A Bandwidth Optimized, 64 Cycles/MB Joint Parameter Decoder Architecture for Ultra High Definition H.264/AVC Applications

Jinjia ZHOU†,††, Dajiang ZHOU†, Xun HE†, Nonmembers, and Satoshi GOTO†, Fellow

SUMMARY In this paper, VLSI architecture of a joint parameter decoder is proposed to realize the calculation of motion vector (MV), intra prediction mode (IPM) and boundary strength (BS) for ultra high definition H.264/AVC applications. For this architecture, a 64-cycle-per-MB pipeline with simplified control modes is designed to increase system throughput and reduce hardware cost. Moreover, in order to save memory bandwidth, the data which includes the motion information for the co-located picture and the last decoded line, is pre-processed before being stored to DRAM. A partition based storage format is applied to condense the MB level data, while variable length coding based compression method is utilized to reduce the data size in each partition. Experimental results show our design is capable of real-time 3840 × 2160@60 fps decoding at less than 133 MHz, with 37.2k logic gates. Meanwhile, by applying the proposed scheme, 85–98% bandwidth saving is achieved, compared with storing the original information for every 4 × 4 block to DRAM.

KEY WORDS: motion vector derivation, DRAM bandwidth, ultra high resolution, video decoder, H.264/AVC

1. Introduction

The recent years have witnessed tremendous advances in digital video technology. While 1080HD has already become a standard for TV broadcasting and home entertainment, even higher resolutions, such as the QFHD (3840 × 2160) format, have been targeted by high-end applications of digital cinema and etc. H.264/AVC [1], as a powerful and popular video coding standard, is sure to be a suitable solution for enhancing the throughput of video codec LSI, have been concentrated on reducing the clock cycles required for processing a certain amount of video samples. Assuming that the whole system is working at 133 MHz, the theoretical time budget for real-time processing a QFHD@60 fps sequence should be no more than 67 cycles/MB for each component, which is to be even more critical for higher specifications. On the other hand, with specifications enhanced, data bandwidth will also bottleneck the design of video codec, for the video system usually rely on large external DRAMs for buffering the mass data such as reference frames and co-located information. Owing to the huge bandwidth requirement, power consumption for DRAM traffic can compose a significant portion of the system power, especially for ultra high definition applications.

1.1 High Throughput Requirement

Motion vector (MV) calculation, which restores the current MV from adjacent decoded MVs and MV difference, is one of the most algorithm-irregular components of the H.264/AVC framework, due to the use of variable-blocksize inter prediction and multiple prediction modes. Hence, efficient hardware design of this component can be very difficult, especially for ultra high resolution applications. It is also the most complex part of our parameter decoder architecture.

Most of the previous works on MV calculation architectures have been targeted to standard HD applications. In Yoo et al.’s work [2], the processing time of the motion vector processor vary from 28 cycles/MB to 260 cycles/MB. Another architecture proposed by Yin et al. [3], takes 260 cycles to process one MB for the worst-case, and 160 cycles for average.

Considering these architectures’ overall speed in a pipelined video decoder is restricted by the worst-case performance, it is difficult to apply them to higher specifications. In order to improve the worst-case performance of the MV calculation part, this paper presents designs for reducing the processing time and control complexity. A joint pipeline mechanism is designed to save the processing time. And then, through categorizing the various partition sizes and prediction modes into two groups, the control logic is reduced and the pipeline stall on partition size alteration can be avoided. As a result, a high constant throughput pipeline mechanism is achieved.

Moreover, while most existing works only pay attention to the MV calculation part, in our work, MV calculation, together with the intra prediction mode (IPM) calculation and boundary strength (BS) calculation, are implemented as a joint parameter decoder architecture, based on the following considerations. On one hand, by combining the IPM calculation and MV calculation components, the function of reading and writing the neighboring data can be reused since both of them require data from the adjacent MBs. On the other hand, integrating the calculation of BS and MV helps eliminate the buffer for transmitting the current MVs, and also enables the sharing of neighboring MV fetching operation, because these MVs are required for BS
calculation. In this work, a pipeline architecture which supports the 3 functions is implemented to process each MB in 64 clock cycles, so that real-time QFHD@60 fps sequences can be supported with less than 133 MHz.

1.2 Low Memory Bandwidth Requirement

Since reference read operation for motion compensation (MC) part composes a dominant portion of DRAM traffic, many works [4]–[7] have been discussed on reducing the bandwidth of this part and their contributions are nearing limits. Therefore, the bandwidth portion for parameter decoding becomes considerable and significant. In MV decoding, motion information (MV and reference index) for a whole co-located picture (colPic) and the last decoded line (picture width) is required, and consequently, mass data must be stored and fetched back which leads to large amount of external memory accesses.

Generally, in order to support random accessibility to avoid “useless” memory operation, the co-located information is stored for each 4 × 4-block with fixed data length. For this method, even when the partition size of inter prediction is larger than 4 × 4, the same co-located information is repeatedly stored for each 4 × 4.

We propose to store the co-located information by each partition to eliminate the data overlapping and reduce the memory writing bandwidth, at the cost of giving up the support for random access. Although the extra memory reading traffic is incurred due to the fact that all the stored data should be fetched back, the long latency of DRAM access can be avoided and the bandwidth reduction for writing operation is significant. After that, different low-complexity variable length coding methods are applied to compress the difference value of MVs between the partitions, and the reference index (refIdx), respectively. Finally, a combination method is employed to minimize the total bandwidth for processing the co-located information and line information.

The rest of this paper is organized as follows. The proposed joint pipeline architecture is presented in Sect. 2. Then the bandwidth reduction methods are explained in 3. Sections 5 and 6 give the experimental results and conclusion respectively.

2. Proposed Pipeline Architecture

2.1 Pipeline Analysis

The design of joint parameter decoder architecture is mainly challenged by the application of variable-block-size and multiple-mode inter/intra prediction, which means the computation complexity varies significantly for MBs with different coding features. Taking MV calculation as an example, a non-pipelining implementation (Fig. 1(a)) requires at least three stages performed sequentially to derive the MV for a partition, including memory read (Mem Rd.) stage for fetching reference MVs, calculation (Calc. MV) stage for arithmetic operations, and memory write (Mem Wr.) stage for writing back the results. This type of design is easy to implement with a state machine, whereas it suffers from a relatively low throughput because the calculation hardware is not efficiently used.

Theoretically, by parallelizing arithmetic and memory operations, the processing flow in Fig. 1(a) can be scheduled to be as Fig. 1(b) for eliminating the time for memory

![Pipeline architecture analysis](image-url)
read/write. Meanwhile, the classic data forwarding techniques can help bypass the results from one calculation stage to the next, instead of transferring them through the memory, so as to solve the data dependency problem. However, with various partition sizes and prediction modes applied, each combination of these two factors may require a different cycle count for arithmetic operation, or a different time point to issue the memory operations. Furthermore, the operations need careful scheduling for avoiding resource hazards (e.g., conflicts on use of memory ports) under various conditions, and result forwarding also adds to the control complexity. All these issues result in a very large pipeline controller that may be too difficult and cost ineffective to be implemented, while the situation can be even more complicated if IPM and BS, in addition to MV calculation, are taken into consideration.

To solve these problems, Yoo et al. [2] developed their architecture based on the solution in Fig. 1(c). For this solution, each partition requires a constant cycle count and fixed sequence for memory and arithmetic operations. Therefore the pipeline control can be significantly simplified. However, this means the simplest and the most complicated prediction modes are allocated with the same processing time, which reduces the hardware efficiency. And the worst-case (when one MB consists of 16 partitions) cycle count for each MB is 16λ, where λ is the calculation cycle count for each partition. Although the best-case cycle count is λ when there is only one partition in the MB, this hardly helps the design of a decoding system, which should be planned according to the worst-case, or at least, the average performance of its components.

In this work, we try to realize a more hardware efficient implementation by taking advantage of the high-level features of H.264/AVC standard. For level 3 or higher, which is designed for video specifications higher than or equal to 720×576@25 fps, bi-prediction MV is not allowed for partition sizes smaller than 8×8, and direct prediction is performed in 8×8, instead of 4×4 partitions. This means the MV calculation of 8×4, 4×8 and 4×4 partitions can be less complicated than that of the larger ones, and therefore requires less cycle count. Based on this consideration, we propose a dual-mode pipelining solution as shown in Fig. 1(d). For this solution, two types of calculation operations for 4×4 and 8×8 blocks are organized together. While the processing time for a 8×8 block can still be regarded as λ (λ = 16 in this implementation), each 4×4 block only needs λ/4. As a result, considering each MB can contain at most 4×8×8 blocks or 16×4×4 blocks, the worst-case cycle count for each MB is reduced to 4λ, compared to the 16λ for Fig. 1(c). By using this dual-mode solution, control logic can be a bit more complicated than the single-mode solution, but still significantly simpler than the ideal pipeline in Fig. 1(b). Details of the pipeline design, together with the support for IPM and BS calculation, are discussed in 2.2.

2.2 Pipeline Details

The intra/intra calculation (IC) task, which determines the processing time of each partition, is composed of two operations, MV calculation for inter MB and IPM calculation for intra MB. Between the two operations, the derivation process of MV is more complicated and irregular due to the use of various partition sizes (4×4, 4×8, 8×4, 8×8, 16×8, 8×16, 16×16) and multiple prediction modes (skip, forward, backward, bi-directional, spatial direct and temporal direct). The irregularity results in variable throughput of the whole JPDex architecture, and consequently may bring overhead to the pipelined video decoder. Moreover, hardware implementation for such algorithm-irregular component can be very difficult. However, it is difficult to obtain the fixed processing time for each partition, since the partition size and prediction mode, which determine the computational complexity of the partition, is variable.

In order to overcome the above problem, an MB level constant throughput pipeline mechanism is designed. We propose to simplify the processing patterns according to the following considerations on high-level limits described in H.264 standard [1]. Firstly, when the level is higher than 2.2, direct\(\lambda\times8\times8\)Inference\_Flag is set equal to 1, which means derivation process of MV on the B\_Skip, B\_Direct\_16, and B\_Direct\_8\times8\)modes can be processed based on 8×8 block. Secondly, on levels higher than 3, bi-prediction mode is not allowed in partitions smaller than 8×8. Based on the above specifications, the small block sizes, which should be processed more times in one MB, consume less calculation time, because only the single-directional prediction is allowed. Meanwhile, the complex B skip, B direct mode, and bi-prediction are based on large partition sizes. Thus, we classify the various partition sizes and prediction modes into two groups, 4×4-block group and 8×8-

<table>
<thead>
<tr>
<th>Table 1 Group categorization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
</tr>
<tr>
<td>Intra Prediction Mode</td>
</tr>
<tr>
<td>Partition Patterns for inter MB*</td>
</tr>
</tbody>
</table>

Partition patterns for inter MB* is represented in this way: *Partition\_PredMode\_Partition\_Width\_Partition\_Height
block group as shown in Table 1. For the $4 \times 4$-block group, it contains partitions smaller than $8 \times 8$ in regular mode. The others including the $P_{\text{skip}}$ mode, $B_{\text{Skip}}$, $B_{\text{Direct}}16 \times 16$, $B_{\text{Direct}}8 \times 8$, and the partitions with support to bi-prediction in regular mode are classified to $8 \times 8$-block group. Although the calculation for $8 \times 8$-block group is more complex than the $4 \times 4$-block one, it has four times processing time budget as the $4 \times 4$-block group. In order to share the same processing schedule with MV calculation part, the IPM calculation part is classified to the same two groups as the MV calculation part, for which the $4 \times 4$-block based intra processing is categorized to $4 \times 4$-block group while $8 \times 8$ and $16 \times 16$ are classified to $8 \times 8$-block group.

The pipeline flow is derived based on the two groups. For the $4 \times 4$-block group, all the members in this group are divided into several $4 \times 4$ blocks. For each $4 \times 4$ block, the processing time mainly depends on memory reading and writing part since the calculation parts (Inter/Intra Calc and BS Calc) are not complicated. In the memory related part, the first line of blocks in one MB require the longest time for fetching back the adjacent information because the information of upper blocks is stored in the external memory, as shown in Fig. 2. For the other blocks in this MB, the adjacent data can be got instantly, since the neighboring information from either the left MB or the decoded blocks in current MB, is stored in registers. Then, one cycle is needed to write back the information of the previously decoded block. Therefore, the time budget for memory writing and reading is four cycles (three cycles for reading the adjacent data for current block and one cycles for writing back the information of the previous one). Moreover, since the calculation is not complex for the $4 \times 4$-block group, the MV calculation for inter MB, IPM calculation for intra one, and BS calculation can all be finished within four cycles.

In the $8 \times 8$-block group, the members are segmented into several $8 \times 8$ blocks. The similar pipeline structure, which requires sixteen cycles for each $8 \times 8$ block, is employed to avoid the stall between the boundary of different groups.

The pipeline mechanism, as shown in Fig. 3, brings the following benefits. Firstly, by combing the MV and IPM calculation, the \textit{Mem wr. & re.} part which prepares the data of neighboring MBs, can be reused for saving the area cost. Secondly, MV calculation and IPM calculation can share the same pipeline to avoid hazards on MB type alteration. Finally, since the three operations of reading/writing the neighboring information for the next block, calculating the MV or IPM for the current one, and calculating BS for the previous one, can be processed at the same time, the processing time of the JPDec can be optimized.

As a result, no matter how an MB is partitioned (sixteen
units of $4 \times 4$-block group, four units of $8 \times 8$-block group, or combination of $4 \times 4$-block group and $8 \times 8$-block group), and no matter what type (inter or intra) an MB is, a constant-throughput of 64 cycles/MB can be obtained.

3. DRAM Bandwidth Reduction Strategy

In order to reduce the DRAM bandwidth, we propose a partition based storage format and variable length coding based compression method to condense the co-located information (co-located MV and refIdx) for a whole colPic and the line information (MV, refIdx and IPM of the last decoded line). The detailed data compression design is presented in 3.1 and 3.2. Moreover, in 3.3, the scheme on reusing the overlapped data of co-located and line information is introduced. Finally, 3.4 summaries the memory bandwidth of each part in the JPDec.

3.1 Partition Based Storage Format

Generally, in order to support random access of the co-located information for avoiding extra memory traffic when only a certain part of data is required, the co-located information is stored for each $4 \times 4$-block with fixed data length. For this method, even when the size of partition, which is the basic processing unit of inter prediction, is larger than $4 \times 4$, the same co-located information is repeatedly stored for each $4 \times 4$. Consequently, the large memory size causes large DRAM bandwidth. In our work, one partition is utilized as a basic storage unit to eliminate the data overlapping and reduce the memory writing bandwidth. However, since the partitions are variable, the storage size for each MB becomes unfixed which brings data dependency between MBs. Hence, random access can no longer be supported, so that the stored data for a whole colPic should be fetched back when only part of data is required. Although the extra memory reading traffic is incurred, the long latency of DRAM access can be avoided. Moreover, for higher specification applications which requires larger memory bandwidth, the partition based storage format is more efficient because of the following reasons. On one hand, as defined in the H.264 standard [1], on levels higher than 3.0, the maximum number of motion vectors per two consecutive MBs should be less than 16. Accordingly, the average number of partitions for each MB is smaller than 8, and consequently result in more than half memory writing reduction, compared to the original $4 \times 4$-based storage format. On the other hand, B_skip and B_direct modes which requires the co-located information, occupies high proportion in usual, so the reading bandwidth increase can not be significant.

Figure 4 shows the detailed storage format. When the partition size is larger than $8 \times 8$, 2 extra bits are required to represent the partition size for each MB. Otherwise, 10 bits are needed. And then, the co-located information of each partition is stored in order. Since the bit cost of the co-located information for one MB is from 29 to 464 bits (29 bits for one partition, and the maximum partition number for one MB is 16), the bits denoting partition sizes (3–7% of the total size) are likely to be negligible. Another extra one bit is needed to denote the intra MB. When the current MB is intra, this intra MB flag is set to one, the partition size is set to $16 \times 16$, and the followed information is not required, so that this extra bit saves 29 bits (90% storage size) for the intra MB.

3.2 Variable Length Coding Based Compression

The co-located information is composed of two parts includ-
ing co-located MV and co-located refIdx. According to the different properties of the two parts, two variable length coding (VLC) methods are applied to encode them respectively.

For reducing the spatial redundancy, when storing the motion information of the current frame for subsequent use as co-located picture, the difference of MVs (DMV: in order to be different with the syntax element MVD in the standard) between the current and the last decoded partitions is calculated when these two share the same refIdx. Considering the probability distribution of DMV values approximates to geometric distribution, Exp-Golomb coding is proceeded to these values so as to express the co-located MVs with less bits. If refIdx is different between the current and the previous partitions, the DMV value becomes large which deteriorates the efficiency of Exp-Golomb coding. Therefore for this case, the original MV values, instead of DMV, are stored. When the co-located information is fetched, the MVs are restored in the identical sequence in which they have been stored. As a result, the restoring component is capable of identifying whether DMV or original MV has been stored, based on whether or not the refIdx of the current partition equals to that of the last restored partition. This means no extra flags are required to indicate which one is used.

According to the probability distribution of co-located refIdx, for which most of the values are close to zero, the unary coding is selected to encode this part. However, since the unary coding can only represent the natural numbers, redefined unary coding table shown in Fig. 4 is designed to suit the coding for co-located refIdx.

3.3 Co-located and Line Data Combination

The partition based and VLC compression method which are applied to condense the data of co-located information, can also be utilized to compress the line information shown in Fig. 2. Although the potential for usable dependency between the partitions in line information buffer is lower, there is no extra memory traffic for not supporting random access since all the data in line information buffer should be fetched back. As a result, the memory traffic for processing the line information buffer also can be reduced.

Moreover, since the line information is included in the co-located information buffer in the P frame, some data is stored two times (both in the co-located information buffer and in the line information buffer). The straightforward way to solve the problem of data overlapping, is eliminating the line information buffer for P frame and fetching back the line information from the co-location information buffer. However, with this method, all the co-located information should be fetched back due to the data dependency between the partitions, but only a part of it is required. A lot of useless data fetching results in increasing the memory traffic. Therefore, the following organization is designed to reuse the overlapped data, and avoid extra memory access overhead. Firstly, as shown in Fig. 2, the co-located information buffer and line information buffer are combined together to store the data of the whole colPic. If the data of one partition is stored in line information buffer, it will not be stored in the co-located information buffer again. Hence, every time for fetching back the co-located information, the line information buffer is read first, and then, the co-located information buffer is accessed except for the mb_type is 16 × 16 or 8 × 16 for which all the data is buffered in the line information buffer. Furthermore, to prevent from affecting the compression ratio for lowering the data dependency potential, the partitions in co-located information buffer can refer to the previous one even if it is in line information buffer. Finally, the partitions in the line information buffer can only refer to the previous one in the same buffer, and as a result, the data dependency potential within the line information is lowered. However, on the sacrifice of the compression ratio, the extra memory traffic caused by mass useless data being fetched back can be removed.

3.4 DRAM Bandwidth Summary

Figure 5 summaries the memory bandwidth. By using partition based storage format, the bandwidth is reduced by 78.7%, and then, it is further reduced by 76.9% depending on the variable length coding based compression method. Considering the bandwidth overhead for accessing the line information buffer when reading the co-located information, total bandwidth decreases by 87.5% though the combination method.

4. Architecture Summary

As shown in Fig. 6, the pipelined calculation part and data
compression/decompression parts are connected by buffers. The calculation part can read the required data from FIFO\(_1\), and write back the data to FIFO\(_0\) for being stored to DRAM. Meanwhile FIFO\(_0\) outputs the data to compression module, and FIFO\(_1\) gets back the decompressed data for the calculation part. As a result, the data compression and decompression behaviors are transparent to the pipelined calculation part, for which the processing time of the whole system will not be restricted from the compression scheme.

5. Implementation Results

The proposed architecture is implemented in Verilog HDL on RTL level, and then synthesized with Synopsys Design-Compiler by using TSMC 90 G standard cell library. Under a timing constraint of 200 MHz, synthesis result shows a logic gate count of 37.2k. This design is verified both independently in a test environment with inputs given as software generated data, and in a whole QFHD video decoder architecture.

A comparison between this architecture and state-of-the-art works is shown in Table 2. The number of clock cycles required for processing one MB is reduced by 75% from the previous works. There are several techniques utilized to reduce the number of cycles to process one MB. As described in 2, pipelining can speedup the whole system. Through categorizing the various partition sizes and prediction modes into two groups according to the high-level limits specified in H.264 standard [1], a constant high throughput architecture is obtained.

Table 3 shows the bandwidth test result on various sequences. Firstly, the bandwidth is reduced by applying the partition based storage formats. Then, the multiple VLC including the Exp-Golomb coding and unary coding, is utilized in a test environment with inputs given as software generated data, and in a whole QFHD video decoder architecture.

### Table 2

Comparison between this work and state-of-the-art architectures.

<table>
<thead>
<tr>
<th>Supported Function</th>
<th>Logic Gate Count</th>
<th>Technology</th>
<th>Processing Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV calculation</td>
<td>41.1 k</td>
<td>65 nm</td>
<td>126 MHz for 1920×1080@60 fps</td>
</tr>
<tr>
<td>MV calculation</td>
<td>52 k</td>
<td>0.18 μm</td>
<td>505 MHz for 3840×2160@60 fps</td>
</tr>
<tr>
<td>Intra prediction mode derivat</td>
<td>37.2 k(^*)</td>
<td>90 nm</td>
<td>31 MHz for 1920×1080@60 fps</td>
</tr>
</tbody>
</table>

37.2 k\(^*\): Consisting of 3.1 k for compressing the DRAM bandwidth, 4.7 for decompressing, and 29.1k for the calculation part.

### Table 3

Bandwidth comparison on various sequences.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>QP</th>
<th>Frame Format</th>
<th>Original(^1) (M Bits)</th>
<th>Compressed (M Bits)</th>
<th>Par. 2) VLC 3) Comb. 4)</th>
<th>Par. 1) ABW VLC 3) Comb. 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluesky</td>
<td>24</td>
<td>IBBP</td>
<td>55.32</td>
<td>15.54 4.57 1.25</td>
<td>–91.91% –91.75% –92.31%</td>
<td></td>
</tr>
<tr>
<td>(1080p, 10 frames)</td>
<td>24</td>
<td>IBBP</td>
<td>52.11</td>
<td>7.36 2.23 1.56</td>
<td>–85.87% –95.72% –97.00%</td>
<td></td>
</tr>
<tr>
<td>(search range: 64)</td>
<td>32</td>
<td>IBBP</td>
<td>52.11</td>
<td>14.20 3.74 3.50</td>
<td>–75.20% –93.46% –93.89%</td>
<td></td>
</tr>
<tr>
<td>Pedestrian</td>
<td>24</td>
<td>IBBP</td>
<td>50.50</td>
<td>10.59 4.17 3.90</td>
<td>–97.98% –96.76% –97.81%</td>
<td></td>
</tr>
<tr>
<td>(1080p, 10 frames)</td>
<td>24</td>
<td>IBBP</td>
<td>52.11</td>
<td>7.30 3.52 2.47</td>
<td>–86.00% –93.24% –95.26%</td>
<td></td>
</tr>
<tr>
<td>(search range: 64)</td>
<td>32</td>
<td>IBBP</td>
<td>52.49</td>
<td>10.31 3.43 3.21</td>
<td>–80.36% –93.47% –93.88%</td>
<td></td>
</tr>
<tr>
<td>Station</td>
<td>24</td>
<td>IBBP</td>
<td>53.37</td>
<td>12.78 4.17 3.94</td>
<td>–97.95% –91.99% –92.62%</td>
<td></td>
</tr>
<tr>
<td>(1080p, 10 frames)</td>
<td>24</td>
<td>IBBP</td>
<td>52.11</td>
<td>7.31 2.33 1.57</td>
<td>–85.97% –95.53% –96.99%</td>
<td></td>
</tr>
<tr>
<td>(search range: 64)</td>
<td>32</td>
<td>IBBP</td>
<td>55.19</td>
<td>12.41 3.79 3.51</td>
<td>–77.51% –93.13% –93.64%</td>
<td></td>
</tr>
<tr>
<td>(RDO turned off)</td>
<td>32</td>
<td>IBBP</td>
<td>52.11</td>
<td>6.66 2.05 1.37</td>
<td>–87.22% –96.07% –97.38%</td>
<td></td>
</tr>
<tr>
<td>InToTree</td>
<td>24</td>
<td>IBBP</td>
<td>119.15</td>
<td>69.29 32.69 30.19</td>
<td>–65.16% –83.59% –84.84%</td>
<td></td>
</tr>
<tr>
<td>(3840×2640, 10 frames)</td>
<td>24</td>
<td>IBBP</td>
<td>206.90</td>
<td>61.00 33.80 25.84</td>
<td>–70.52% –83.66% –87.51%</td>
<td></td>
</tr>
<tr>
<td>(search range: 128)</td>
<td>32</td>
<td>IBBP</td>
<td>232.61</td>
<td>57.31 12.68 11.92</td>
<td>–75.36% –94.55% –94.88%</td>
<td></td>
</tr>
<tr>
<td>(RDO turned off)</td>
<td>32</td>
<td>IBBP</td>
<td>206.90</td>
<td>25.00 5.85 3.91</td>
<td>–87.93% –97.17% –98.11%</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) The original uncompressed data stored by 4×4 blocks.

\(^2\) The partition based storage format is utilized to condense the data.

\(^3\) Multiple Variable Length Coding is added to compress the partition based stored data.

\(^4\) Co-located and Line Data combined storage method is employed to the partition based re-encoded data.
Table 4  Gate count summary.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Gate count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory wr. &amp; re.</td>
<td>3.8 k</td>
</tr>
<tr>
<td>Inter-intra Calculation</td>
<td>18.3 k</td>
</tr>
<tr>
<td>BS Calculation</td>
<td>1.8 k</td>
</tr>
<tr>
<td>Register Control</td>
<td>4.6 k</td>
</tr>
<tr>
<td>Compression</td>
<td>3.1 k</td>
</tr>
<tr>
<td>Decompression</td>
<td>4.7 k</td>
</tr>
<tr>
<td>Total</td>
<td>37.2 k</td>
</tr>
</tbody>
</table>

lized to further condense the data stored by partition. By combing the compressed co-located and line information buffer, the DRAM bandwidth can be finally reduced by 85–98%.

Moreover, the total gate count of proposed architecture is also the smallest one. The detail gate count distribution is summarized in Table 4. The area cost for compressing and decompressing the DRAM access which reduces 85–98% memory bandwidth, occupies only 20% of the total gate count due to the selected Exp-Golomb code and unary code are suitable and simple.

Finally, another merit of this architecture is that the MV calculation, intra prediction mode derivation and BS computation are combined into a single architecture, for which the operation of fetching the neighboring information can be shared between and MV and IPM calculation components, and the MV buffer between the MV and BS calculation components is eliminated.

6. Conclusions

In this paper, we propose a joint parameter decoder architecture, which integrates the calculation functions of MV, IPM and BS for the H.264/AVC standard. To achieve an efficient design, a 64-cycle-per-MB pipeline with simplified control modes is designed to integrate the three functions, so that QFHD@60 fps sequences can be processed with less than 133 MHz. Moreover, considerable memory bandwidth reduction is achieved, by applying efficient partition based storage formation, multiple variable length coding, and combined storage strategy. When synthesized with TSMC 90 nm process, the architecture costs a logic gate count of 37.2 k. Compared with the state-of-the-art works, our design achieves at least 75% clock cycle reduction. Meanwhile, 85–98% bandwidth saving is achieved, compared with storing the original information for every $4 \times 4$ blocks to DRAM.

Acknowledgment

This research was supported by “Ambient SoC Global COE Program of Waseda University” of the Ministry of Education, Culture, Sports, Science and Technology, Japan, and by the CREST project of Japan Science and Technology Agency.

References


Jinjia Zhou received B.E. degree from the Department of Electronic Engineering of Shanghai Jiao Tong University, China, in 2007. She is currently pursing an M.E. degree at the Graduate School of Information, Production and Systems of Waseda University. Her interest is in VLSI implementations for video coding technologies.

Dajiang Zhou received B.E. and M.E. degrees from the Department of Electronic Engineering of Shanghai Jiao Tong University, China, in 2005 and 2008, respectively. Since April 2008, he has been working toward a Ph.D. degree at the Graduate School of Information, Production and Systems of Waseda University, Japan. Since April 2009, he has been a research fellow of Japan Society of the Promotion of Science (JSPS). Currently, his main interest is in algorithms and VLSI architectures for multimedia and communication technologies.
Xun He received the B.S. degree and M.E. degree in Electronics and Information from Graduate School of Electronic, Information and Electrical Engineering of Shanghai Jiao Tong University. He is currently a Ph.D. candidate in Graduate School of Information, Production and Systems, Waseda University, Japan. His research interests include video coding technology, low power multicore processor architecture.

Satoshi Goto received B.E. and M.E. degrees in Electronics and Communication Engineering from Waseda University in 1968 and 1970, respectively. He received the Doctor of Engineering degree from the same university in 1981. He joined NEC Laboratories in 1970 where he worked on LSI design, multimedia systems and software as GM and VP. Since 2003, he has been a professor at the Graduate School of Information, Production and Systems of Waseda University. Currently, his main interest is in VLSI design methodologies for multimedia and mobile applications. He has published 7 books, 38 journal papers and 67 international conference papers. He served as GC of ICCAD and ASPDAC and was a board member of IEEE CAS society. He is IEEE Fellow and Member of Academy Engineering Society of Japan.