WAITING CYCLE ANALYSIS ON H.264 DECODER RUN IN PAC DUO PLATFORM

Wen-Chien Su, Jen-Kuei Yang, Kuei-Chun Liu, Shau-Yin Tseng, Wen-Shan Wang

SoC Technology Center, Industrial Technology Research Institute, Hsinchu, Taiwan 31040, ROC

ABSTRACT

Two approaches for parallelization of H.264 decoder, data partition and function partition, are realized on a PAC Duo platform, which contains two Parallel Architecture Core Digital Signal Processors (PACDSP’s). Eight baseline CIF sequences are decoded and their execution cycles and waiting cycles are examined. There are three roots hindering the performance of dual-core decoders: inter-core synchronization, resource contention, and cache miss. Through the waiting cycle analysis, the major reasons causing the degradation of dual core H.246 decoders are found. The inter core synchronization and resource contention principally slow down the execution speed of the dual core with function partition and dual core data partition, respectively. The precious experience and analysis will help the software and hardware designers explore the mechanisms to improve performance of the multi core scenarios.

Index Terms—data partition, function partition, H.264 decoder, PACDSP, PAC Duo.

1. INTRODUCTION

As the application of computing device is rapidly developed, the requirement of its computational capability is raised as well. In order to support the high computational requirements, multi-core architecture is a good solution to improve computational capacity. However, the multi-core also causes the inter-core synchronization issue, which degrades the performance sharply.

In this paper, we analyze the roots causing the waiting time during executing H.264 decoder software by decoding eight baseline 4:2:0 CIF H.264 sequences with three scenarios: single PACDSP, dual PACDSP with function partition, and dual PACDSP with data partition, which are also denoted as scenarios I, II, and III, respectively. The waiting cycle analysis will provide valuable strategies to reduce the waiting time, especially for the dual PACDSP scenarios, and will become the critical reference for the future design.

The waiting time includes stalling time, when PACDSP idles, and polling time, when PACDSP polls some events happen. There are three main sources causing the degradation of the dual PACDSP scenarios in performance: inter core synchronization, resource contention, and cache miss. All of them will cause the PACDSP stalls but the first two also make the software wastes time on polling. Of course, the resource contention and cache missing also happen to the single PACDSP scenario.

The analysis indicates that the waiting time caused by the cache miss is the least significant one. However, the inter core synchronization and resource contention contribute the biggest parts of waiting cycles for scenarios II and III, respectively. In average, the waiting time caused by the inter core synchronization for scenario II takes about 23% and 38% of the total execution time of their corresponding PACDSP’s, respectively. Meanwhile, the waiting time caused by resource contention for scenario III takes about 13% of the total execution time of each PACDSP.

The rest of this paper is organized as follows. Section 2 states the background of H.264 and partitions. Sections 3 and 4 briefly describe the PAC Duo platform and the H.264 decoder flow, respectively. In Section 5, the detail of two partitions is illustrated. Section 6 provides the experimental results and the conclusion is summarized in Section 7.

2. BACKGROUND

Video compression technologies are primarily based in the fact that the temporal and spatial redundancies among a video sequence can be exploited to reduce the number of bits. The H.264 [1], one of the newest video compression standards, exploits the redundancies to deeper extent than prior standards.

Compared with MPEG-4 Profile, up to 50% bit rate reduction is achieved at the cost of more than four times of computational complexity [2]. Therefore, hardware or software acceleration, especially parallel structure, is necessary for real time application.

However, the modern multimedia application requires very higher resolution video, such as 1280×720 pixels. The single core solution consumes too much power for the high-resolution video because it must run at very high frequency. As a result, multi-core processors are more energy effective for this application, because it can achieve the same performance as a single core processor at a low frequency by distributing the load amount the processing cores. [3]

Generally, there are two approaches for multi-core parallelization: function partition and data partition. Each core in function partition has different functions to handle
the data in the same frame while that in data partition takes care of all functions for decoding the data in different frames. Due to the characteristic of H.264 coding, there are dependences among neighboring macroblocks [4]. It will result in large synchronization overhead and more complication of flow control among cores.

3. PAC DUP PLATFORM ARCHITECTURE

As shown in Fig. 1, the PAC Duo is a heterogeneous multi-core SoC, which consists of one ARM926EJ, the main processor and denoted as MPU, and two PACDSP’s, which are developed by SoC Technology Center, Industrial Technology Research Institute (STC/ITRI), Taiwan, as the digital signal processors. The PACDSP is a 32-bit fixed-point digital signal processor (DSP) with 5-way VLIW pipeline with 32 KB of instruction memory and 64 KB of local data memory.

Before PACDSP’s start processing data, the ARM926EJ must bring the bitstream to be processed to the DDR2, download the execution codes to the instruction memory of PACDSP’s, and then trigger the PACDSP’s to work.

On PAC Duo platform, the Enhanced Multimedia Direct Memory Access (EMDMA) is designed for multimedia application. The EMDMA provides many features as follows: (1) 2D/3D data movement, (2) multiple reference blocks movement, (3) non-alignment data movement, (4) boundary data padding, and (5) linking-list command. With these functionalities, user can utilize it moving data between DDR2 and PACDSP local memory efficiently.

4. H.264 DECODING ALGORITHM

The decoding flow of H.264 can be divided as four main parts: ED (Entropy Decoder and Reorder), IQ/IT (Inverse Transformation/Inverse Quantization), PPC (Intra/Inter Prediction that include prediction and compensation), and DF (Deblocking Filter), as shown in Fig. 2.

The decoder receives a compressed bit-stream from the Network Abstract Layer (NAL). There are two types of data decoded from bit-stream: parameter information, e.g., macroblock (MB) type, prediction mode, QP, reference index, and differential motion vector etc., and the quantized residual data. In the decoding of Context-Adaptive Variable Length Coding (CAVLC), the residual data are decoded and reordered to produce the quantized coefficients X. The block residual data is inverse quantized and transformed to give D’.

From the parameter information, we know the MB type is. If the MB type is intra, the decoder creates prediction P by the information of the previous reconstructed MB (upper and left MB’s) on the current frame F’n. Otherwise, it is inter MB type and the decoder must refer to previous reference frame F’n-1 to create the predication P by motion compensation. Add P to D’n to create uF’n. Finally, uF’n is filtered to produce the reconstructed MB.

5. PARALLEL IMPLEMENTATIONS

5.1 Function Partition Implementation

Firstly, we analyze the ratio of computation for each module of H.264 decoding on single PACDSP as shown in Fig. 3. Obviously, the DF occupies almost half of pie chart in Fig. 3. Therefore, we assign three modules, ED, IQ/IT, and PPC, to PACDSP0 and DF to PACDSP1. The circled numbers in Fig. 3 also indicate the order of the data movements.

In the local data memory of each PACDSP, four data buffers (SD0~SD3) are allocated to store the intermediate or reconstructed data for four microblocks. The data flow of H.264 decoding is shown in Fig. 4. Four data buffers are designed to lower the loading unbalance slightly. In addition to the four data buffers, there is SD count for each PACDSP, which increases its SD count by one after processing one macroblock. The PACDSP0 will check both SD counters to see if any of its SD buffer is free before it saves its processed data and the PACDSP1 will examine both counters to make sure there is at least one SD buffer in PACDSP0 is filled before it accesses the data.

Fig. 3. Computation profile of H.264 decoding on single PACDSP
5.2 Data partition implementation

Unlike algorithm in [7], which divides data based on macroblock, we proposed a frame-based data partition due to only two cores in PAC Duo. ARM partitions the source H.264 bitstream into odd-frame and even-frame bitstreams. The first PACDSP, PACDSP0, performs the odd-frame decoding and the second PACDSP, PACDSP1, executes the even-frame decoding, as shown in Fig. 5.

In general, the H.264 bitstream contains the data of I-frame and P-frame in the same video sequences. This means that there are data dependency between odd-frame and even-frame. Therefore, one PACDSP might need to wait another PACDSP to finish the reference macroblock.

Before a PACDSP decodes an inter macroblock, it also decodes the motion vector and must access the MB count, which records how many macroblocks are decoded, of another PACDSP. From the motion vector and the MB count of another PACDSP, a PACDSP knows if its reference macroblock is ready or not. Fig. 6 illustrates the reference macroblock in frame N-1 is ready for the current macroblock in frame N while frame N-1 is being decoded.

6. EXPERIMENTAL RESULTS

6.1 Experimental Environment and results

We evaluate our proposed parallelizing methods for H.264 decoder on PAC Duo platform, which contains two PACDSP’s running at 312 MHz and one ARM running at 312 MHz. The decoder is running on two PACDSP’s; ARM performs the initialization of DSP and the flow control of display. There are eight video sequences, as shown in Table 1, are used as the test input streams. All of these video sequences have 100 frames at a resolution of 4:2:0 CIF and contain one I frame for every 29 P frames.

Table I shows the performance measurements of H.264 decoding for single and dual core scenarios. The result shows that the function partition and data partition for dual core scenarios speed up 32% and 64%, respectively, compared with the H.264 decoding of single PACDSP.

<table>
<thead>
<tr>
<th>Bit-stream (100 frames, 129P)</th>
<th>Scenario I (fps)</th>
<th>Scenario II (fps)</th>
<th>Scenario III (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coastguard</td>
<td>33.3</td>
<td>44.9</td>
<td>54.9</td>
</tr>
<tr>
<td>Container</td>
<td>90.7</td>
<td>119.1</td>
<td>151.5</td>
</tr>
<tr>
<td>Foreman</td>
<td>39.7</td>
<td>55.1</td>
<td>68.4</td>
</tr>
<tr>
<td>Highway</td>
<td>62.5</td>
<td>81.7</td>
<td>101.7</td>
</tr>
<tr>
<td>Mother_daughter</td>
<td>78.5</td>
<td>102.8</td>
<td>126.8</td>
</tr>
<tr>
<td>News</td>
<td>96.4</td>
<td>129.1</td>
<td>159.4</td>
</tr>
<tr>
<td>Silent</td>
<td>84.1</td>
<td>109.8</td>
<td>131.3</td>
</tr>
<tr>
<td>Stefan</td>
<td>37.1</td>
<td>49.5</td>
<td>60.5</td>
</tr>
<tr>
<td>Average</td>
<td>65.3</td>
<td>86.5</td>
<td>107.4</td>
</tr>
<tr>
<td>Speedup</td>
<td>-</td>
<td>1.32</td>
<td>1.64</td>
</tr>
</tbody>
</table>

6.2 Performance analysis and discussion

The waiting time includes stalling time, when PACDSP idles, and polling time, when PACDSP polls some events happen. There are three main roots make the degradation of the decoding performance: inter core synchronization, resource contention, and cache miss. The first one happens only in multi core scenarios while the other two appear in single core and multi core scenarios. Fig.7 illustrates the polling cycles and stalling cycles for all three scenarios. The cycle counts for inter core synchronization and resource contention in Fig. 7 are polling cycles only; they contribute stalling cycles as well but those cycle counts are minor and covered in the total stalling bars. The cycle counts for cache miss in Fig. 7 are stall cycles and they take the major part of the total stalling cycles.

Fig. 7 points out the total stalling cycle ratios for the function partition and data partition over the single core scenario are 1.56 and 1.37, correspondingly, and polling cycles for function partition is much larger than that for data partition. The detailed analysis for each is described in the
tion. The detailed analysis for each is described in the following subsections.

Fig. 7. Waiting cycle comparisons

6.2.1 Inter core synchronization

In multi-core scenarios, it is very common that one core has to wait until its incoming data is completed by another core or its outgoing data is consumed by another core. For the function partition, one PACDSP almost has to wait the data completion or consumption of every macroblock; for the data partition, one PACDSP waits the data completion for the inter macroblocks only. Table II shows the percentage of waiting cycles caused by the inter-core synchronization over the total decoding cycles.

<table>
<thead>
<tr>
<th></th>
<th>PACDSP0</th>
<th>PACDSP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Partition</td>
<td>23%</td>
<td>38%</td>
</tr>
<tr>
<td>Data Partition</td>
<td>3%</td>
<td>5%</td>
</tr>
</tbody>
</table>

6.2.2 Resource contention

In PAC Duo system, only one EMDMA is designed for the movement of reference data and bitstream used for the motion compensation and entropy decoding, respectively. In data partition approach, two PACDSP’s may need EMDMA to move reference data and bitstream at the same time. This will cause that one PACDSP needs to wait for the release of EMDMA resource occupied by another PACDSP. However, this case does not happen often for the function partition.

Fig. 7 shows the polling cycle count for EMDMA ready is 1.07 and 1.37 times, for function partition and data partition, respectively, as much as that for the single PACDSP scenario.

6.2.3 Cache miss

The code size of H.264 decoder is 50 KB, which is larger than the cache size of PACDSP and will cause cache miss during H.264 decoding. In general, the larger code sizes, the higher probability of cache miss. For scenario II, the code of H.264 decoder is divided into two pieces of code, one is 42 KB and another is 10 KB running in separated PACDSP’s. Consequently, cache miss count of the scenario II is less than that of scenario I.

Theoretically, the cache miss count of scenario III should be same as that of scenario I because their code sizes are identical. However, the waiting clocks caused by cache missing for scenario III are 27% higher than that of scenario I. The main reason is the cache miss may extend the stalling time when cache miss happens in two PACDSP’s simultaneously because there is only one DDR2 saving the software for both PACDSP’s.

7. CONCLUSIONS

Although the inter core synchronization and resource contention will contribute the stalling time; however, Fig. 7 shows that the cache miss produces most of the stalling time. The inter core synchronization bars in Fig. 7 explains why scenario III is faster than scenario II as the speedup data shown in Table I.

The analyses in previous section also indicates that the main source of waiting cycles for the function partition is the inter core synchronization while that for the data partition is the resource contention. The data partition also has to face the cache miss issue, which is somehow related to the resource contention.

According to the discussion above, our future works will focus on the EMDMA and DDR2 module redesign. The new EMDMA should be able to provide more channels and synchronization mechanism to solve the resource contention issue for data partition. Similarly, the banking DDR2 module should allow the PACDSP’s access the code or data without contention. We will also develop dynamic load balance approach to improve performance for function partition.

REFERENCES