separation for pipeline granularity is feasible. By applying split 4×4 pipeline scheme, only 8 neighboring blocks (the delay between stage 2 and 3) are needed to be buffered, which helps reduce 75% of buffering memory.

Process speed is analyzed for split 4x4 pipeline scheme. In stage 3 only left and top edges of a PU is calculated to get an internal BS, which will be further refined by TU information as shown in Fig. 31. For example, if a CU is completely a PU for a 64x64 CU, 64/4 = 16 4x4 blocks on PU left edge needs 16 cycles while another 16 cycles are for the PU top edge. Considering left and top PU edges have a 4x4 block overlap, totally (16+16-1) = 31 cycles are required. 46 cycles are for the case where CU contains two PU so that one more PU edge inside CU will further cost another (16-1) = 15 cycles compared to the previous case, as is shown in Table 6.

Notice that I propose a dedicated data path for bypassing the current motion data results by to stage 2, as shown in Fig. 31. This is to solve the data dependency problem existing in both motion data prediction and boundary strength decoding. Both calculations require the left and top blocks' results as their prediction candidates. Due to the pipeline design, the result of previous block may be required by the current block for calculation while it has not been written back into memories. If no special consideration is done for this issue, the fetched data will be a dirty one, resulting in the wrong results. Therefore, I bypass the block's result in stage 3 back to stage 2. When blocks in stage 2 are fetching neighbors, the circuits will check whether a required neighbor is the one in stage 3. If so, the data will be fetched through the bypass data path. As the delay between stage 1 and stage 2 is fixed as 4 cycles, there will be only one set of result needed to be bypassed to stage 2. This design eliminates the potential hazard caused by the data dependency.

### 2.2.6 Analysis of Design Overheads

In the above discussion, I have introduced the advantages of the proposed unified CU-adaptive pipelined dataflow. This section will give a discussion about the design overheads for achieving this proposal.

I conclude that three technical problems have to be solved which will decrease the area efficiency of the proposed architecture: (1) Hardware should support four different CU block types; (2) Hardware should handle the cases where the second CU may not exist; (3) Due to the pipeline design, the data dependency is introduced. I will explain these three problems in detail one by one.

The first design overhead comes from the more block types that should be handled by the hardware architecture compared to the previous work [19]. If a finite-state machine is used to describe the hardware behavior with two states, idle and busy. The problem is that the busy cycles vary from CU to CU and from stage to stage as shown in Table 6. Although I have chosen the slowest stage as the allocated cycles to simplify the hardware design, extra hardware costs are still required for the implementation. In detail, I implement the fifth column as a pre-defined look-u

-u

In total, at least a look-u

In the real implementation, I use an idea called data gating to solve this technical challenge. In detail, a sophisticated finite-state machine is used with a state called data gating as shown in Fig. 36. The states of Block 1 and Block 2 are in charge of normal processing for the first and second block inside a CU. Like the clock gating which shut off the clock when necessary, the state of data gating is to latch the primary input registers of a module. By doing this, the input registers are disabled for data toggling so that all the related combinational circuits can avoid the

unnecessary data toggling. This can save the dynamic power while complex hardware is required like more enable signals for latching the input registers.

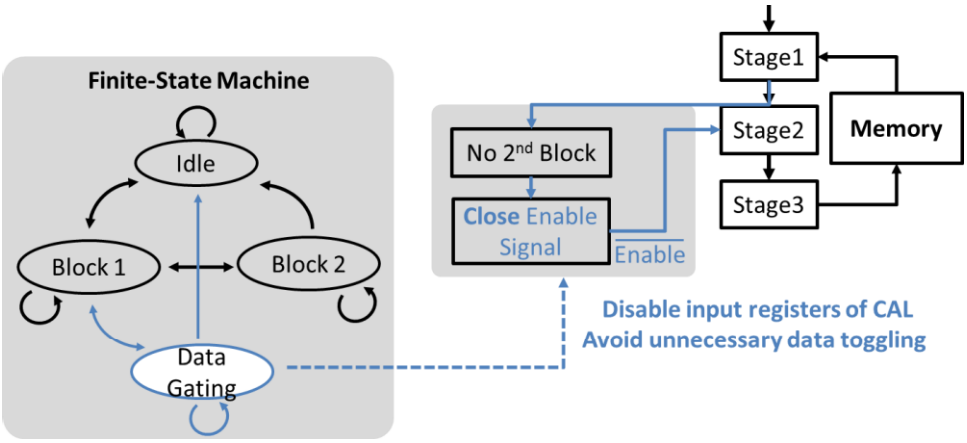


Fig. 36 Data gating for skipping the process when 2nd block doesn't exist

The third design overhead is introduced for solving the data dependency problems due to the usage of pipeline technique. An example in shows the problem of data dependency where Block B will use the result of Block A. The stage 1 may fetch old data from the memory because the results of Block A is not written back into memories yet. This will decrease the pipeline performance. Therefore, data bypass technique is used for propagating the data as soon as they are available. For example, the result of Block A in Fig. 37 is available as soon as the stage 2 finishes its process. Therefore, additional data paths are introduced to send the results to the calculation for the next block in advance. Moreover, extra multiplexer is also required so that stage 2 can decide whether to receive data from stage 1 or from the bypass paths. In total, in order to eliminate the pipeline stall, extra data paths inside the circuits are required.

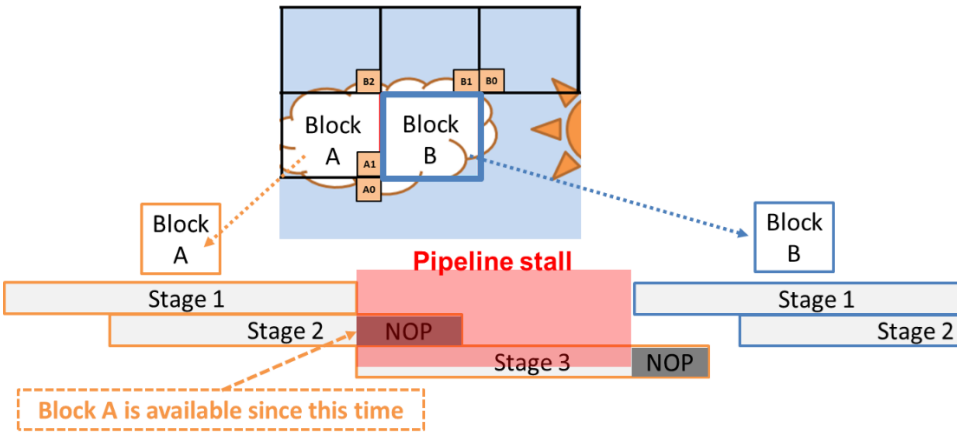


Fig. 37 Pipeline stall due to the data dependency issue.

In conclusion, the high performance of the proposed CU-adaptive pipelined dataflow is realized at the expense of more complex hardware architectures for solving the above three technical problems.

## 2.3 Multi-level Memory Hierarchy Design

Parameter decoder involves fetching neighboring blocks as the reference of current decoding block. These reference data are usually reused in a short future. I call it temporal data locality in PDec. In previous work [19], these data are stored in the off-chip DRAM regardless when they are reused in the future. Therefore, whenever PDec uses these data, it has to fetch them from the off-chip DRAM, which will result a high energy consumption and a long access latency. This section shows how I design the memory hierarchy to utilize these localities for energy efficiency. It contains the multi-level storage for the spatial neighbors and a cyclic memory for the temporal neighbors with corresponding compression scheme.

### 2.3.1 Memory Hierarchy for Spatial Storage

In parameter decoder design, data accessing for spatial neighboring is considered as the bottleneck for achieving high throughput requirement of 8K UHDTV application. As is mentioned in Section 2.1, five neighboring blocks are needed for MV process and all left and top neighboring blocks for BS calculation. All the data should be fetched

within the scheduled cycles listed in Table 6. In order to meet the throughput, line buffer is maintained as shown in Fig. 38. Considering the ability to distinguish different blocks, the proposed line buffer's cell is set equal to minimum TU size (4x4). Meanwhile, to simplify the control logic, single line buffer is employed to consist of not only the storage of the top row (Top-Buffer) but also the left blocks of current CTU (Left-Buffer). Under this memory organization, the neighboring block A0, A1 are stored in Left-Buffer while B0, B1 are in Top-Buffer correspondingly. For each prediction unit, these four blocks can be read out from line buffer in four cycles. If CU is divided into two PUs, at most two blocks (A0, B1 or A1, B0) can be reused so extra two cycles is needed. Further, A1 and B1 can be reused for BS calculation to save extra two cycles.

In addition, fifteen registers are maintained to store the top-left B2 blocks. One is that the proposed replacing strategy for line buffer can't store the blocks at the concave corner in the decoded regions. So the top-left registers help implement the defection of line buffer. On the other hand, I use register instead of reusing line buffer for B2 storage. The reason is that B2 will be read and refreshed frequently inside CTU, implementation by register will eliminate the potential memory accessing conflict problem (read and write simultaneously). The Left-top register consists of fifteen identical registers, each of which is 95 bits. The content of these 95 bits is the same as that in line buffer, which is shown in Fig. 38. Whenever a block at concave corner is written into line buffer, it is also written into correspondent Left-top register simultaneously. Thus there is no communication between line buffer and Left-top register, so that further memory conflict problem is avoided.

The proposed pipeline stages in Section 2.2 leads to potential hazards for memory access. Forward by-passing is used to eliminate the problems. In detail, I allocate memory reading and writing in stage 1 and stage 3 respectively. Thus two potential memory conflict hazards exist. Firstly, the same memory address may be read and written in one cycle simultaneously. The second is write-after-read hazard caused by delayed writing operation. For these two hazards, two by-pass registers are inserted between stage 1 and 2 as shown in Fig. 31. The proposed pipeline fixes the delays between stages so that stage 3 is always 12 cycles delayed than stage 1. Meanwhile, the

minimum process cycles are 8 cycles when CU is 8x8. Therefore, when current CU is on its third stage, at most following two CUs go through its first stage. Potential memory hazards can only happen during these 10 cycles. Thus I use registers to store the previous two CUs' result. Whenever current CU wants to access contents of previous two CU, I directly fetch them from registers instead of reading from line buffer. Therefore, the forward by-pass scheme can efficiently deal with the potential hazards for the line buffer.

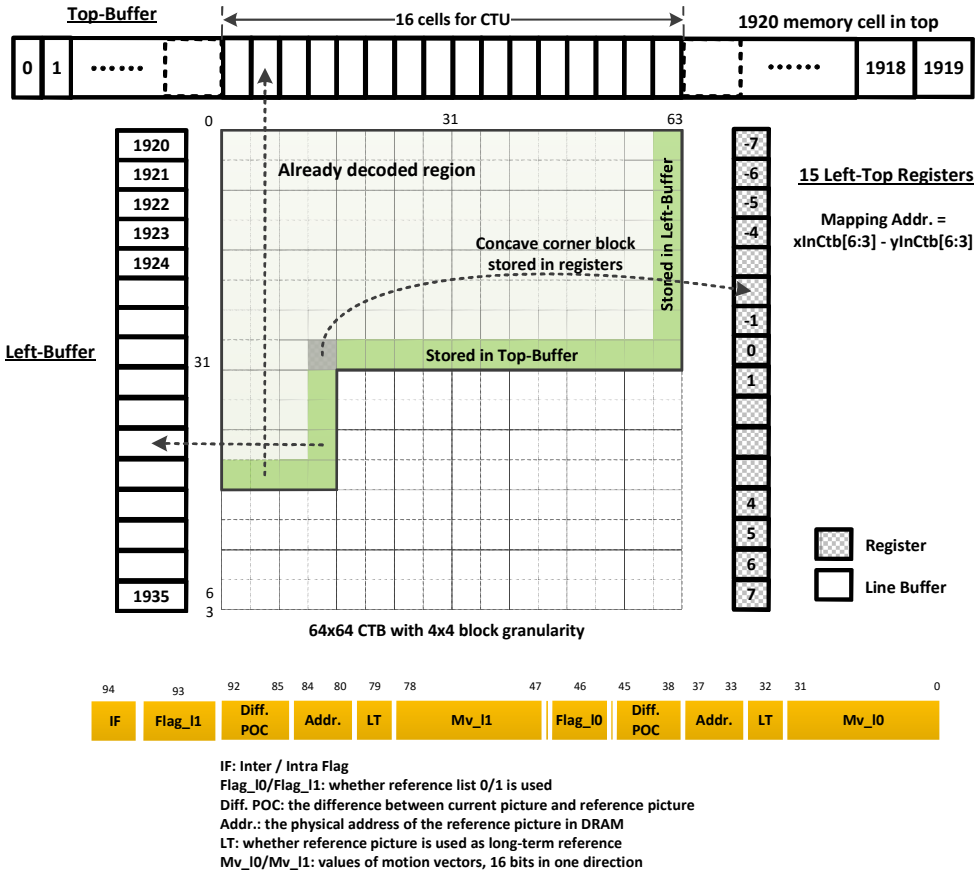


Fig. 38 Line buffer and left-top register organization

## 2.3.2 PU-based Coding Scheme

In this section PU-based coding scheme and the cyclically mapped SRAM are introduced in detail. All the col-located data is stored in the off-chip DRAM because of the huge data volume. Generally, DRAM bandwidth issue is regarded as the bottleneck for the real-time video decoder. In terms of 8K UHDTV, more attentions should be paid to relieve the DRAM bandwidth burden.

In the previous work [19], large blocks are divided into small constant blocks as mentioned in Section 2.1.4. When the data are written to DRAM, they have to execute some pre-processing to merge these small blocks into the original size. In my work, the CU-adaptive pipeline ideally keeps the block information so that no pre-processing is required. Therefore, the PU-based coding scheme is proposed.

The PU-based coding scheme is proposed for DRAM bandwidth reduction. Writing data back to DRAM for each 4x4 blocks, just like the way of line buffer, is unacceptable because of the huge data volume. I notice that only the top-left corner 4x4 block in each 16x16 block will be used as the temporal candidate in HEVC standard, which means I only need to store one set of prediction parameters for each 16x16 block for col-located storage. It will help reduce 93.75% of DRAM bandwidth. Further, considering the prediction unit larger than 16x16, the same co-located information is repeatedly written into DRAM for each 16x16, leading to meaningless burden on bandwidth requirement. Thus I propose the PU-based coding scheme for co-located data compression. The data stored in DRAM contains not only the prediction parameters but also the 8-bit description for current PU in CTU. As depicted in Fig. 39, a 32x32 PU will write identical data (95 bits) into DRAM for four times. PU-based scheme will avoid such kind of redundancy by writing this identical data (PU info.) only once with 8 trailing bits. Though extra 8 bits for PU description is added for each co-located block, the redundant DRAM storage is avoided. The experiment result shows that by applying the PU-based coding scheme, further 30% of bandwidth requirement reduction can be achieved.

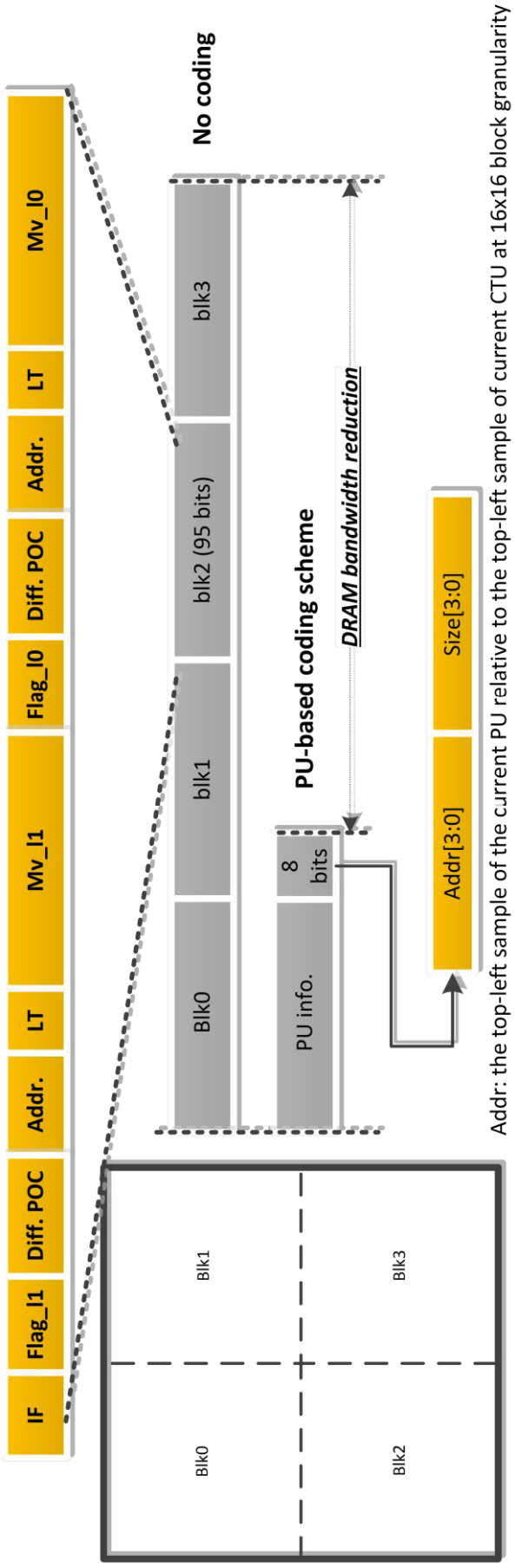


Fig. 39 PU-based coding scheme for DRAM bandwidth requirement reduction

### 2.3.3 Cyclic Memory for Temporal Storage

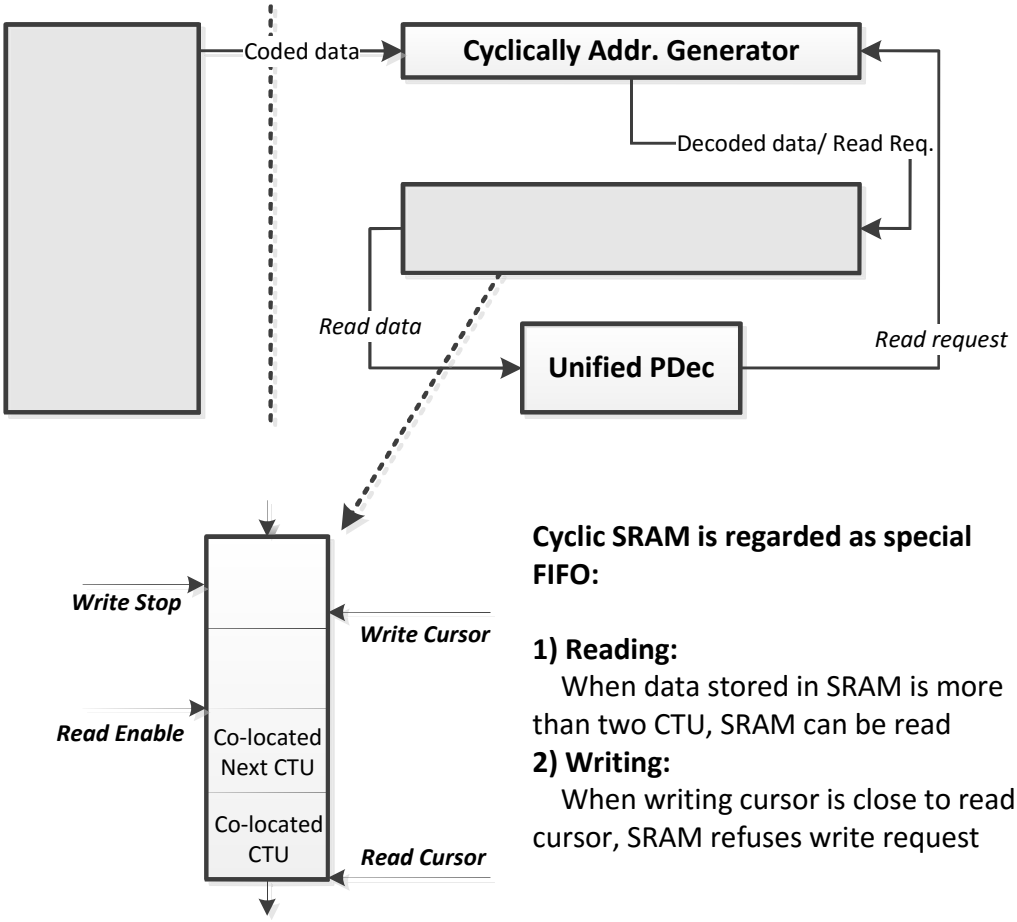


Fig. 40 Working mechanism of cyclically mapped SRAM

Fig. 40 illustrates the proposed cyclic SRAM. It is designed to support the random access characteristic for accessing co-located data. As is mentioned above, the co-located data is pre-coded before storage. In the reading side decoding process is first done. After that the decoded data will be written into the SRAM for random access. The proposed cyclic SRAM can be regarded as an addressable FIFO whose size is four CTU's co-located information. Based on the analysis on HEVC, the current decoding block will only use the co-located CTU and the col-located next CTU's data for MV calculation, as shown in Fig. 41. Thus, as long as col-located following two CTUs' content is stored inside, the cyclic SRAM can be read by PDec. In the non-read cycle, cyclic SRAM will keep writing decoded data until the cyclic SRAM meets the full condition, when writing cursor is equal to the write stop address. Single-port SRAM is

capable of above functions so as to reduce control logic and save area cost.

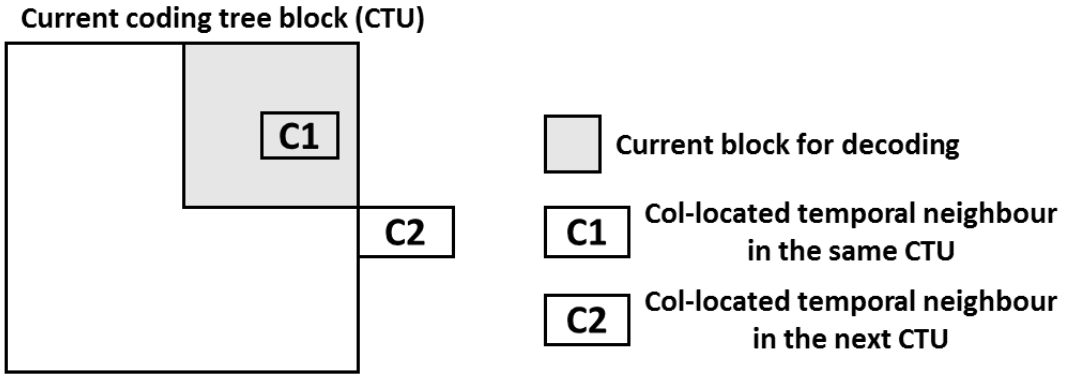


Fig. 41 Distributions of collocated temporal neighbors

For the implementation, I proposed a cyclically address generator as shown in Fig. 40 to control the data writing. In detail, the address generator has an input called read request. This input is used to monitor the process status inside the pipeline. The status represents the current processing block's coordinates. By adding a redefined address shift for two coding tree unit sizes, the result is regarded as the write stop address. If current write cursor is less than the write stop cursor, this generator will write the fetched data from off-chip DRAM into cyclical SRAM and increase its write cursor. The write operation will be stopped as long as the updated write cursor is larger than the write stop address.

A read enable signal is also implemented. This signal is used to indicate that the cyclical SRAM has prepared enough data for the pipeline's access. Generally, when the address difference between write cursor and read cursor is larger than two coding tree units, this signal is pulled high. If this signal is low, the pipeline has to be stalled to wait the generator to write enough data into the cyclical SRAM. These hardware implementations guarantee the correctness of temporal neighbor fetching in Fig. 41.

## 2.3.4 Design overhead of This Section

The design overhead for saving memory access energy is the more complex memory architectures and organizations compared to the previous works like [19]. In [19], the memory mainly consists of one global line buffer. All the data that need to

transfer between off-chip DRAM memory and the core architecture should be buffered by this global buffer. In comparison, the proposed memory hierarchy in this section contains separated memories as shown in Fig. 42 which increases the design costs.

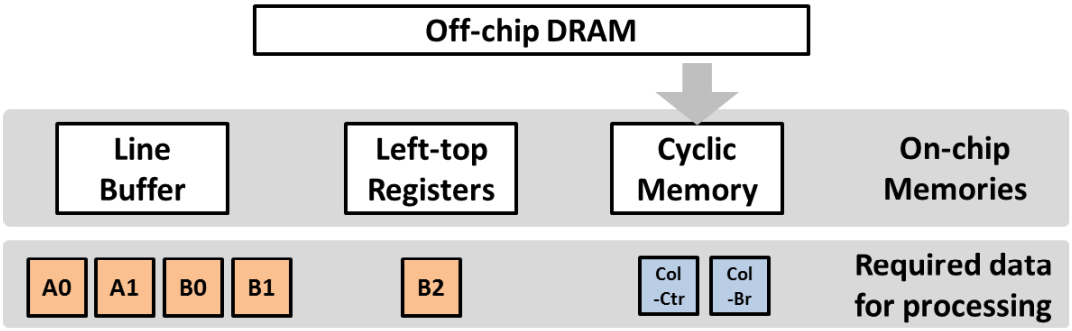


Fig. 42 On-chip available memory storages and their contents

Firstly, the more global buffer resources are required because this proposal tries to buffer the data of an entire row for spatial storage. It is unlike [19] which buffers the data mainly at the off-chip DRAM. Therefore, at least two times more memory resources are necessary for supporting 8K applications compared to the 4K applications in [19]. Meanwhile, because the increased complexity of HEVC, the required bits for each motion information are also increased, which will consume more memory resources for this proposed memory hierarchy.

Secondly, extra addressing hardware costs are necessary for deciding when and where to fetch reference data because they can be stored in either spatial storage including line buffer and left-top registers in Section 2.3.1 or cyclic memory introduced in Section 2.3.3. As shown in Fig. 42, processing of each CU may require five spatial neighbors and two temporal neighbors, which are distributed in different on-chip memories. Hardware should have the ability to decide when and where to fetch the correct data. This is achieved by using the internal counter which has been introduced in the first technical challenge of Section 2.3.4. This internal counter will decide the hardware behavior for each specific cycle like shown in Fig. 32.

In conclusion, the proposed memory design in this section will require more on-chip resources and complex addressing circuits for realization.

# 2.4 Experimental Results

## 2.4.1 Simulation on PU-based Coding Scheme

Table 7 DRAM bandwidth reduction of PU-based coding scheme

| Video Sequence | Configuration | QP | Optimized Bandwidth | BW Reduction |
|---|---|---|---|---|
| 06000 | RA | 25 | 28.4MB/s | 69.3% |
| | | 40 | 11.2MB/s | 87.9% |
| 12000 | RA | 25 | 23.3MB/s | 74.8% |
| | | 40 | 10.8MB/s | 88.3% |
| ClassA | LD | 25 | 66.6MB/s | 27.8% |
| | | 40 | 35.8MB/s | 61.2% |
| | R | | .0MB/s | 33.9% |
| | | 40 | 34.7MB/s | 62.4% |

PU-based coding scheme helps to reduce the DRAM bandwidth (BW). As this work is aiming at UHDTV application, I choose ClassA sequences (Traffic and PeopleOnStreet) which is 2560x1600 with 300 frames and two 7680x4320 video sequences (06000 and 12000) with 150 frames as my test sequences. Different configurations (RandomAccess (RA) and Lowdelay (LD)) and QP value are considered. The result is shown in Table 7.

In Table 7 all the bandwidth is normalized to 8K@60fps. If I use 16x16 block storage as is mentioned in Section 4.2, necessary DRAM BW is 92.3MB/s. PU-based coding scheme can further reduce the bandwidth by at least around 30%. For a 7680x4320 sequence 12000 at QP=40, up to 88.3% BW reduction can be achieved.

### 2.4.2 Synthesis Results

The unified PDec is implemented on register-transfer level design using Verilog HDL and further synthesized by Synopsys Design Compiler with TSMC 90nm process, which is the same with the latest previous work [19] for a fair comparison.

In detail, the maximum clock frequency of my proposal can achieve 400MHz. The area under 300HMz is 93.3k. The size on-chip line buffer is 23.0kB for buffering the last rows information. Here I analysis HEVC standard and define the worst is that the whole frame is coded as 8x8 coding unit. From Table 6 I know that the cycles for worst case are eight, equally to 8 pixels per cycle. This throughput is enough for real-time decoding for 7680x4320@60pfs video sequence. Compared to the previous work [19], we not only well pipelined the designs for a faster clock frequency, but also supports the newest HEVC standard. In HEVC, the algorithm for generating motion vector is much more complicated compared to that in H.264. I also simulate my proposals on HEVC test sequence and the result shows the average process speed is 17.8pixel/cycle, which is able to finish decoding highest profile 6.3 7680x4320@120fps applications at only around 111.8MHz on average.

Table **8** shows the comparison with other related works. Compared to existing works, mine is the only one that supports HEVC standard. Notice that though in [18][19] intra prediction mode calculation is included, it affects final area and timing cost little since algorithm for it is quite simple compared to that of MV, especially in HEVC. On the other hand, line buffer size is much larger than others for three reasons. 1) HEVC support PU's edge equals to 4 at least, so storage for 4x4 block granularity is needed; 2) MV parameters in HEVC is more than that in H.264; 3) 8K UHDTV's frame width is twice larger than 4K's, leads to double size of line buffer. Finally, as total area is related to the throughput, I define the normalized gate number in the table for a fair comparison. The normalized results show that my proposed unified architecture has around 36% efficiency on area cost, even if supported HEVC's complexity is more than that of H.264/AVC.

Totally, my proposed unified motion vector and boundary strength parameter decoder can support real-time decoding for 7680x4320@60fps video under 249MHz in the worst case. The worst case is that all the shapes and sizes of prediction units in the coded bit stream are 8×4 or 4×8, although this situation has few possibilities to be appeared in the real cases.

Table 8 Comparison with state-of-the-a

| | | | | |
|---|---|---|---|---|
| Standard | H.264/AVC | H.264/AVC&AVS | H.264/AVC | H.265/HEVC |
| Function | MV | MV/BS/Intra | MV/BS/Intra | MV/BS |
| Worst-case Throughput (Pixel/cycle) | 0.98 | 0.73 | 4.0 | 8.0 |
| Avg. Throughput (Pixel/cycle) | 1.6 | 1.6 | 4.0 | 17.8 |
| SRAM Size (Line buffer) | 2.8k(1080p) | 4.75k(1080p) | 3.6k(2160p) | 23.0k(4320p) |
| Logic Gate | 52k | 63.0k | 37.2k | 93.3k |
| Norm. Logic Gate | 41.80 | 101.27 | 7.47 | 4.69 |
| Max. Resolution | 1920x1080 @60fps 126MHz | 1920x1080 @30fps 84MHz | 3840x2160 @60fps 124MHz | 7680x4320 @60fps 249MHz |
| Technology | 90nm | 65nm | 90nm | 90nm |

## 2.5 Summary

This chapter presents a unified parameter decoder VLSI a

contributions are concluded in this chapter:

1) The unified architecture for MV and BS parameter decoder is proposed to achieve memory sharing. Due to this scheme, only one line-buffer is employed inside the whole decoder system for 50% saving of memory resources.

2) The CU-adaptive pipeline is proposed to support high throughput of up to 8K application with reasonable implementation complexity. It guarantees the high efficiency for processing the diverse block types in HEVC and enhances the data sharing inside each block type.

3) The proposed Index-mapping and resource reuse schemes are introduced for irregular process algorithm to achieve 43.2k logic gates reduction. Instead of generating all the intermediate results, a mapping relationship between inputs and final outputs is first built up so that the complicated circuits for intermediate results can be omitted.

4) Memory organization is well designed and PU-based coding scheme for co-located storage can reduce 30% of the total off-chip DRAM bandwidth requirement. Due to the proposed CU-adaptive pipeline, the pre-processing can be omitted before the compression.

5) Memory architecture including top line buffer, left line buffer, top-left RegisterFile and bypass designs can support energy-aware and sufficient burst memory bandwidth requirement.

Temporal data reuse is utilized in these contributions for pursuing an efficient design. For example, due to the preliminary knowledge of the data reuse between motion vector and boundary strength decoding, a unified architecture is proposed. Similarly, the CU-adaptive pipeline schemes can reduce the design complexity and enhance the data reuses because I can predict the potential data reuse inside adjacent blocks. For memory design, I maintain a top line buffer to store the motion data which will be reused by the next row of 64×64 coding tree units. Instead of storing them in to registers, SRAM memory is used to organize it. The reason is that I know the reuse of these motion data and I have to store all the data of a frame width. These will consume a huge amount of registers with significant energy costs.

As a result, energy efficiency is achieved by my block merging based approach.

For memory energy, around 30% to 90% of the temporal neighbor access can be achieved by the PU-based coding scheme. I also mentioned that CU-adaptive pipeline can further enhance the data reuse for unified decoding of motion data and boundary strength. In terms of the computing energy, optimization like index-mapping scheme remove the redundant calculations for intermediate results to save up to 43.2k logic gates, which is 46% of the final logic gate count. The reduced logic gates mean saving for required computing energy, especially for the leakage power.

On the other hand, the design overheads are also increased in order to achieve the above performance. Firstly, the pipeline design is more complex compared to [19] as the hardware should handle four different CU types. Moreover, the required process times vary according to the CU types. This is solved by a pre-defined Look-u

In total, these issues increase the design overheads by requiring more logic gates up to 93.3k as shown in Table 8. Considering the high throughput requirement and the supported new HEVC standard, the increasing overheads are acceptable.

In conclusion, the proposed unified parameter decoder supports real-time HEVC video decoding for 7680x4320@60fps application at 249MHz even in the worst case.

# 3. Distance-biased Cache based HEVC Motion Compensation Architecture

In this chapter, an energy efficient HEVC motion compensation architecture (MC) with proposed distance biased direct mapping cache is proposed for real-time decoding 8K UHD applications. MC in HEVC decoder system contains two part, prediction part and cache system. As the most area and power consuming modules inside the many decode designs like [21], it is essential to put sufficient attentions on finding energy efficient VLSI design solutions.

Computing energy issue mainly consists of two aspects in the design of motion compensation. Firstly, the previous designs involve many redundant computations in the pipeline designs. This redundancy will significantly influence the performance for 8K applications. Secondly, the higher throughput requirement also involves higher design complexity of cache controller. Especially, cache conflict checks in Section 3.4.3 involves more computation resources in the previous designs. An efficient architecture is proposed in this chapter for the computing energy.

Memory energy in motion compensation involves the design of cache system. Existing cache designs cannot fully exploit the features of videos. The cache performance is enhanced at the expense of higher logic gate costs, or vice versa. A better solution for reducing memory energy with reasonable design complexity is also discussed in this chapter.

Both temporal and spatial locality is involved in this design due to the random-access features in motion compensation. To utilize these localities, cache based architectures will be deeply discussed in this chapter. Meanwhile, the inefficient pipeline designs in previous works destroy the data reuse potentials between pipeline blocks so that the performance is dropped. This chapter will show a novel cache based motion compensation architecture that can enhance the memory energy efficiency with acceptable design complexity which is related to the computing energy efficiency.

# 3.1 Introduction

High Definition Television (UHDTV) has been attracting more and more attentions in recent years. It can support 3840×2160 and 7680×4320 high resolution formats, which are 4 to 32 times larger compared to high definition television (HDTV). Besides that, higher frame rate at 120 frames/s and up to 10-12 bits depth per sample are specified by UHDTV to further enhance the visual experience. Behind the advantages, the challenge for supporting real-time UHDTV application is the critical throughput requirement. Under this circumstance, new High Efficiency Video Coding (H.265/HEVC) has been standardized by JCT-VC in 2013 [12]. It announced to double the compression ratio when compared to previous Advanced Video Coding (H.264/AVC) standard [6].

Inherited from past generations of mainstream video coding standards, motion compensation undertakes the indispensable role on predicting inter coded samples. Compared to H.264/AVC, inter prediction in H.265/HEVC achieves a better coding efficiency at the cost of higher complexity. Hierarchical coding units ranging from 4×4 to 64×64, not only produce much more input patterns than H.264/AVC, but also extend the largest block from 16×16 in H.264/AVC to 64×64, leading to a notable increase of buffer memory. Meanwhile, a new 8-tap interpolation filter increases the computational complexity as well as imposing higher bandwidth requirement on off-chip DRAM memory. All these mentioned above significantly challenge efficient hardware implementations. If I further consider the high throughput of UHDTV and DRAM bandwidth requirement, a novel MC architecture is desired for the upcoming UHDTV and HEVC era.

## 3.1.1 Function Description in Decoder

Motion compensation is a block-based function unit in HEVC/H.265 as shown in Fig. 43. In this paper, I define the MC architecture whose inputs are related motion information of a prediction unit (PU), including its description (location and size),

motion vectors (MV) and reference frame. Outputs are the motion-compensated prediction for a block of pixels before reconstruction. Note that MV decoding is pre-executed in an independent parameter decoder module prior to MC as shown in Fig. 43, due to the complicated algorithm of MC in HEVC and considerations of the data sharing in the level of the whole decoder. The detail can be found in Chapter 1.

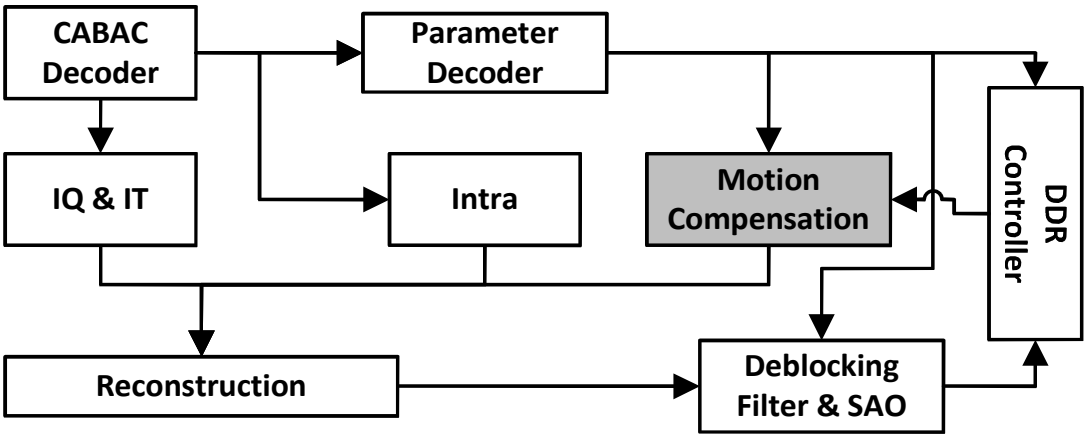


Fig. 43 MC in decoder connected with PDec and reconstruction module .

I conclude the function of MC into three steps. The first step is fetching reference data of each inter-predicted PU from DRAM. The reference data refers to a block of similar samples in previous decoded picture. The decoded pictures are stored in the off-chip DRAM, so the reference data should be fetched first. The second step is the interpolation. As the predictor is usually aligned to a fractional pixel location, MC will call the 8-tap interpolation filter to generate the values of fractional samples. The last step is weighted prediction (WP) for bi-predicted PU, which contains two available modes. The explicit mode signals coefficients of WP directly through the bit stream. The other mode is simply averaging two motion compensated predictions.

## 3.1.2 Data Reuse of Reference for Design Challenges

### *3.1.2.1 Throughput: Compute Energy*

The most notable difficulty for 8K UHDTV design is the throughput issue. Considering 7680×4320@60fps specification, if 350MHz working frequency is

assumed, at least 5.69 pixels/cycle should be achieved for real-time decoding. Moreover, high precision interpolation filter kernels are supported by extending filter taps to 8 for luma and 4 for chroma. This directly enlarges the calculation region for interpolation as shown in Fig. 44. Taking one luma sample as an example, up to 8×8 reference samples should be motion compensated to produce the fractional samples' values. The example of calculating two vertical adjacent fractional pixels is given in Fig 50. Each fractional pixel requires 8×8 reference pixels and data sharing can be found because of overlapped internal results. A large amount of filtering calculations is executed during the interpolation. Therefore, a large amount of logic gates is required to achieve the high throughput requirement for real-time decoding 7680×4320@60fps video sequences.

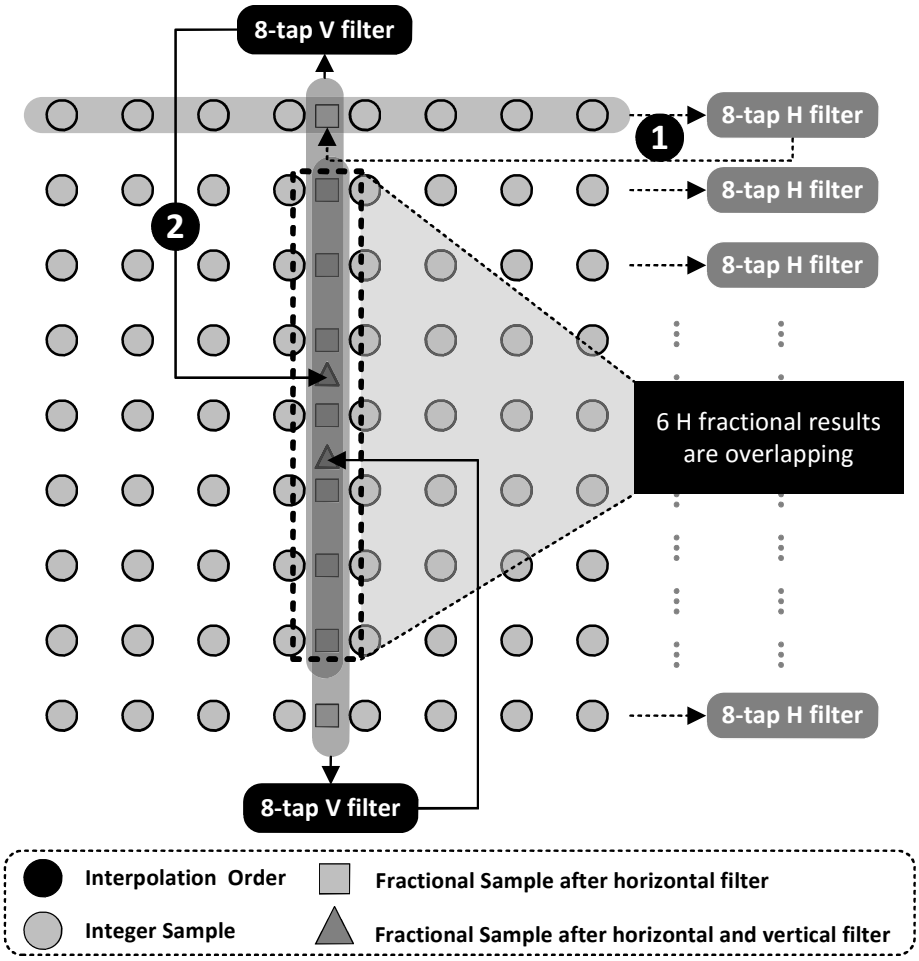


Fig. 44 8-tap interpolation filter kernel in HEVC/H.265

Parallel design of interpolation is applied in many previous approaches. For

example, [21]'s approach increases the number of horizontal filters with variable pattern, to achieve a throughput of 4 samples/cycle. The parallel implementation of interpolation itself involves few new challenges. However, other MC modules, such as MC cache, should be compatible with the throughput of interpolation. A higher throughput usually requires a much more complicated implementation of cache. My design aims to construct a reasonable interpolation so that it can take both performance and compatibility into account.

#### *3.1.2.2 DRAM Bandwidth: Memory Access Energy*

Another indispensable issue is the DRAM bandwidth requirement, which is even regarded as the bottleneck of a whole video decoder. According to Fig. 43, there are three modules in the decoder sharing the external DRAM, which are parameter decoder, motion compensation and filters. In -prediction is designed to have no communication with the external memory. Both the parameter decoder and filters access the external memory in a pre-known order. It allows us to pre-buffer the read and write accesses of these data in queues, and perform the real memory access only when the memory interface is not being used by MC. However, the reduction of bandwidth requirement of MC is quite challenging because of the random access for reference pixels. In this work, I mainly focus on the MC-related DRAM issues because of its crucial impact on the throughput performance.

This issue comes from two aspects. The major reason lies in the massive data volume of UHDTV. Along with the increasing resolution and frame rate, the amount of reference pixels proportionally increases. The other reason is the longer interpolation taps. Its effect is obvious when PU size is small. An 8×8 PU has up to 15×15 reference region, leading to 2.5 times larger data volume compared to original PU size.

The importance of bandwidth issue attracts many researchers' attentions. MC cache is an efficient contribution to reduce DRAM bandwidth requirement. However, the cache design should handle the problems that multiple reference frames exist in HEVC.

To show this problem caused by multiple reference frames, the reuse of reference is first given as shown in Fig. 45 as the overlapped data during the process. The example

in Fig. 45 contains three blocks processed sequentially and block B has a different reference frame. If the MC cache consists of only one cache set, the reference data must be buffered by this cache set and reference of B will flush the data of reference of A. When Block C is processed the available reuse of reference between A and C inside the cache set is reduced, resulting in repeated accesses to the off-chip DRAM. Therefore, this problem implies that an efficient cache design is required for keeping this reuse of reference so that a better hit-rate can be expected.

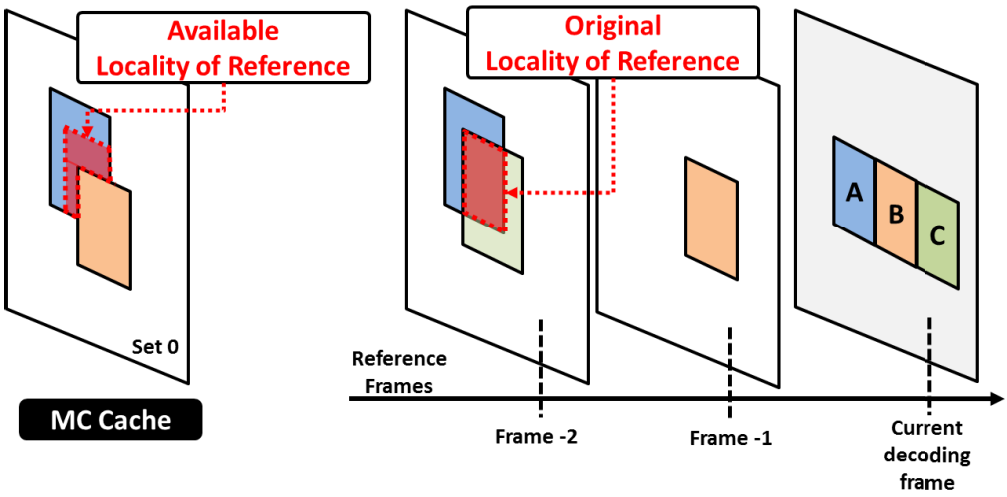


Fig. 45 Reuse of reference in the MC cache design

To solve this problem, a basic idea is to increase the number of cache sets as shown in Fig. 46. By doing this, the reference of Block B can be separately written into different cache sets to avoid the data flushing mentioned in Fig. 45. This idea is used by many previous works. For example, multi-way set associative cache has been widely adopted to tolerate the potential tag conflict for multiple reference frames and distinguishing luma and chroma component in comparison to the direct mapping. Though it achieves acceptable performance on bandwidth issue, the complicated control logic and mapping method causes MC cache to cost a notable amount of logic gates. Moreover, up to 64×64 PU size is supported by HEVC/H.265 rather than 16×16 in H.264/AVC. Correspondingly, cache size must be enlarged so that equivalent performance can be maintained. This results that the size of MC cache and its control logic become new issues. In conclusion, I expect a more efficient MC cache architecture

to deal with the intensive DRAM bandwidth issue.

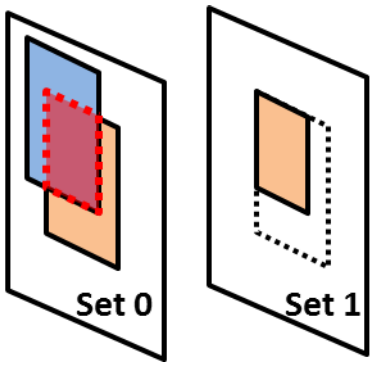


Fig. 46 An approach to increase cache sets for the problem in Fig. 45

### 3.1.3 Literature Review

Pixel prediction in video decoder, called motion compensation, has several improvements compared to H.264 as mentioned in Section 1.1.3. There are many works for H.264 including two works from my research lab [22][26]. These works mainly focus on H.264 algorithms and pursue high throughput with an area efficiency. However, insufficient discussions are given for energy efficiency except approaches on off-chip DRAM bandwidth reduction. Therefore, high energy efficient architectures are expected.

Two HEVC based works, P. Chiang [43] and M. Tikekar [21], are discussed.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 4 | 5 | 16 | 17 | 20 | 21 |
| 2 | 3 | 6 | 7 | 18 | 19 | 22 | 23 |
| 8 | 9 | 12 | 13 | 24 | 25 | 28 | 29 |
| 10 | 11 | 14 | 15 | 26 | 27 | 30 | 31 |
| 32 | 33 | 36 | 37 | 48 | 49 | 52 | 53 |
| 34 | 35 | 38 | 39 | 50 | 51 | 54 | 55 |
| 40 | 41 | 44 | 45 | 56 | 57 | 60 | 61 |
| 42 | 43 | 46 | 47 | 58 | 59 | 62 | 63 |

Fig. 47 Direct mapping with 32×32 block mapped in a coding tree unit [43]

P

-chip memory accesses.

Moreover, the work focuses on the simple direct mapping scheme for reducing the design costs. The cache size is also very small (32×32) in Fig. 47, which would cause many cache misses. This may not be an issue for 4K@30fps applications but will dramatically drop the performance when working for my 8K@60fps specification. (4x4 redundant memory and compute, direct mapping with small size)

M. Tikekar [21] proposed another architecture for motion compensation with the same 4K@30fps specification. For interpolation, it defines the basic pipeline blocks as 16×16. Compared to the scheme of 4×4 in [43], This can remove some redundant computations and memory accesses while still not so efficient for larger block sizes. For the cache design, this work chose a cache associativity of four to handle the random accesses for multiple reference frames. Although the cache performance is good, the memory size, number of control registers and address mapping are relatively high which has been mentioned in [43]. For example, the extra circuits for cache replacements should be designed for these multi-way associative cache camping. Moreover, four set of addressing circuit for each set are also introduced. The logic gate count for cache is as many as 90.4k. Therefore, this problem is expected to be enlarged for higher throughput requirement like 8K@60fps applications.

In conclusion, although both works are designed for HEVC, there still remain some design challenges for my research targets. Firstly, when the throughput requirement is 8K@40fps, the current design schemes are not efficient. For example, they both choose a small block sizes (4×4 or 16×16) as the pipeline block. Although the design complexity is reduced, the efficiency is scarified. These efficiency is valuable for the

high-throughput applications like 8K@60fps. Therefore, a more efficient pipeline designs should be considered. Secondly, cache design still has space for improvement. In [43], resource costs are the main concern so they choose a simple direct mapping scheme with a relatively low cache performance. On the other hand, M. Tikekar [25] pursues a high cache performance by employing the multi-way associative cache mapping. The involved design costs for logic gates also dramatically increased. An efficient solution that can balance between cache performance and design cost is urgently required as this part highly relates to the energy consumption of motion compensation.

Besides, many other researchers have made their contributions to the architecture design for MC. Chang et al. in [23] presented an optimal data mapping scheme to reduce the required bandwidth. Chuang et al. in [24] proposed a bandwidth-efficient cache-based MC architecture by exploiting intra-MB and inter-MB data reuse. Chen et al. in [25] discussed a unified MC design supporting multiple video codec standards. Zhou et al. in [26] extended Chen's work by giving approach on efficient detecting circuits for conflict hazard when accessing MC cache. While above approaches and many others like [27][28][29][30][31][32][33][34][35][36][37] are designed for H.264/AVC, Guo et al. in [38] first proposed an optimized MC interpolation filter for HEVC/H.265 that saves area cost. Other solutions of interpolation can be found in [39][40][41]. Sanghvi et al. in [42] proposed an efficient cache architecture to relieve pressures on DRAM bandwidth demand. Meanwhile, Tikekar et al. in [21], Chiang et al. in [43] and Cho et al. in [44] all gave integrated video decoder designs for HEVC/H.265 that incorporate well-designed MC architecture to support 3840×2160@30fps throughput.

## 3.2 System-level Architecture

In Fig. 48 the top-level block diagram of my proposed MC architecture is illustrated. The prediction part, the cache-related part and the off-chip DRAM memory are depicted from left to right in turn. This work proposes several novel schemes on the MC cache, which will be introduced in the following sections. MC core (containing

interpolator and weighted prediction), MC cache and DRAM are three key modules inside MC architecture. Before discussing these schemes, in this section I will briefly state my system-level architecture design including the external memory choice.

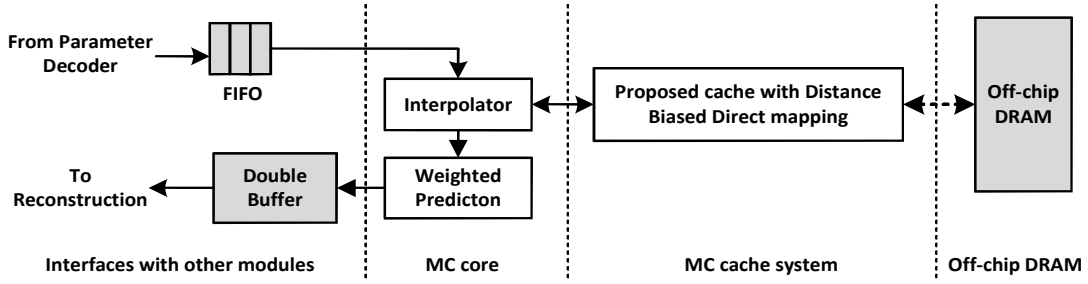


Fig. 48 The top-level block diagram of the proposed MC architecture.

## 3.2.1 Data Organization in External DRAM Memory

All of DRAM requests are processed by a DRAM controller. This controller has the ability to arbitrate when and which requests are sent to DRAM. I set a high priority for the requests of MC compared to the rest two modules sharing this DRAM because of the random accessing attribution and the real-time throughput requirement mentioned in Section 3.1.2.

A 64-bit DDR3 DRAM memory forms my DRAM system in Fig. 49. DRAM address consists of 13-bit row, 3-bit bank and 10-bit column. As the burst length of DDR3 is eight, the minimum access unit (MAU) between DRAM and MC core is 512-bit which maps an 8×4 block of 10-bit pixels containing luma and chroma components (Actually 480 bits are mapped to this 512-bit space with only 6.7% overhead). MAU exactly maps to a MC cache line, which will be further discussed in Section V. Reference pixel data are stored in DRAM in an interleaved manner. The lower bits of the DRAM address consist of bank addresses followed by the column addresses, while row addresses are assigned to organize the higher bits. This is to avoid frequent pre-charge/activate operations for row switching.

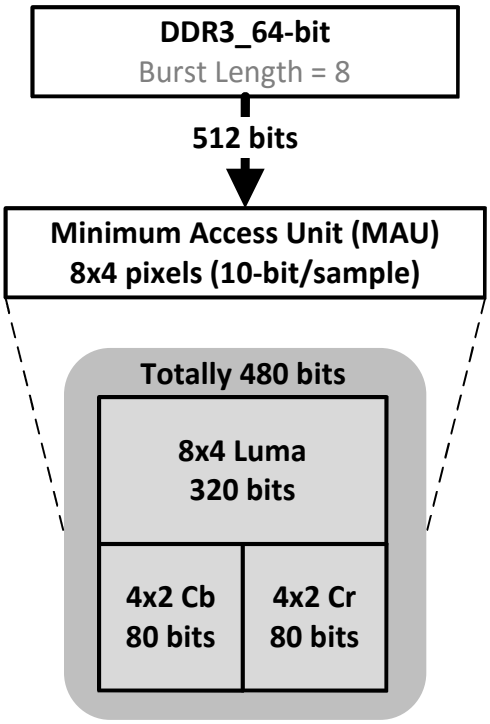


Fig. 49 DRAM organization and interface with MC decoder.

This unified storage for luma and chroma samples brings several advantages. Firstly, in HEVC/H.265 the MV of chroma component derives from that of corresponding luma part. Both luma and chroma components share the same MV information when accessing DRAM. Compared to a separated storage method, this unified storage helps to avoid frequent DRAM row pre-charge/activation operations when switching between luma and chroma components. Secondly, as the prediction of luma and chroma has no dependency on each other, it is possible to process them in parallel. Unified storage provides a suitable input pattern for the parallel interpolation design, which is utilized in my design and will be discussed in the next subsection. Due to the parallel design, the throughput of interpolation can be improved to 1.5 times compared to serial design of interpolation like [21][43]. The following discussion on the interpolation and cache will focus on the luma component, while readers should have a sense that the chroma part is designed by similar architecture of luma part. The word sample infers either luma or chroma component while pixel means both luma and chroma samples.

### 3.2.2 Width-fixed Strip Process Pattern

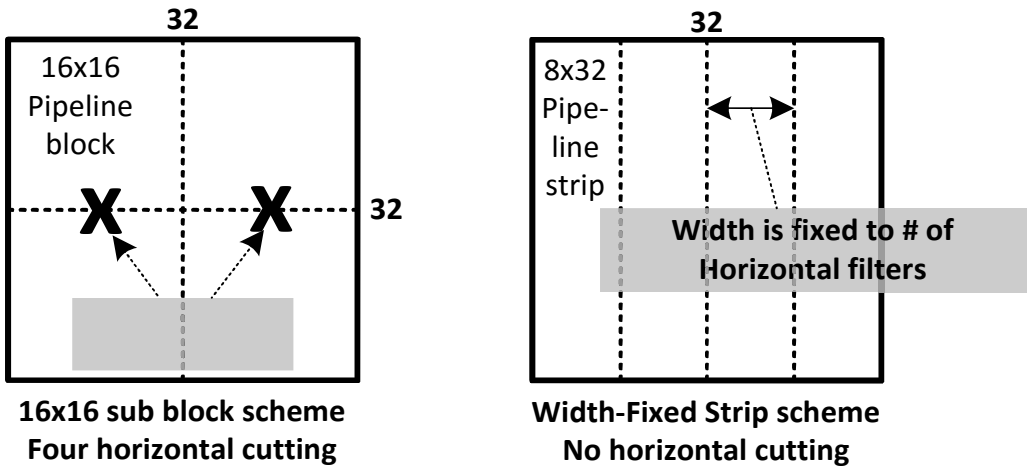


Fig. 50 The division method for Width-fixed strips process pattern of a PU.

A pipeline system is utilized to hide the long latency when accessing the off-chip DRAM memory. The pipeline granularity is defined as width-fixed strip (which I refer as to strip in the following). Each PU will be split into several strips with a fixed width. In this dissertation I unify this width as eight pixels. A strip may have a width of four in cases where the width of PU is not a multiple of eight. Specifically, this happens when the prediction unit's size is 4×8, 4×16 and the last strip in a 12×16 block.

In previous works [21][43], the pipeline block size is defined as 16×16. It is based on the observation that I can interpolate a large PU in smaller blocks by treating each of them as an independent PU whose motion information is identical to that of original PU. However, the throughput of pipeline is not efficient for PU larger than 16×16. Taking a 64×64 PU as an example, I actually need 71 × 64 = 4544 horizontal interpolation operations originally. This is assumed that I have enough calculation resources so that all the pixels in a 64×64 prediction unit can be processed in parallel. Meanwhile, I assume that the two-dimensional interpolation is done by first doing the horizontal filer followed by the vertical filter. The process order is like the example shown in Fig. 44. Therefore, the horizontal filters can be shared by adjacent pixels in vertical direction. Therefore, a column of 64 pixels will consume totally 71 horizontal

interpolation operations considering the data sharing. Therefore, 4544 horizontal filters are required.

However, 16×16 pipeline block is not efficient, especially for the blocks larger than 16×16. Inside a 16×16 block, I require 23×16 horizontal interpolation operations even if I assume that all the operations are done together. Therefore, 16 × (23 × 16) = 5888 operations are needed by continuously processing 16×16 blocks for a 64×64 block, leading to around 30% performance decrease. If there are insufficient computation resources to process all 16×16 pixels in parallel, the situation may become worse with more redundant computations.

The strip-based process pattern is able to avoid this performance drop because no horizontal cutting is introduced into a PU like Fig. 50. No horizontal cutting means that the horizontal interpolation results can always be reused followed by the 8×4 pipeline block which will be introduced in Fig. 52. Thus, a higher throughput performance can be expected.

### 3.2.3 Reuse-aware Design of Parallel Interpolator

The implementation of MC core starts with the design of interpolation filter kernel, which is the most computation intensive processing unit. The 8-tap filter kernel in HEVC is implemented by the adder-tree structure so that area-consuming multiplication is avoided to pursue a small area cost. There are three types of filter coefficients for 1/4, 2/4 and 3/4 fractional locations in [6], which I call Type A, B and C in Fig. 51 respectively. Three kinds of kernel coefficients supported by HEVC share common parts, which are first extracted and implemented so that it can be reused. The differential parts between the common part and each filter kernels are then implemented respectively. The total structure is shown in Fig. 51. Note that the horizontal and vertical interpolation filter kernels have the same architecture except the bit width for each function units. The inputs of horizontal filter kernel are original pixels with 10-bit data width, while the inputs of vertical filter kernel are the interpolated results from horizontal filters so that wider data width is expected.

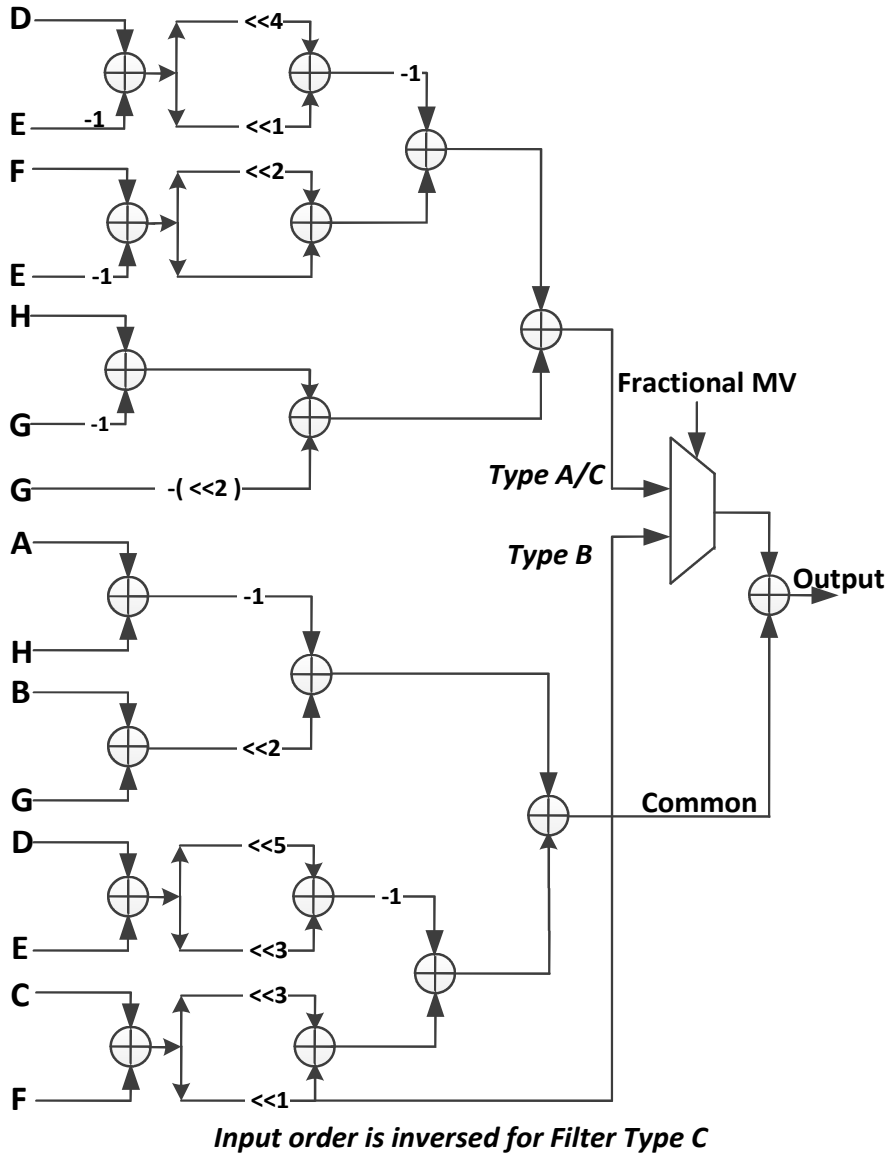


Fig. 51 The adder-tree structure for implementing an interpolation filter kernel.

With the interpolation filter kernel, I can implement an interpolator unit whose throughput is 1 pixel/cycle. It is implemented by serially connecting a horizontal filter kernel, eight registers and a vertical filter kernel. The algorithm has been discussed in Section 3.1.1. Totally 8×8 pixels are required to calculate one interpolated pixel. By continuously consuming a row of eight pixels per cycle, horizontal filter kernel will generate a horizontal interpolated sample value and push it into the register chain. Not until eight registers are filled with interpolated results can I u

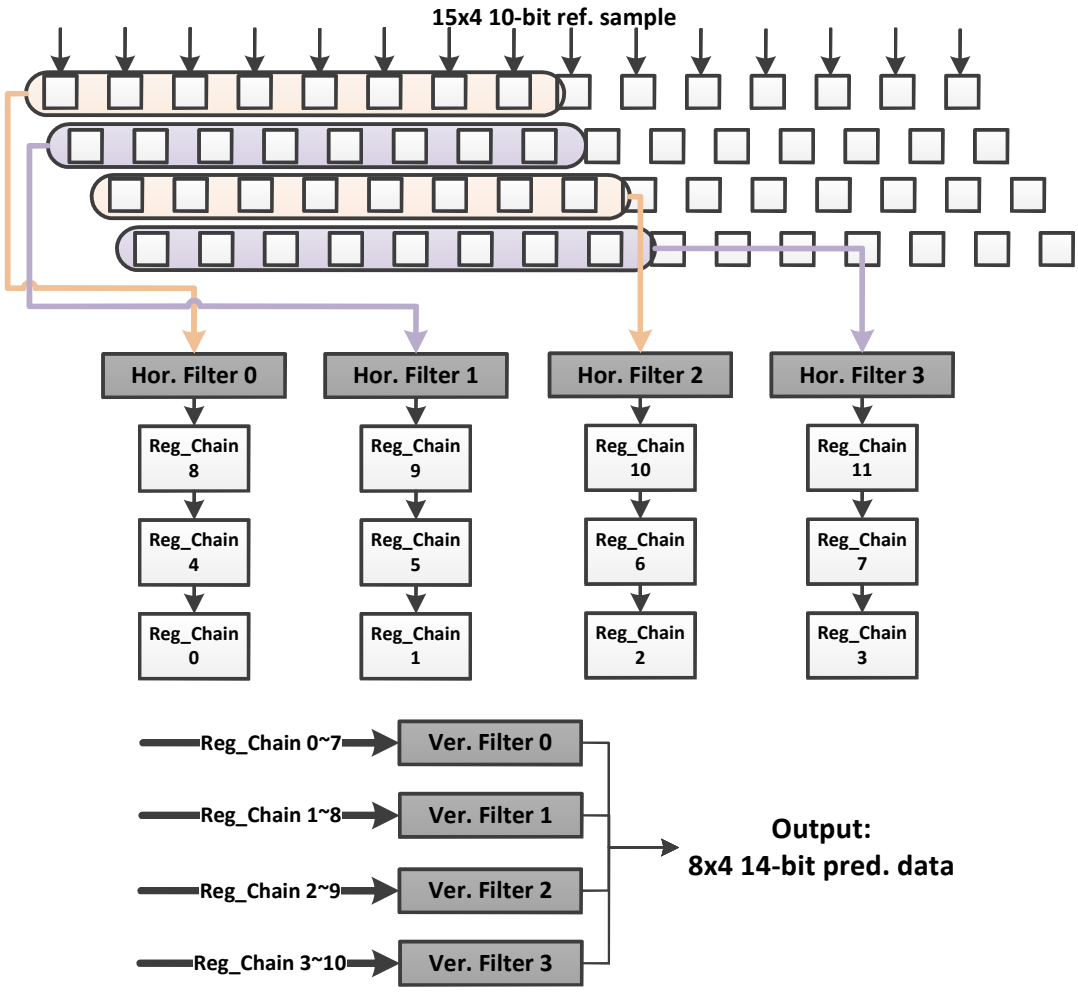


Fig. 52 8×4 parallel prediction pattern for interpolation design .

Inside the strips, I further improve the parallelism to enhance the throughput by stacking interpolator units in two directions. A prediction pattern of 8x4 pixels/cycle are de-signed in Fig. 52 Because of the overlap, only 15×4 input reference pixels and 12 internal registers are required for 8×4 parallelism. The interpolation is capable of outputting 8×4 pixels per cycle when the internal register chains are loaded with valid data. I a

I first divide it into eight strips. For each 8×64 strip, the reference region is 15×71, which can be read out from cache in 19 cycles. Considering the bi-prediction case, totally 2×8×19 = 304 cycles are required for interpolating this 64×64 PU, equivalently to a throughput of 13.5 samples/cycle.

Table 9 Throughput analysis of the strip-based pipeline for each PU type

| PU Type | Partition | # of Cycles (Bi) | Pixel/Cycle |
|---|---|---|---|
| 64×64 | All | 2×8×19 | 13.5 |
| 32×32 | All | 2×4×11 | 11.6 |
| 16×16 | 2N×2N, N×2N | 2×2×7 | 9.1 |
| | 2N×N | 2×2×2×5 | 6.4 |
| | nL×2N, nR×2N | 2×7×3 | 6.1 |
| | 2N×nU, 2N×nD | 2 × (2×5+2×6) | 5.8 |
| 8×8 | 2N×2N | 2×5 | 6.4 |
| | 2N×N | 2×4(Uni) | 8.0 |
| | N×2N | 2×5(Uni) | 6.4 |

## 3.2.4 Weighted Prediction

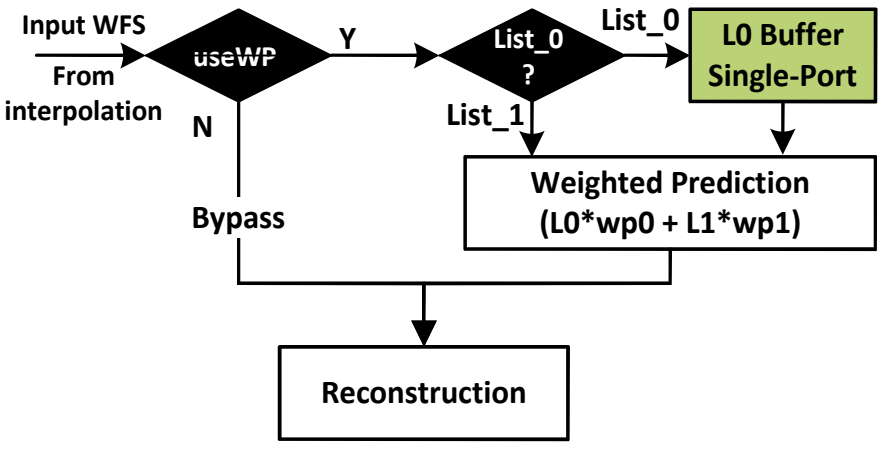


Fig. 53 The brief block diagram for the weighted prediction.

Weighted prediction (WP) is an optional coding tool in HEVC but it must be supported by a generic decoder. It is usually utilized to efficiently coding scenes with

lighting changes like fading scenes by multiplying a weight coefficient for compensation.

In bi-directional cases, two weighted coefficients, wp0 and wp1, are used to generated the final inter-predicted pixel values by doing the weighted sum. The implementation is shown in Fig. 53. The input of WP is the strips, which is defined as the pipeline granularity in Section 3.2.2. The output of WP is the reconstruction module as shown in Fig. 43. A double-buffer with two CTU-size are inserted to connect these two modules. In detail, I will receive an 8×4 block of interpolated pixels from interpolator every clock cycle. When WP is switched off, all the input data will be bypassed and sent to the reconstruction module. When WP is switched on, there will be two possible cases. If the input data are predicted from reference list 0, it will be temporally buffered by a single-port SRAM inside WP. If the input data are predicted from reference list 1, corresponding data stored in the SRAM will be read out and the weighted sum of the two interpolated results of reference list 0 and 1 will be calculated. The single-port SRAM whose size is equal to the largest PU is employed.

### 3.2.5 Interfaces and Data Flow

The input of motion compensation is the motion parameters of a PU from parameter decoder like motion vector and the reference index. An FIFO is utilized to buffer the outputs from parameter decoder. The output of MC is the inter-predicted pixels of a PU and are buffered by a double buffer memory. It is connected with the reconstruction module, which also has connections with intra prediction module and transformation module as shown in Fig. 43.

In detail, MC starts working with the PU information received from parameter decoder. This PU will be divided into several width-fixed strips according to the proposed strip-based pipeline in Section 3.2.2. Then, for each strip I will check the cache inside the motion compensation to find the missed data and fetch them from off-chip DRAM. This cache can exploit the data reuse across strips to further reduce the required DRAM memory bandwidth. The fetched data will be written into the cache

sequentially under the supervision of conflict checking, which detects the potential hazard of simultaneously reading and writing the same address. When the reference data of a strip are ready, they will be read out from the cache and sent to the interpolation. The throughput of interpolation in my proposed architecture is able to consume this input pattern to produce a block of 8×4 predicted pixels. After weighted prediction, the output of MC can be sent to the reconstruction module with a throughput of 8×4 block pixels per cycle. Finally, in order to handle all cases that may be faced in HEVC standard, I maintain a two CTU-size buffer between the output of MC and the reconstruction module to ensure pipeline utilization.

## 3.3 Distance Biased Direct-mapped Cache Design

Facing the intensive DRAM bandwidth requirement, many previous approaches utilize two-dimensional (2D) cache to alleviate this issue. They can be classified into two groups, direct-mapped cache and multi-way set associative cache.

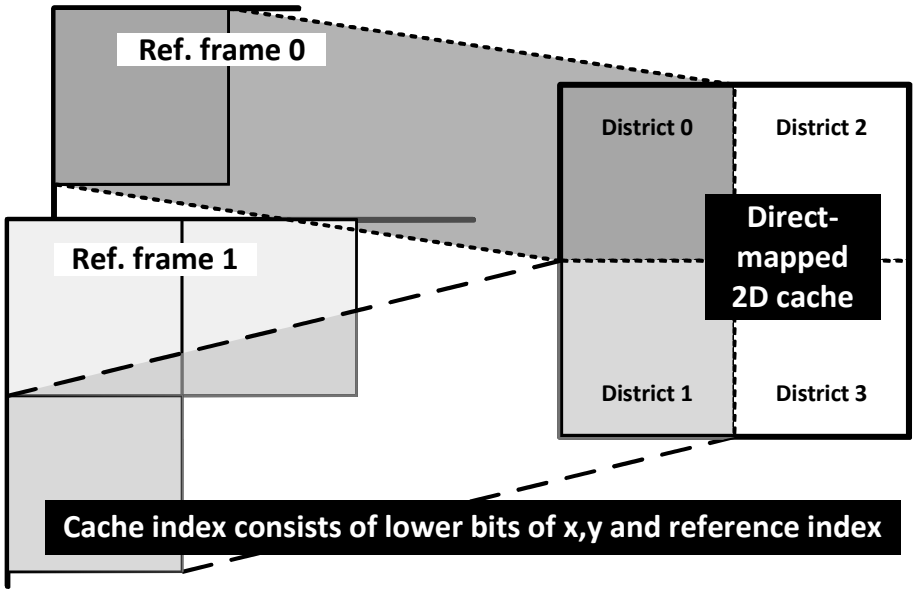


Fig. 54 An example of direct-mapped 2D cache with reference index.

A direct-mapped cache organization has its advantages on area-efficient characteristics. A general implementation for a direct-mapped 2D cache maps each cache line based on the lower bits of both x and y coordinates. However, the support

for multiple reference frames of HEVC/H.265 is not efficient. Two cache lines with the same coordinates across different reference pictures will conflict to each other. Therefore, the cache performance is decreased due to these conflicts.

In order to solve this problem, another direct mapping method adds reference index to the mapping. Thus, a cache index consists of not only the lower bits of coordinates but also the reference index, as is shown in Fig. 54. The cache is equally divided into four cache districts and each reference frame is directly mapped to a separated district inside the cache to avoid the conflict across different pictures.

The drawback of direct mapping with reference index is the large cache memory size. In some severe cases, HEVC will support up to 16 possible reference frames. If the cache is designed for the worst case, it will consume a huge amount of memory resources. Compared to the general direct mapping, N times larger cache size is required where N is the number of possible reference frames, which becomes unacceptable despite of its cost-efficiency for the control logic.

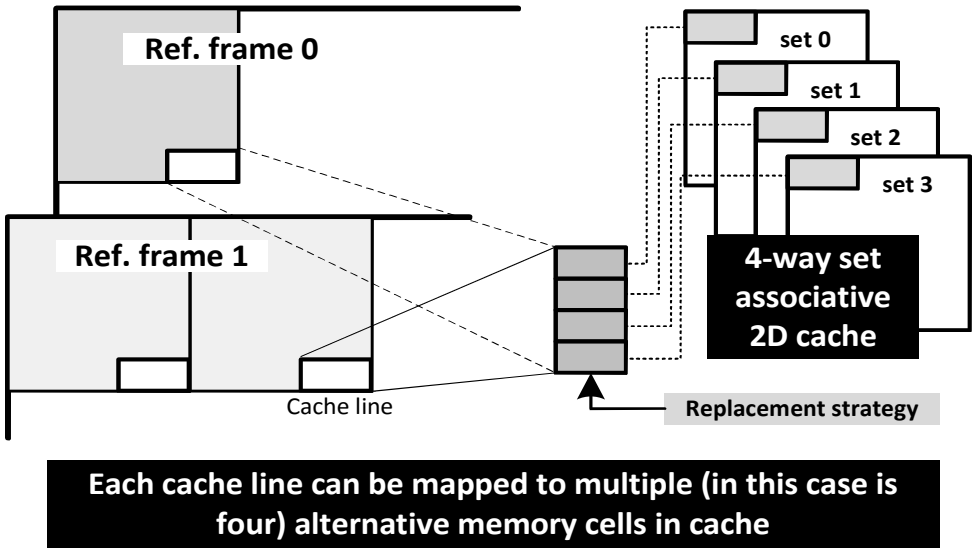


Fig. 55 An example of 4-way set associative 2D cache organization.

The multi-way set associative cache organization is efficient to address the multiple reference issue like [21]. Taking the 4-way cache in Fig. 55 for example, it maps each cache line based on its x and y coordinates. Different from the direct mapping, a cache line of the 4-way cache can be stored in any one of four alternative memory cells in the

cache. Replacement strategy should be considered as 4 alternatives are available when writing a cache line into cache.This redundancy is capable of relieving the conflicts caused by multiple reference pictures. Compared to a direct-mapped cache with the same size, this cache organization can usually achieve better performance at the expense of higher area cost due to the complicated control. In [21] where a 4-way set associative cache is employed, up to 126k logic gates are required for implementation.

Table 10 Cache size and hit rate performance on several mapping methods

| Test sequence: PeopleOnStreet, 2560×1600 | | |
|---|---|---|
| Cache Organization | Cache Size (pixel) | Hit rate |
| Direct Mapping | 128×128=16,384 | 45.0% |
| 4-way set associative mapping | 64×64×4=16,384 | 56.6% |
| Direct mapping with reference index | 64×64×4=16,384 | 52.7% |
| | 128×128×4=65,536 | 56.9% |

A simple comparison for the mentioned cache organizations are given in Table 10. Considering the same cache size (16k) for a fair comparison, the 4-way cache shows the best performance at the expense of high complexity. If I enlarge each district size of the direct-mapped cache with reference index from 64×64 to 128×128, the performance can be improved to be equivalent to the 4-way cache. This proves that the previous works are trading-off between memory resources and the hardware design complexities.

Based on the discussions, I propose the distance biased direct mapping scheme to simplify control logics without sacrificing the performance. The distance biased direct mapping is a kind of direct-mapped cache organization. A general example is shown in Fig. 56. Cache is divided into several cache districts, each of which can be mapped to a specific reference frame. The size of cache districts depends on the distances between the reference frames and current decoding frame. Similar to the direct-mapped cache, the cache index also consists of the reference frame index and the lower parts of x and

y coordinates. The distinction is that the cache district sizes for each reference frames vary, depending on the distance values between the reference frame and current frame. The details of distance biased direct mapping are discussed in the following subsections.

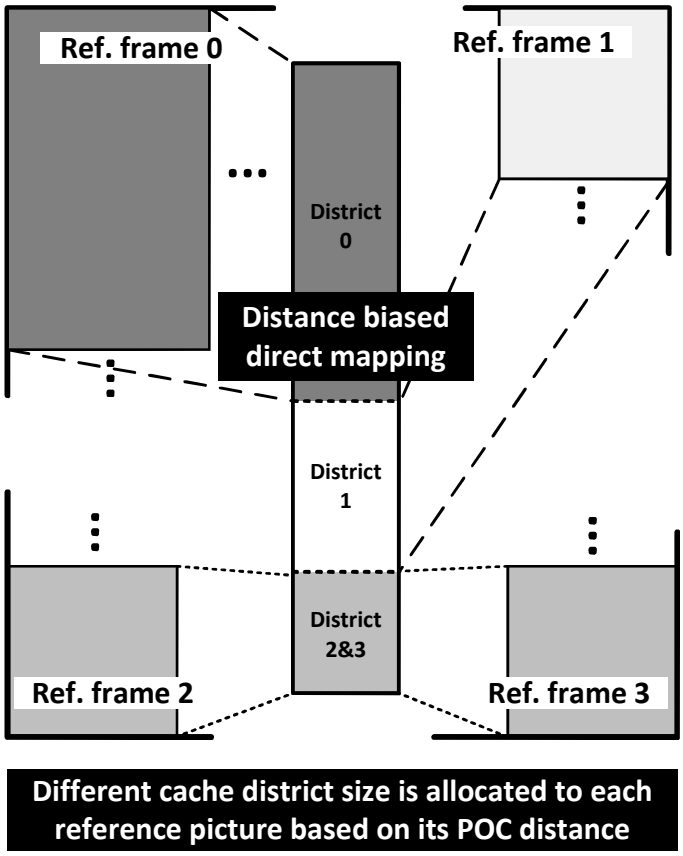


Fig. 56 An example of proposed DBDM 2D cache organization

From the viewpoint of hardware architecture, the proposed cache approach is like Fig. 57. Compared with the previous approaches as shown in Fig. 45 and Fig. 46, this approach proposed that the cache can be regarded to consist of a big physical cache set. The motivation is to use direct mapping for simplifying the hardware design complexity mentioned above. To solve the problem of multiple reference frames, the solution is to divide this big cache set into multiple logical cache sets. By doing this, each cache set is in charge of handling the reuse of reference from a specific reference frame. Therefore, no data flushing from different reference frames happens due to the separated storage in this approach. Moreover, because the multiple cache sets are logically divided, it can be on-the-fly programmed as will be discussed in the following sections in detail FIXME, so that the video features, which will be introduced in Section

3.3.1 and 3.3.2, can be used to improve the performance of cache.

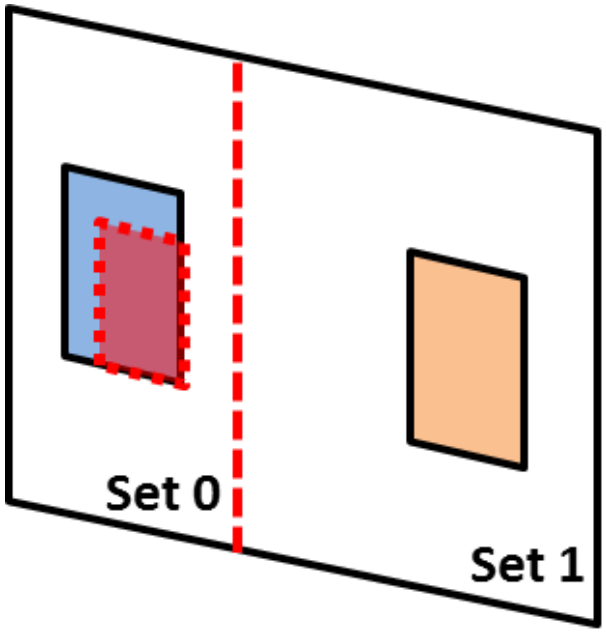


Fig. 57 The proposed cache approach compared with Fig. 45 and Fig. 46

## 3.3.1 Distance Biased Direct Mapping (DBDM)

Table 11 DRAM access distribution from each reference picture with different quantization parameter (QP)

| Test sequence: PeopleOnStreet, 2560×1600 | | | | |
|---|---|---|---|---|
| Distance | QP=22 | | QP=37 | |
| | Access # | Percentage | Access # | Percentage |
| 1 | 78578051 | 82.3% | 66304610 | 88.5% |
| 2 | 9787409 | 10.3% | 5389776 | 7.2% |
| 3 | 4003179 | 4.2% | 1690667 | 2.3% |
| 4 | 3080105 | 3.2% | 1532751 | 2.0% |

The first video feature proposed in this work is the relationship of distance and similarity between video frames. The distance is the absolute value of the picture order count (POC) difference between the reference frame and current decoding frame.

Generally, video consists of continuous frames which are captured in a short time. The contents of these frames are expected to be similar as shown in Fig. 58. Moreover, if two frames are closer to each other, the more similarity they may have. This is the first video feature used for cache design.

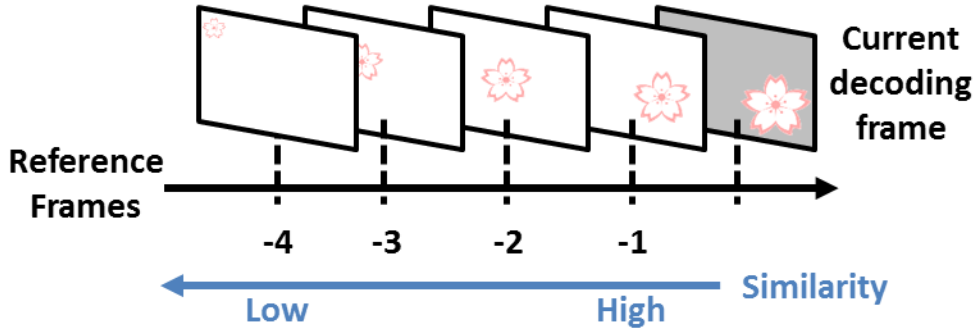


Fig. 58 The relationship of similarity and distance in a video

The experiments also prove the proposed video feature is feasible. I notice that pictures with smaller indexes in L0/L1 usually own smaller distance values due to the algorithm of constructing the reference list in HEVC/H.265. As the inter prediction exploits the temporal correlation between pictures, a picture with small distance should occupy a high probability to be chosen as the reference. Experiments were conducted for the lowdelay configuration, where the reference frame with smaller index always has a smaller distance. The number of DRAM requests without cache is counted. The results are shown in Table 11. The requests for the first reference frame account for the major proportion (more than 80%). On the other hand, only less than 5% of the total requests are for the picture with largest index.

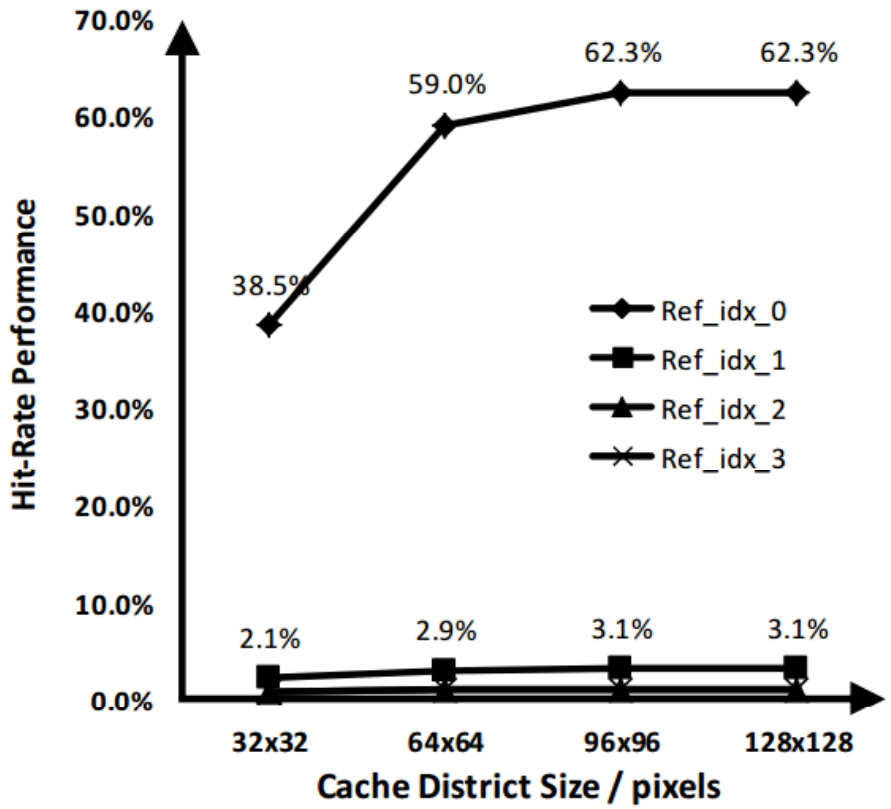


Fig. 59 The relationship between hit rate and cache size for Slice P.

The distance biased direct mapping is proposed to shrink the size of direct-mapped cache with reference index while keeping its hit rate performance. I have recognized a reference frame with a smaller distance leads to a higher requirement on memory bandwidth in the last paragraph. It inspires us to allocate different sizes of cache districts according to their distances. I first conduct experiments to observe the relation between the hit rate and the district size for each reference pictures under the lowdelay configuration. As shown in Fig. 59, Slice P has one reference frame list, in which distances of each frames are larger if the indexes are larger. For a single reference picture with direct mapping, increasing cache district size leads to an improvement on hit rate. The improvement of Ref_idx_0 is much clearer than the rest. On the other hand, only around 1% of the total DRAM requests are for Ref_idx_2 or Ref_idx_3. Therefore, a relatively larger cache district size should be allocated to Ref_idx_0, so as to guarantee the hit rate performance. On the other hand, a small district size can be assigned to pictures with large indexes to pursue a smaller cache size in total without

sacrificing performance. An example is given in Fig. 60 where the sizes of cache sets vary based on the importance in Fig. 59. These form the basic idea on the distance biased direct mapping.

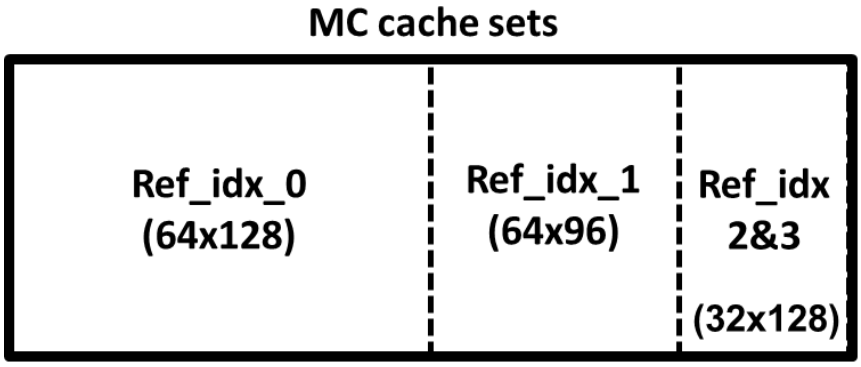


Fig. 60 An example of allocation of cache resources based on frame distances

## 3.3.2 Frame Structure Adaptive Mapping Scheme

The second video feature proposed in this approach is the reference frame structure. Fig. **58** illustrates an example of low-delay-P configuration where all the reference frames are prior to the current frames. In such a case, the similarity of these frames gradually decreases when they distance becomes larger. In other cases, the frame structure may be changed so that the reference frames may either prior to the current frame or after the current frame as shown in Fig. **61**. This section tries to utilize this video feature to further enhance the cache performance.

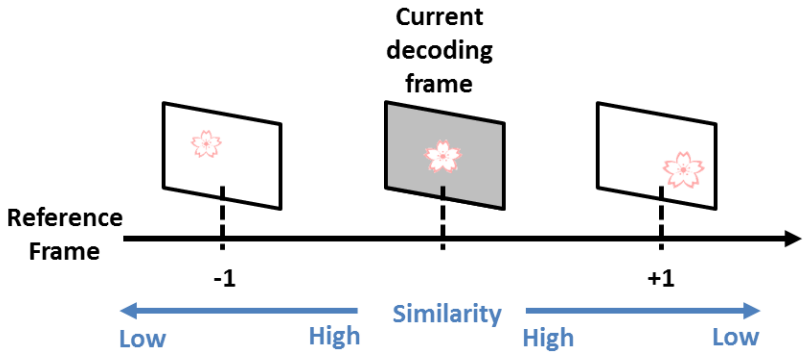


Fig. 61 An example of reference frames located prior and after current frame

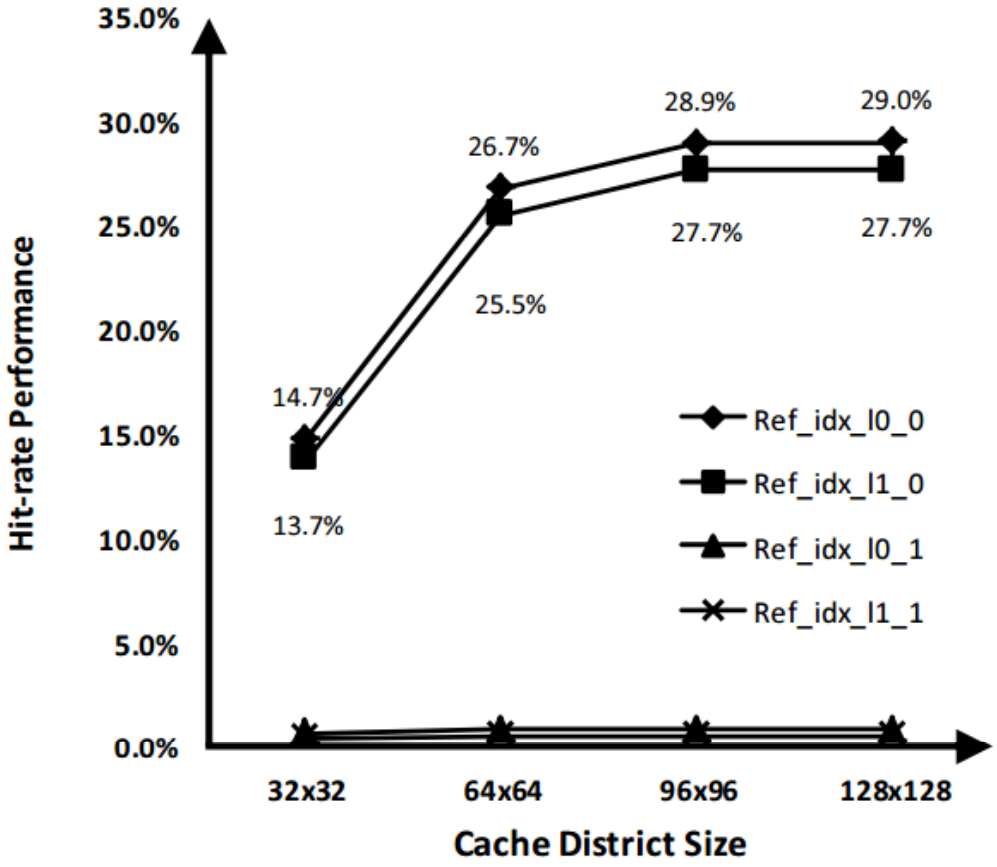


Fig. 62 The relationship between hit rate and cache size for Slice B.

I further propose the adaptivity based on the current frame structure. The frame structure refers to the content of reference picture list (RPL). The adaptation means cache mapping is dynamically adjusted based on the variation of RPL content. In detail, HEVC/H.265 support two slice type, slice_P and slice_B. They usually own different compositions of RPL. Slice_P in lowdelay configuration usually has two identical RPL, L0 and L1, while slice_B contains two reference pictures in each list and each list has the similar rule of Slice P. As shown in Fig. 62, two pictures with idx_0 from both L0 and L1 occupy most of the DRAM access. The results prove that smaller distance infers a larger proportion of the total DRAM accesses. The reason is that Ref_idx_l0_0 and Ref_idx_l1_0 are the closest pictures prior to and after the current decoding picture, respectively. These two pictures should both be allocated with larger cache district sizes, which is quite different with slice_P. When my proposed MC cache decodes a video sequence, it can adaptively adjust its mapping strategy based on current decoding slice

type to achieve a better hit rate performance.

### 3.3.3 Simulation Results for Distance Biased Direct Mapping

For the default configuration in HEVC/H.265, DBDM is designed to be a size of 18,432 pixels (576 8×4 cache lines) and is divided into three districts as shown in Table 13. For slice_P, the first and the second largest districts are assigned to the first two pictures in the reference list. The last district is shared by the rest of two reference frames. Note that cache district sizes contain a number of 96, which is not a power of 2. When calculating a tag or index of a cache line, the division of 96 will be involved. It is realized by right-shifting dividend with 5 bit and then dividing three. Division of three is achieved by LUT whose input is 7-bit width for a 7680×4320 format.

Experiment results in Table 12 show that the hit rate performance of DBDM can achieve around 10% improvement compared to the direct mapped cache with equivalent cache size, since the proposed DBDM is more efficient in handling with the multiple reference pictures. Meanwhile, the performance of DBDM is comparable to that of the 4-way set associative cache, despite the direct mapping of DBDM.

Table 12 Simulation and comparison results for the proposed DBDM.

| Seq. Class | Config. | Hit rate performance | | | |
|---|---|---|---|---|---|
| | | Direct mapping (16,384) | 4-way (16,384) | Proposed DBDM (18,432) | Improvement over direct mapping |
| Class A | LD22 | 63.0% | 67.7% | 67.2% | 4.3% |
| | LD37 | 66.6% | 67.7% | 67.6% | 1.0% |
| | RA22 | 49.1% | 60.1% | 59.1% | 10.0% |
| | RA37 | 53.0% | 58.6% | 58.1% | 5.1% |
| Class B | LD22 | 67.0% | 72.3% | 71.9% | 5.0% |
| | LD37 | 66.7% | 67.7% | 67.6% | 0.9% |
| | RA22 | 50.2% | 63.3% | 62.4% | 12.2% |
| | RA37 | 52.9% | 58.4% | 58.3% | 5.4% |
| Class C | LD22 | 58.5% | 66.1% | 65.4% | 7.0% |
| | LD37 | 62.6% | 64.1% | 64.0% | 1.4% |
| | RA22 | 44.8% | 59.6% | 58.8% | 14.0% |
| | RA37 | 49.4% | 56.3% | 56.0% | 6.7% |
| Class D | LD22 | 57.8% | 69.2% | 68.3% | 10.6% |
| | LD37 | 64.3% | 66.5% | 66.3% | 2.0% |
| | RA22 | 40.1% | 63.3% | 62.0% | 22.0% |
| | RA37 | 48.0% | 59.0% | 58.6% | 10.5% |
| Class E | LD22 | 47.6% | 50.5% | 50.4% | 2.8% |
| | LD37 | 38.6% | 39.0% | 38.9% | 0.4% |
| | RA22 | 33.1% | 42.2% | 42.0% | 8.9% |
| | RA37 | 23.2% | 26.0% | 26.1% | 2.9% |
| Class F | LD22 | 34.8% | 36.7% | 36.5% | 1.7% |
| | LD37 | 33.6% | 34.2% | 34.1% | 0.5% |
| | RA22 | 25.3% | 29.8% | 29.4% | 4.1% |
| | RA37 | 24.5% | 26.8% | 26.5% | 2.0% |

### 3.3.4 Design Overheads of This Section

Table 13 A cache district division example for DBDB

| Slice Type | Ref. Picture | Cache district size (pixel) | |
|---|---|---|---|
| Slice_P | Ref_idx_0 | 64×128 | 18,432 |
| | Ref_idx_0 | 64×96 | |
| | Ref_idx_0 | 32×128 | |
| | Ref_idx_0 | | |
| Slice_B | Ref_l0_idx_0 | 96×96 | 18,432 |
| | Ref_l1_idx_0 | 64×96 | |
| | Ref_l0_idx_1 | 32×96 | |
| | Ref_l1_idx_1 | | |

The complex for hardware implementation of distance biased direct mapping cache is higher than the previous works. In the design phase, the distance biased direct mapping is implemented by a reconfigurable Look-Up-Table. Designers can flexibly initialize the content of Look-Up-Table based on their design demand. Thus, the distance biased direct mapping is capable of handling different configurations of frame structure. Look-Up-Tables are utilized to store the parameters of each cache districts (width and height, district offset inside cache). These parameters are important to address the cache structure (tag, index, offset). For example, the cache district division in Table 13 corresponds to the content of Look-Up-Table as shown in Fig. 63.

When checking whether a cache line is hit or miss, I can first calculate the location of this cache line by using motion information. Then, the index of the cache line is generated by the lower bits of coordinates and the reference frame index. The cache district offset value is then added to generate the index, like the formulas in Fig. 63. Meanwhile, the tag information combines the rest of the coordinate bits. The definition of the lower bits is the modulus of dividing the coordinates by the cache district size. Especially in my proposed distance biased direct mapping, the cache district size is distance biased and can be read from the Look-Up-Table. Finally, tag memory is read

with the calculated index and the read data are matched with the calculated tag to decide a cache hit or miss. This is the basic processing flow of the proposed cache organization.

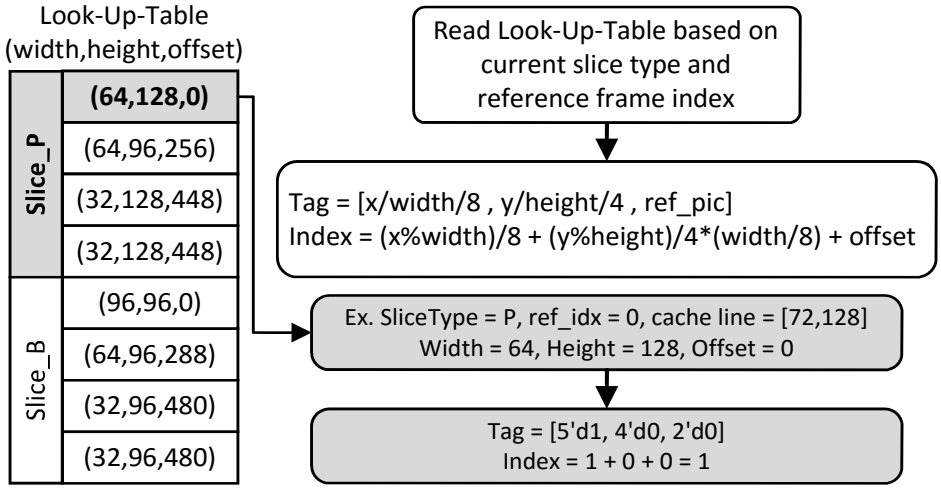


Fig. 63 An example of Look-Up-Table for cache address calculation

# 3.4 Optimization for High Integration Density

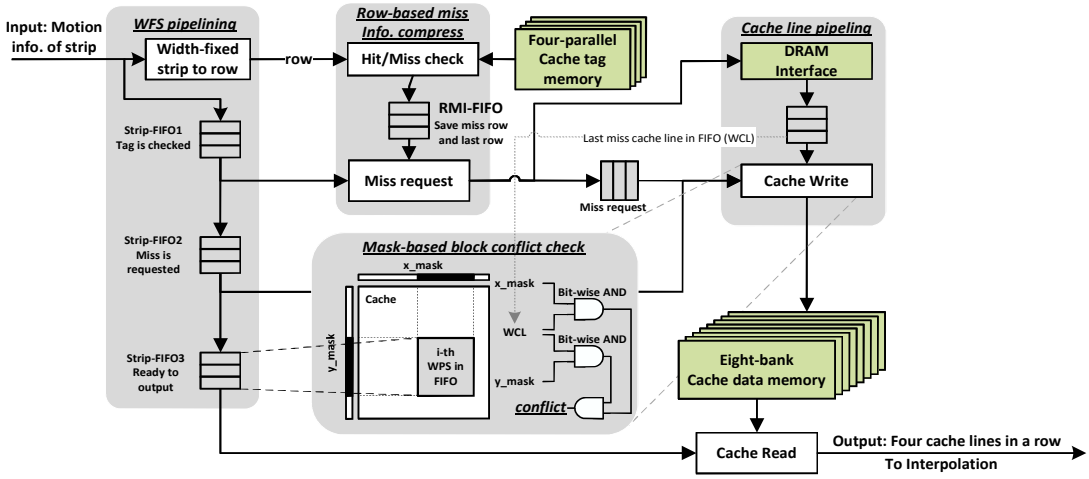


Fig. 64 Detailed Block diagram on the proposed MC cache architecture

Fig. 64 illustrates the details of MC cache. Every input motion information of width-fixed strips will be buffered throughout the whole process by three Strip-FIFOs in a relay manner. In detail, all the cache lines of strips in Strip-FIFO1 are checked. This process may find several missed cache lines. When all missed cache lines have been requested to the DRAM, this strip will be transferred from Strip-FIFO1 to Strip-

FIFO2. Strip-FIFO3 stores strips which are ready for output. It means all the missed data in this FIFO has been stored in the cache. The strips will be kept in Strip-FIFO3 until all the reference data has been output to the interpolation.

## 3.4.1 Bank Organization of Cache Memory

### *3.4.1.1 Tag Organization*

The tag memory consists of four banks. Each tag information corresponds to a 8×4 block cache lines. Four banks support up to four cache lines (32×4 block) in a row to be checked simultaneously. This helps to improve the throughput due to the assumption that more cache hits occur

than misses. The last two bits of the index of a cache line indicate which bank the current MAU belongs to. The tag information will be updated immediately as soon as a cache hit occurs so that following tag checking can be pipelined without having to wait until the corresponding pixels in the data memory are really updated.

### *3.4.1.2 Eight-bank Data Memory*

A DRAM MAU has been defined as 8×4 pixels containing both luma and chroma components, which should be written into the cache in one cycle. Meanwhile, the interpolation filter needs reading 15×4 pixels per cycle from the cache.

Generally, the cache data memory is designed to be consistent with the tag memory, consisting of four banks. Each bank has a throughput of 8×4 pixels. Hence, totally 32×4 pixels can be read out from cache, among which only 15×4 pixels are used by the interpolation. These 15×4 pixels are not aligned and could locate inside anywhere of these 32×4 pixels. A barrel shifter which can achieve circular shift mechanism is used to extract the specified 15×4 useful data. The shift offset ranges from 0 to 15, which is able to be expressed by four bits. Thus I have organized the barrel shifter with four layers. The first layer handle with the most-significant bit to decide whether to shift the input 16×4 pixels by eight. Similarly, the last layer is for the least-significant bit with a shifting value of one. However, concerns are voiced for this architecture. As the useful

data only account for 46.9% of 32×4 pixels, the barrel shifter has to filter out more than half of the input data. This causes the barrel shifter to be implemented by a large amount of multiplexer, leading to a huge area cost.

To resolve the concerns, the proposed memory consists of eight banks, each of which supports a throughput of 2×4 pixels containing both luma and chroma parts. It can reduce the cache output ports from 32×4 to 16×4 without affecting the performance. Thus, the proportion of the useful data increases from 46.9% to 93.8%. The specified 15×4 pixels can be easily selected from the 16×4 input pixels. Obviously, the circular shift for this architecture can be quite smaller than that of the four banks solution. On the other hand, this architecture does not affect the writing requirement. A DRAM MAU can be written into specified four of eight banks based on the cache line address.

### 3.4.2 Row-based Miss Information Compression

Generally, every cache miss will produce a reading request to the off-chip DRAM, which requires the corresponding location information to be buffered in FIFOs. Many previous works like [21][25] check cache miss with a processing unit of cache line. In detail, in each cycle a cache line request produced by MC core will be processed by checking whether its tag is the same as that in tag memory. In case of a cache miss, a reading request to the external DRAM is produced. In my consideration, this data structure is not efficient for the hardware design. Firstly, a much more notable amount of cache misses would appear because of the higher throughput requirement of UHDTV. This significantly increases the internal memory size. Secondly, the four-parallel tag memory supports four cache lines to be checked simultaneously and each of them can be hit or miss. This means that the number of produced requests is uncertain, ranging from 0 to 4. The uncertainty is not friendly to the hardware implementation which has to be designed for the worst case. I ha

-based Miss Information Compression scheme is proposed to tackle with the above concerns. I define the row as the four cache lines which are checked together.

Row-based miss information means that the miss information is represented based on the row, instead of cache line. Row-based Miss Information Compression of each row consists of two parts. One is the 5-bit offset of the row, indicating the distance between the current row and the first row inside this width-fixed strip. The other is the 4-bit status flags, representing the hit or miss status for these four cache lines. According to these nine bits, rows are classified into three types, miss row, all-hit row and last row, as shown in Fig. 65.

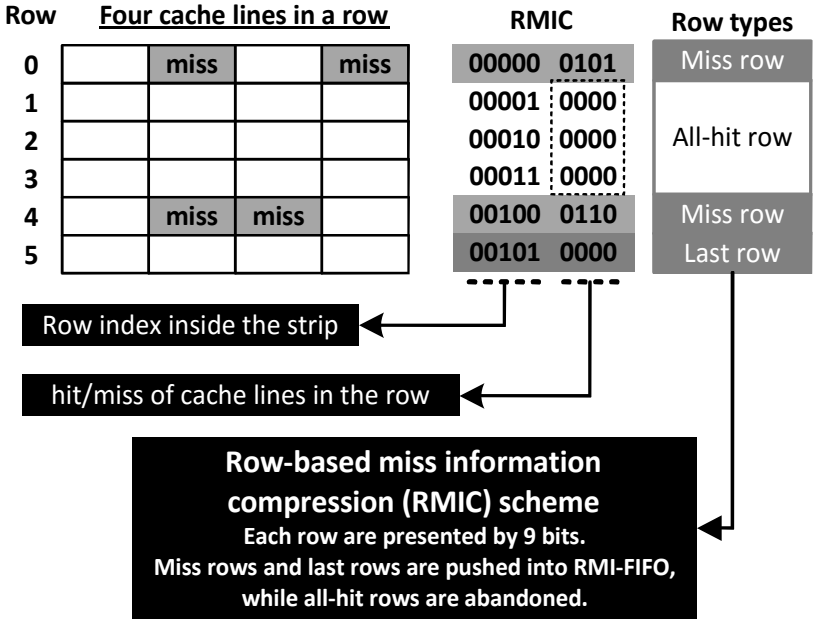


Fig. 65 An example of row-based miss information compression.

Row-based miss information compression is capable of compressing the miss information. Firstly, row-based information itself can be regarded as a compression compared to the cache line based one, when multiple misses exist inside a row. Secondly, I can expect more all-hit rows due to distance-biased direct mapping. The row-based miss information compression of an all-hit row is meaningless because it contains no reading request to the DRAM so I will not buffer them. Only the miss rows require to be buffered. As a supplementary step, the last row of a width-fixed strip should always be buffered even if this row is an all-hit row, to guarantee that following modules in MC cache can recognize that this is the end of the strip. The reason is like this. Based on the block diagram in Fig. 64, the Miss Request module would receive the

information asserting the last row of a strip. This information indicates that all the cache lines of the current processing strip have finished matching with the content of the tag memory. Therefore, I can use this information to trigger Strip-FIFO1 for pulling this strip out and Strip-FIFO2 for pushing the strip in. If the last row information is excluded from the mechanism, I consider a corner case where all cache lines of a strip are hit. In this case, the row-based miss information compression scheme will not produce any information to the following Miss Request module. No information means no pull operation of Strip-FIFO happens which should happen, resulting in errors of the total system.

### 3.4.3 Mask-based Block Conflict Check

The pipeline is widely employed to eliminate the long DRAM accessing latency. Because of pipeline, different modules might work in different stages. Specifically, cache read module and cache write module may process two different strips at a certain moment. All the strips in Strip-FIFO3 are ready to be fetched from the cache. If cache write module writes a missed MAU into the cache, whose address coincides with that in Strip-FIFO3, a conflict will occur.

In many previous works, this conflict is checked as a line-to-line manner. The first line in (line-to-line) means a missed cache line (MCL) which is about to be written into the cache. The second line in (line-to-line) implies all the cache lines of width-fixed strips in Strip-FIFO3. The address of MCL has to be compared to the addresses of all cache lines for conflict checking. Note that all the comparisons should be done in parallel. Because of the pipeline, I can infer that there would be several strips in Strip-FIFO3, each of which contains several cache lines. This would lead to a huge area cost and a long critical path.

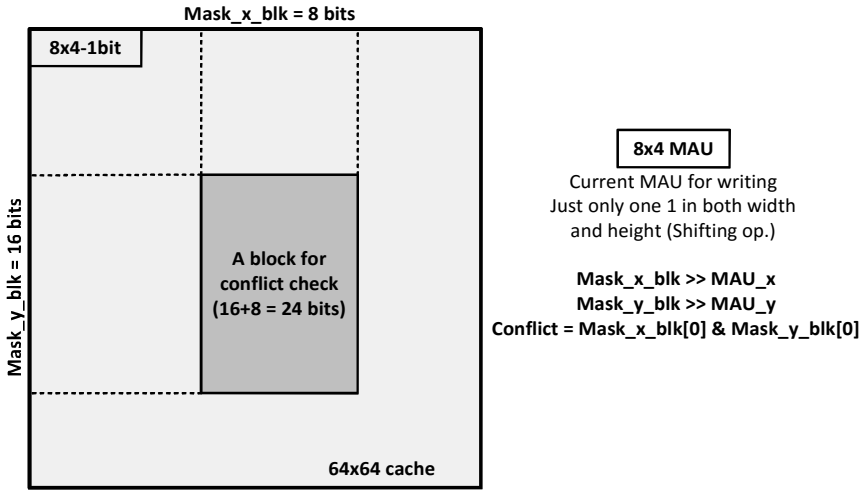


Fig. 66 Mask calculation in Mask-based Block Conflict Check

I propose the mask-based block conflict check scheme as shown in Fig. 66. The mask-based block conflict check is based on a line-to-block architecture scheme. A block means the strips in Strip-FIFO3. If the missed cache line overlaps with any strips, a conflict is detected and cache write module is pended until the related strip is popped out from FIFO. Because the number of strips in Strip-FIFO3 must be smaller than that of cache lines, the circuit area for the line-to-block scheme should also be smaller.

The block mask is introduced in the mask-based block conflict check. A block mask designates the region of a strip in Strip-FIFO3 within the corresponding cache district, which cannot be modified until this strip is popped out. Since a cache district can be defined by its width and height, the block mask is designed to consist of x-mask and y-mask correspondingly, as shown in Fig. 66. Taking the x-mask for example, if I assume cache width is 96 and the width of cache line is 8, then x-mask would consist of 12 bits, in which a sequence of ones (1s) exists. These ones designate the interval as being the mapping of the width of strip. For example, a mask of 0x007 means the width of current strip is mapped into the interval from 0 to 23. Instead of directly comparing the coordinates of the cache line and every strips, I do the bitwise AND operation between the current line and every block masks to reduce the area cost. If the result is not all-zero, a conflict is detected.

## 3.5 Experimental Results

The proposed MC has been implemented by Verilog HDL. Simulation has been performed on both register-transfer level and post-synthesis level. Both the low-delay and random-access configurations are tested with Quantization Parameter 22 and 37 for verifying the hardware's correctness. I have mentioned in Section 3.2.2 that theoretically 5.69 pixels should be processed under 350MHz clock frequency. Therefore, I implemented the highly parallel interpolation architecture with the fixed 8×4 prediction pattern inside the width-fixed strip pipeline in hardware level. This allows us to be capable of predicting 5.8 pixels/cycle even in the worst case shown in Table 9, which is sufficient to support the MC core for 7680×4320@60fps specification at 342MHz.

The RTL design is synthesized with SMIC 40nm CMOS technology with Synopsys Design Compiler. The result shows that the total gate count for 8/10-bit is 419.0k (equivalent to 2-input NAND gate) and the achieved maximum clock frequency of my proposal can achieve 400MHz. In detail, the major part of the area cost is from the interpolation because of the high parallelism for 8K UHDTV application. About 297.3k gates are consumed for its implementation. 62k logic gates are taken to realize the cache-related parts, containing the proposed distance-biased direct mapping cache and several optimizations discussed in Section 3.4. Meanwhile, 276.5kb on-chip SRAM memory for 10-bit storage is utilized for the cache data memory.

Table 14 shows the comparison with previous works. Compared to the H.264/AVC approaches, implementations on HEVC/H.265 present a relatively larger cost on logic gates due to the more complicated algorithm like 8-tap interpolation. In comparison to the HEVC approaches, I mainly give a detailed discussion in the following about the cache and interpolation parts since previous works do not involve the description of weighted prediction though it closely tights with the motion compenstaion. Therefore, the logic gates and memory resource utilizations are over-used because this work supports the weighted prediction function.

Table 14 Comparison with state-of-the-art works

| | | Li [39] ISCAS'07 | Azevedo[40] ISCAS'07 | Zhou [38] IEICE'11 | Tikekar [25] JSSC'14 | Cho [47] JCTVC'13 | Chiang [46] TCSVT'15 | This work | |
|---|---|---|---|---|---|---|---|---|---|
| Standard | | H.264/AVC | H.264/AVC | H.264/AVC | HEVC/H.265 WD4 | HEVC/H.265 WD6 | HEVC/H.265 Final | HEVC/H.265 Final | |
| Format | | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8/10-bit |
| Throughput | | 1920×1080 @30fps | 1920×1080 @30fps | 3840×2160 @60fps | 3840×2160 @30fps | 1920×1080 @60fps | 4096×2160 @30fps | 7680×4320 @60fps | 7680×4320 @60fps |
| Frequency | | 100 MHz | 100 MHz | 166 MHz | 200 MHz | 266 MHz | 270 MHz | 342 MHz | 342 MHz |
| Technology | | 180nm | FPGA | 90nm | 40nm | 65nm | 90nm | 40nm | 40nm |
| Logic Gate Count | Cache | 13.03k | N/A | 37.6k | 90.4k | 440k | 47.1k | 62.0k | 62.0k |
| | Interpolator | | | 71.2k | 69.4k | | 34.6k | 243.3k | 297.3k |
| | WP | N/A | | N/A | N/A | | N/A | 48.7k | 59.7k |
| | Normalized Gate Count | $2.08\times10^{-4}$ Gate/(pixel/s) | N/A | $2.19\times10^{-4}$ Gate /(pixel/s) | $6.42\times10^{-4}$ Gate /(pixel/s) | N/A | $3.08\times10^{-4}$ Gate /(pixel/s) | $1.53\times10^{-4}$ Gate/(pixel/s) | $1.80\times10^{-4}$ Gate/(pixel/s) |
| Memory | MC cache | 16kb | N/A | 24.6kb | 131.0kb | 400kb | 16.4kb | 221.2kb | 276.5kb |
| | Misc. | | | | 69.4kb | | 9.8kb | 76.0kb | 88.3kb |
| | Normalized Memory | $0.26\times10^{-4}$ Bit/(pixel/s) | N/A | $0.49\times10^{-4}$ Bit/(pixel/s) | $8.05\times10^{-4}$ Bit/(pixel/s) | $32.2\times10^{-4}$ Bit/(pixel/s) | $0.99\times10^{-4}$ Bit/(pixel/s) | $1.49\times10^{-4}$ Bit/(pixel/s) | $1.83\times10^{-4}$ Bit/(pixel/s) |
| Power | MC core | N/A | N/A | N/A | 6.32mW | N/A | N/A | N/A | 48.12mW |
| | MC cache | | | | 14.17mW | | | | 13.54mW |
| | Normalized Power | N/A | N/A | N/A | 82.37pJ/pixel | N/A | N/A | N/A | 30.97pJ/pixel |

The conception of normalized gate count is introduced for a fair comparison since

the logic gate cost is proportional to its maximum throughput. It is calculated by dividing the gate count of cache and interpolation parts by the throughput. As is shown in Table 14, compared to [43], my proposal achieves at least 2.01× area efficiency based on the normalized gate count despite the increased complexity of 8× throughput requirement. This area efficiency benefits from two parts. Firstly, the proposed cache is capable of supporting the data delivery requirement of 7680×4320@60fps sequences at the equivalent area cost of other works, due to the direct mapping of distance-biased direct mapping scheme and optimizations like row-based miss information compression and mask-based block conflict check discussed in Section IV and V, respectively. Secondly, the high parallelism design of interpolation helps to eliminate the hardware redundancy by exploiting the possible data sharing, such as reference data and sharing information of luma and chroma components. Meanwhile, the interpolation part contains an area-efficient circular shift owing to the eight-bank 2×4 memory structure in Section 3.4.1. In total, the proposed MC cache architecture can realize a near-optimum hit rate and area-efficient performance although a relative large size of memory is utilized.

I further define the concept of the normalized memory. From the Table 14, the memory requirement for HEVC is much more intensive compared to that of H.264 like [26], due to the new coding tools like larger coding unit and 8-tap interpolation filter. For the HEVC works, my memory shows a better efficiency in the normalized memory usage except [43]. [43] does not provide a detailed discussion on their cache architecture. I a

I think this cache is not capable of real-time decoding an 8K@60fps video sequence unless the size could be enlarged so as to reduce the intensive bandwidth requirement for 8K UHDTV. By contrast, my proposed MC cache has proved to achieve near-optimal performance with an efficient memory cost.

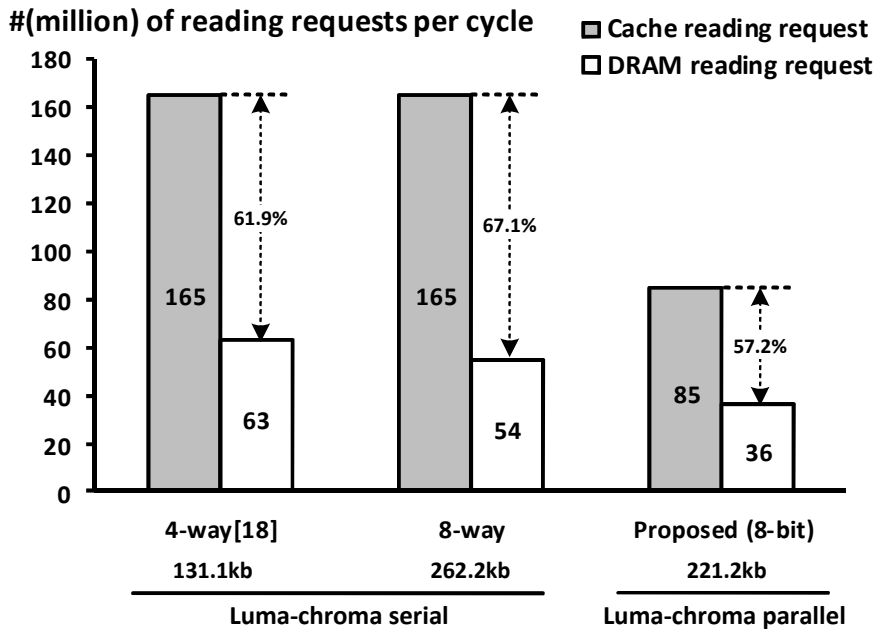


Fig. 67 Cache workload comparison with [21]

The power consumption results are also provided in Table 14. As far as I know, only Tikekar et al. in [21] provided the power consumption results of a whole decoder. To make a relatively fair comparison, I calculate the power consumption of corresponding modules (MC core and MC cache) according to the logic gate counts. In my architecture, I employ an eight-bank 2x4 memory structure as cache memory to double the data delivery efficiency, which is supposed to be helpful to save power. The results prove that my proposed MC architecture is more efficient than [21] and 62.4% of the normalized power consumption can be saved.

In Fig. 67 I present the comparisons of MC cache workload across three cache organizations. The gray and white bars respectively present the numbers of reading and writing requests for the cache. Compared to [21], my proposed the distance-biased direct mapping cache shows a lower workload by around 50%. The reason why [21] involves a higher workload comes from two aspects. The first one is the serial process of luma and chroma components, which theoretically produces around half more requests than the one in parallel. Secondly, the pipeline granularity is defined as 16×16, which is not efficient for large PU as discussed in Section 3.2.2. A larger workload per

unit time means a higher throughput requirement, which generally costs more logic gates for the hardware implementation. Meanwhile, the more workload also means the power consumption is larger. Additionally, I further propose several hardware optimizations like row-based miss information compression and mask-based block conflict check to reduce the area cost. In total, the logic gate count of my proposed cache is 62.0k, which consumes similar area cost with others [21][43] despite the increased data throughput.

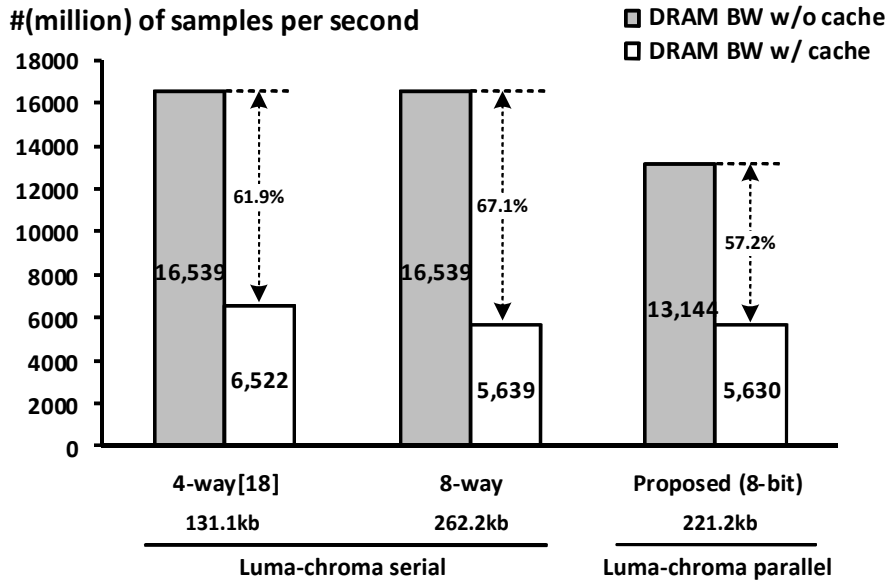


Fig. 68 DRAM bandwidth comparison with [21]

Fig. 68 depicts the bandwidth requirements and hit rate performances. Although the 4-way set associative cache of [21] shows a higher hit rate and a smaller cache size, the bandwidth requirement is still larger due to the more DRAM reading requests shown in Fig. 67. I conclude the main reason as follows. Theoretically, a 4-way set associative cache is capable of handling the multiple reference frames in HEVC/H.265 in which four reference frames are supported in default configurations. However, [21] processes luma and chroma components serially and they share the same cache. To guarantee the performance, the cache associativity should be further increased. Therefore, I add a simulation on an 8-way set associative cache organization and show the result in both Fig. 67 and Fig. 68. I can find that the 8-way set associative cache organization can provide a similar bandwidth requirement to mine. However, this performance

improvement is at the expense of doubling the cache size compared to [21]. The 262.1kb of the 8-way set associative cache is 18.5% larger than mine. Moreover, as the logic gate count of the cache control part reflects the complexity, the 8-way cache would consume much more logic gates than the 4-way one. This makes the efficiency of the cache part worse.

## 3.6 Summary

This chapter presents a motion compensation architecture which is capable of real-time decoding 7680×4320@60fps video sequences at 342MHz clock frequency. It significantly improves energy efficiency with several novel architecture optimizations, as summarized below:

1) A distance biased direct mapping scheme is proposed to realize a high-performance motion compensation cache that can achieve a near-optimum hit rate close to that of a similarly sized multi-way associative cache. In the meanwhile, the proposed cache involves significantly lower complexity by being direct mapped.
2) An eight-bank 2×4 memory structure doubles the data delivery efficiency of the cache by providing different word length in the input and output ports, which saves area and power of the on-chip memory.
3) Width-fixed strip based pipeline design maintain the data reuse between adjacent pipeline blocks, enhancing the data reuse and reducing the redundant calculations to achieve the optimal data reuses in the vertical direction.
4) In implementation of the MC cache, row-based miss information compression is applied to reduce the size of FIFOs in the pipeline. Mask-based block conflict check is also proposed to efficiently detect the pipeline hazards.

Both temporal and spatial locality are exploited by the distance biased direct mapping cache for achieve both computing and memory energy efficiency. The first and second contributions come from this idea. Compared to previous works, this scheme further utilizes the features of video to improve the design. Moreover, the

temporal locality of horizontal interpolation results is also maintained in the proposed width-fixed strip based pipeline design to remove the 30% redundant computations for energy saving. Finally, some hardware optimization schemes in the last contribution point are proposed to further reduce the computing energy.

The demerits of the proposed architectures are mainly the more consumption of logic gates due to the following reasons. Firstly, the required logic gates are significantly increased to support the high throughput requirement of 8K applications. Secondly, the width-fixed strip pipeline design requires a sophisticated controller as the height of strips is variant. Thirdly, the circuit for calculating the addresses is becoming complicated compared to the direct mapping as the cache district can vary according to the design demands, which may introduce the area-consuming division operations into the circuits. Finally, the increased logic gates are used for supporting more functions like 10-bit process and weighted prediction, which cannot be supported by the previous work like [21].

In conclusion, the system is pipelined with the proposed width-fixed strip pipeline granularity and is highly parallel designed so that 8×4 pixels can be processed simultaneously. By applying these schemes, around 60% of redundant DRAM accesses can be eliminated with a smaller control logic gate count due to the direct mapping. To further reduce the area cost, several optimizations have been presented based on hardware implementation. The row-based miss information compression helps to compress the miss information and mask-based block conflict check has been designed to achieve an efficient conflict checking circuits. By employing these approaches, my MC design consumes 419.0k logic gates and 364.8kb on-chip SRAM for 10-bit application in total, proving a better performance and the efficiency on area cost.

# 4. Conclusion

## 4.1 Summary of This Dissertation

In order to achieve the big target of HEVC decoder for 8K UHDTV, this dissertation focuses on the inter prediction related functions for energy efficient VLSI a

It contains two important modules, decoding motion vector (parameter decoder in Chapter 2) and decoding inter prediction samples (motion compensation in Chapter 3).

To improve the energy efficiency, the basic motivation to investigate the potentials from a data reuse point of view for these two modules. This work discovered unexploited data reuse potentials and achieved energy reduction in both computing and memory parts. The detailed summary of this work is as follows:

In Chapter 1, the preliminary of this dissertation is first given. The new features of HEVC are first introduced and related VLSI design challenges are discussed for both motion vector decoding and motion compensation. For 8K HEVC applications, the high throughput requirement and energy efficiency are the main problems for designing an efficient architecture.

In Chapter 2, a unified parameter decoder architecture for 8K UHDTV applications is designed with the proposed block merging idea for data reuse. Block merging can reduce block shapes to reduce hardware costs and maximize data reuse to reduce energies for computations by at least 50% (50% is the case where 8x8 CU contains two PUs). The design can accomplish the algorithm of MV and BS calculation for sharing the memory and logic resources. In particular, CU-based pipeline strategy is the approach to implement block merging to simplify control logic as well as supporting HEVC's new coding tools. Moreover, on-chip line buffer and cyclic SRAM are designed for both spatial and temporal reference storage to guarantee enough bandwidth requirements. PU-based coding scheme can help reduce around 30% of DRAM bandwidth. Finally, optimization on irregular algorithm is adopted for 43.2k logic gates

reduction. In total, the proposed unified parameter decoder supports real-time video decoding for 7680×4320@60fps application at 249MHz in worst case with 36% reduction on the logic gate cost for reducing compute energy compared with the state-of-the-art works. The demerit is the more area consumption for implementing complicated hardware to support the diverse block types in HEVC.

In Chapter 3, a motion compensation architecture for decoding inter prediction samples is presented with the proposed distance-biased cache. This cache can improve cache reuse possibility based on the proposed idea of distance. In detail, the system is pipelined with the proposed width-fixed strip pipeline granularity and is highly parallel designed so that 8×4 pixels can be processed simultaneously. Meanwhile, the distance biased direct mapping scheme for cache organization achieves equivalent hit rate performance to relieve the bandwidth pressure on DRAM. By applying this scheme, around 60% of redundant DRAM accesses can be eliminated with a smaller control logic gate count due to the direct mapping. To further reduce the area cost, several optimizations have been presented based on hardware implementation. Row-based missing information compression helps to compress the miss information and Mask-based block conflict check has been designed to achieve an efficient conflict checking circuits. By employing these approaches, my MC design consumes 419.0k logic gates and 364.8kb on-chip SRAM for 10-bit application in total, proving a better performance and the efficiency on area cost. Compared with the state-of-the-art works, this design achieves 76%, 81% and 62% improvement in terms of logic gate, memory requirement and energy consumption. The demerit is the increased logic gate costs for supporting the reconfigurability of cache.

In total, the proposed architectures can support most of the basic coding tools for HEVC inter prediction defined in HEVC Version 1, profile Main and Main 10 [12], as shown in Table 15. For example, the CTU sizes can range from 16 to 64 and all the possible partitions defined in [5] can be ideally supported by the proposed architectures. Meanwhile, the supported chroma subsampling is 4:2:0 and the precision of each sample can be 8/10-bit. Generally, if a video is encoded with HEVC version 1, profile Main and Main 10[12], this work can be directly employed for decoding inter predicted

samples without further efforts.

Table 15 Supported HEVC coding tools by this dissertation

| **Coding Tools** | **Supported in this dissertation** |
|---|---|
| CTU Size | 16, 32, 64 |
| CU Size | 8, 16, 32, 64 |
| PU Partition | Symmetric/Asymmetric partition |
| Prediction Mode | Merge mode, AMVP mode |
| Interpolation Filter | 7/8-tap for luma and 4-tap for chroma |

However, there are still some remaining problems in the current design. I will discuss them from two views.

From the view of this research themes (PDec and MC), there are still potentials for better efficiency in terms of area and memory costs. For example, the on-chip memory resources in Chapter 2 are increased from 7.2k (normalized from 4K to 8K UHDTV) to 23.0k for data reuse to reduce expensive off-chip memory energy. In Chapter 3, the area costs are also increased compared to [19] due to the complicated cache control circuits for implementing the distance-biased cache. These increased costs not only increase the fabrication costs, but also consumes more leakage power, which might be a problem if CMOS technology cannot scale down these energies in the future [7]. Therefore, it is meaningful to give a further discussion on how to reduce area costs as well as improving energy efficiency.

From the view of the big target (w

as unsolved problems for its better visual experience which might be demanded in the near future.

## 4.2 Future Works

Based on the above summaries, I conclude that there are some meaningful future works in three aspects, each of which is discussed from a data reuse point of view.

First, future works for this research themes could try to find better data reuse techniques for hardware architecture designs. Currently, the better energy efficiency is at the expense of more area costs because more on-chip memory resources are employed for buffering these reused data. For example, one possible solution for reducing memory resource requirements in Chapter 2 is to encode the data before storing them into memories. This can save the memory resources at low costs if a proper coding scheme is found. A possible solution for increased area costs in MC cache is to optimize the high-cost division in current architectures. For example, we can still use distance-biased ideas to divide the cache into sets, whose sizes can be limited to power of 2 to omit the usage of divider. This might help us to reduce the increased demand for the on-chip resources as well as achieving better energy efficiency.

Second, future works for the big target could be discussed in three parts. (a) It is possible to improve energy efficiency of other modules. Some modules like intra prediction or sample adaptive offset in HEVC decoder have the reuse potential in their computing algorithms. For example, the data fetched for intra prediction need to be smoothed for pre-processing. If the same data are fetched several times, pre-processing will be repeatedly executed for each fetch, which consumes more energies. Another meaningful work is to focus on other energy consuming modules like inverse transformation, which dominants the second most energy consumptions. (b) It is also important to find solutions toward a higher throughput like 8K@120fps. One possible solution is to investigate multi-core solutions where each core can inherit the results of this thesis. A new L2 cache is expected which can enhance the data reuse among cores. (c) Moreover, it is also meaningful to explore architectures for other HEVC profiles

instead of profile Main and Main 10. For example, some of the new profiles are defined for the usages under specific cases like 3D, screen content coding and so on. Some new coding features in these profiles may have potentials to explore new architectures for better energy efficiency. The above discussions are some meaningful future works for achieving an energy efficient HEVC decoder system.

Finally, it is possible to extend the motivation of data reuse to other applications as the future works. For example, HEVC encoder consumes more energies than decoder and there are many potentials to reduce energy costs from data reuse. The inter prediction in HEVC encoder, also known as motion estimation, dominates the majority of the total encoding complexity. It usually employs block-matching algorithms by sliding searching window inside the whole searching region. The overlapped searching windows provide good potentials for exploiting energy efficiency VLSI a

# Reference

[1] Cisco, "White paper: Cisco VNI forecast and methodology, 2015-2020," 2016.

[2] Ultra-high-definition television. [Online]. Available: https://en.wikipedia.org/wiki/Ultra-high-definition_television

[3] Vivienne Sze, Madhukar Budagavi and Gary J. Sullivan, "High Efficiency Video Coding (HEVC) Algorithms and Architectures," Springer, 2014.

[4] F. Bossen, B. Bross, K. Suhring and D. Flynn, "HEVC Complexity and Implementation Analysis," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1685-1696, Dec. 2012.

[5] JCT-VC, "High Efficiency Video Coding (HEVC) Defect Report 2," JCTVC-O1003, Nov. 2013.

[6] G. J. Sullivan, J. R. Ohm, W. J. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No. 12, pp. 1649-1668, Dec. 2012.

[7] Moore's law. [Online]. Available: https://en.wikipedia.org/wiki/Moore's_law

[8] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," IEEE In -State Circuits Conference, pp. 10-14, Feb. 2014.

[9] ISSCC 2017 Trends. [Online]. Available: http://isscc.org/doc/2017/ISSCC2017_TechTrends.pdf

[10] JEDEC DDR3 Specification. [Online]. Available: http://ww /JESD79-3E.pdf

[11] Dajiang Zhou, Shihao Wang, Heming Sun, Jianbin Zhou, Jiayi Zhu, Yijin Zhao,

Jinjia Zhou, Shuping Zhang, Shinji Kimura, Takeshi Yoshimura, and Satoshi Goto, “A 4Gpixel/s 8/10b H.265/HEVC video decoder chip for 8K Ultra HD applications,” in Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, USA, pp. 266-267, Feb. 2016.

[12] JCT-VC, "High efficiency video coding (HEVC) test model 13 (HM 13) encoder description," JCTVC-O1002, Nov. 2013.

[13] G. L. Li, C. C. Wang and K. H. Chiang, "An efficient motion vector prediction method for avoiding AMVP data dependency for HEVC," IEEE In
-7366, 2014.

[14] J. Zheng, D. Wu, L. Deng, D. Xie, and W. Gao, “A motion vector predictor architecture for AVS and MPEG-2 HDTV decoder,” Zhuang Y., Yang SQ., Rui Y., He Q. (eds) Advances in Multimedia Information Processing - PCM 2006. PCM 2006. Lecture Notes in Computer Science, Vol.4261, pp.424–431, 2006.

[15] K. Xu and C.S. Choy, “A power-efficient and self-adaptive prediction engine for H.264/AVC decoding,” IEEE Transactions on Very Large Scale Integration Systems, Vol.16, No.3, pp.302–313, Mar. 2008.

[16] H. Yin, D.P. Zhang, X. Wang, and Z. Xia, “An efficient MV prediction VLSI architecture for H.264 video decoder,” In
Image Processing, pp.423–428, July 2008.

[17] K. Yoo, J.H. Lee, and K. Sohn, “VLSI a
IEEE In Image Processing, pp.1412–1415, Oct. 2008.

[18] Y. Tao, G. He, W. He, Q. Wang, J. Ma, and Z. Mao, “Effective multi-standard macroblock prediction VLSI design for reconfigurable multimedia systems,” IEEE In –1490, May 2011.

[19] J. Zhou, D. Zhou, X. He, and S. Goto, “A bandwidth optimized, 64 cycles/MB joint parameter decoder architecture for ultra-high definition H.264/AVC applications,” IEICE transactions on fundamentals of electronics, communications and computer sciences, Vol.93, No.8, pp.1425–1433, 2010.

[20] D. Palomino, F. Sampaio, S. Bampi, A. Susin, and L. Agostini, “FPGA based design for motion vector prediction in H.264/AVC encoders targeting hd1080p resolution,” VIII Southern Conference on Programmable Logic, pp.1–6, Mar. 2012.

[21] M. Tikekar, C. T. Huang, C. Juvekar, V. Sze and A. P. Chandrakasan, "A 249-Mpixel/s HEVC Video-Decoder Chip for 4K Ultra-HD Applications," IEEE Journal of Solid-State Circuits, Vol. 49, No. 1, pp. 61-72, Jan. 2014.

[22] D. Zhou et al., "A 1080p@60fps multi-standard video decoder chip designed for power and cost efficiency in a system perspective," 2009 Symposium on VLSI Circuits, pp. 262-263, 2009.

[23] G.-S. Yu and T. S. Chang, “Optimal data mapping for motion compensation in H.264 video decoding,” IEEE Workshop on Signal Processing Systems, pp. 505–508, Oct. 2007.

[24] Tzu-Der Chuang, Lo-Mei Chang, Tsai-Wei Chiu, Yi-Hau Chen and L. G. Chen, "Bandwidth-efficient cache-based motion compensation architecture with DRAM-friendly data access control," IEEE In

-2012, Apr. 2009.

[25] X. Chen, P. Liu, J. Zhu, D. Zhou, and S. Goto, “Block-pipelining cache for motion compensation in high definition H.264/AVC video decoder,” IEEE In

–1072, May 2009.

[26] J. Zhou, D. Zhou, G. He, and S. Goto, “Cache based motion compensation architecture for quad-HD H.264/AVC video decoder,” IEICE Transactions on

Electronics, Vol. 94, No. 4, pp. 439–447, Apr. 2011.

[27] C.-Y. Tsai, T.-C. Chen, T.-W. Chen, and L.-G. Chen, “Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder,” 48th IEEE In –1202, Aug. 2005.

[28] T.-D. Chuang, P.-K. Tsung, P.-C. Lin, L.-M. Chang, T.-C. Ma, Y.-H. Chen, and L.-G. Chen, “A 59.5mw scalable/multi-view video decoder chip for Quad/3D Full HDTV and video streaming applications,” In -State Circuits Conference, pp. 330–331, Feb. 2010.

[29] V. Sze, D. Finchelstein, M. Sinangil, and A. Chandrakasan, “A 0.7-V 1.8-mW H.264/AVC 720p video decoder,” IEEE Journal of Solid-State Circuits, Vol. 44, No. 11, pp. 2943–2956, Nov. 2009.

[30] D.-Y. Shen and T.-H. Tsai, “A 4x4-block level pipeline and bandwidth optimized motion compensation hardware design for H.264/AVC decoder,” IEEE International Conference on Multimedia and Expo, pp. 1106–1109, Jun. 2009.

[31] C. Xianmin, L. Peilin, Z. Jiayi, and P. Xingguang, “A high performance and low bandwidth multi-standard motion compensation design for HD video decoder,” IEICE Transaction on Electronics, Vol. 93, No. 3, pp. 253–260, Mar. 2010.

[32] J. Zheng, W. Gao, D. Wu, and D. Xie, “A novel VLSI a IEEE Transactions on Consumer Electronics, Vol. 54, No. 2, pp. 687–694, May 2008.

[33] N. Zhou, F. Qiao, and H. Yang, “A hybrid cache architecture with 2D based prefetching scheme for image and video processing,” IEEE In –1096, Apr. 2013.

[34] S. Zuo, M. Wang, and L. Xiao, “A cache hardware design for H.264 encoder,” 2nd

International Conference on Instrumentation and Measurement, Computer, Communication and Control, pp. 922–925, Dec. 2012.

[35] D. Zhou, J. Zhou, X. He, J. Zhu, J. Kong, P. Liu, and S. Goto, “A 530 Mpixels/s 4096x2160@60fps H.264/AVC high profile video decoder chip,” IEEE Journal of Solid-State Circuits, Vol. 46, No. 4, pp. 777–788, Apr. 2011.

[36] Y. Li, Y. Qu, and Y. He, “Memory cache based motion compensation architecture for HDTV H.264/AVC decoder,” IEEE In
–2909, May 2007.

[37] A. Azevedo, B. Zatt, L. Agostini, and S. Bampi, “Mocha: a bi-predictive motion compensation hardware for H.264/AVC decoder targeting HDTV,” IEEE In
–1620, May 2007.

[38] Z. Guo, D. Zhou, and S. Goto, “An optimized MC interpolation architecture for HEVC,” IEEE In
–1120, Mar. 2012.

[39] E. Kalali and I. Hamzaoglu, “A low energy HEVC sub-pixel interpolation hardware,” IEEE In Image Processing, pp. 1218–1222, Oct. 2014.

[40] X. Lian, W. Zhou, Z. Duan, and R. Li, “An efficient interpolation filter VLSI a IEEE China Summit International Conference on Signal and Information Processing, pp. 384–388, July 2014.

[41] C. Machado Diniz, M. Shafique, S. Bampi, and J. Henkel, “High-throughput interpolation hardware architecture with coarse-grained reconfigurable datapaths for HEVC,” 20th IEEE International Conference on Image Processing, pp. 2091–2095, Sep. 2013.

[42] H. Sanghvi, “2D cache architecture for motion compensation in a 4K Ultra-HD

AVC and HEVC video codec system," IEEE International Conference on Consumer Electronics, pp. 189–190, Jan. 2014.

[43] P. Chiang, Y. Ting, H. Chen, S. Jou, I. Chen, H. Fang, and T. Chang, "A QFHD 30fps HEVC decoder design," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 26, No. 4, pp. 725–735, Apr. 2014.

[44] S. Cho and H. Kim, "Hardware implementation of a HEVC decoder," JCTVC-L0096, Jan. 2013.

# Publications

Papers with "○" in the front are involved in this dissertation.

- **Journal Papers (with review)**


○ [1] **Shihao Wang**, Dajiang Zhou, Jianbin Zhou, Takeshi Yoshimura, and Satoshi Goto, "VLSI Implementation of HEVC Motion Compensation with Distance Biased Direct Cache Mapping for 8K UHDTV Applications," IEEE Transactions on Circuit and System Video Technology, Vol. 27, No. 2, pp. 380-393, Feb. 2017.

[2] Jianbin Zhou, Dajiang Zhou, **Shihao Wang**, Shuping Zhang, Takeshi Yoshimura, Satoshi Goto, "A Dual-Clock VLSI Design of H.265 Sample Adaptive Offset Estimation for 8k Ultra-HD TV Encoding," IEEE Transactions on Very Large Scale In -724, Feb. 2017.

[3] Dajiang Zhou, **Shihao Wang**, Heming Sun, Jianbin Zhou, Jiayi Zhu, Yijin Zhao, Jinjia Zhou, Shuping Zhang, Shinji Kimura, Takeshi Yoshimura, and Satoshi Goto, "An 8K H.265/HEVC Video Decoder Chip with a New System Pipeline Design," IEEE Journal of Solid-State Circuits, Vol. 52, No. 1, pp. 113-126, Jan. 2017.

[4] Jianbin Zhou, Dajiang Zhou, **Shihao Wang**, Takeshi Yoshimura, and Satoshi Goto, "High Performance VLSI Architecture of H.265/HEVC In IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E98-A, No. 12, pp. 2519-2527, Dec. 2015.

○ [5] **Shihao Wang**, Dajiang Zhou, Jianbin Zhou, Takeshi Yoshimura, and Satoshi Goto, "Unified Parameter Decoder Architecture for H.265/HEVC Motion

Vector and Boundary Strength Decoding," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E98-A, No. 7, pp. 1356-1365, July 2015.

- **International Conference Papers (with review)**

[1] Kaiyi Yang, **Shihao Wang**, Jianbin Zhou and Takeshi Yoshimura, "Energy-Efficient Scheduling Method with Cross-Loop Model for Resource-Limited CNN Accelerator Designs," IEEE In

[2] _____ -N An Energy-Efficient 1D Chain Architecture for Accelerating Deep Convolutional Neural Networks," Design, Automation and Test in Europe, (DATE), Lausanne, Switzerland, pp.1032-1037, Mar. 2017.

[3] Xushen Han, Dajiang Zhou, **Shihao Wang** and Shinji Kimura, "CNN-MERP: An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks," IEEE International Conference on Computer Design (ICCD), Phoenix, USA, pp. 320-327, Oct. 2016.

[4] Dajiang Zhou, **Shihao Wang**, Heming Sun, Jianbin Zhou, Jiayi Zhu, Yijin Zhao, Jinjia Zhou, Shuping Zhang, Shinji Kimura, Takeshi Yoshimura, and Satoshi Goto, "A 4Gpixel/s 8/10b H.265/HEVC video decoder chip for 8K Ultra HD applications," in Proceedings of IEEE In -State Circuits Conference (ISSCC), San Francisco, USA, pp. 266-267, Feb. 2016.

o [5] **Shihao Wang**, Dajiang Zhou, J. Zhou, Takeshi Yoshimura, and Satoshi Goto, "Unified VLSI architecture of motion vector and boundary strength parameter decoder for 8k UHDTV HEVC decoder," in W. Ooi, C. Snoek, H. Tan, C.-K.

Ho, B. Huet, and C.-W. Ngo, Eds. Advances in Multimedia Information Processing C PCM 2014, Lecture Notes in Computer Science, Kuching, Malaysia, Vol. 8879, pp.74–83, Dec. 2014.

○ [6] **Shihao Wang**, Dajiang Zhou, and Satoshi Goto, “Motion compensation a

ICME), Chengdu, China, pp. 1–6, July 2014.

- **Domestic Conference Paper (without review)**

[1] **Shihao Wang**, Dajiang Zhou, and Satoshi Goto, “Memory Organization in HEVC Motion Compensation for UHDTV Luma and Chroma Parallel”, Niigata, Japan, IEICE General Conference, D-11-62, pp. 69, Mar. 2014.