DSP IMPLEMENTATION OF DEBLOCKING FILTER FOR AVS

Zhigang Yang[1], Wen Gao[1,2], Yan Liu[1], and Debin Zhao[1]

[1]Department of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
[2]Institute of Computing Technology, Chinese Academy of Science, Beijing 100080, China
{zgyang, wgao, liuyan, dbzhao}@jdl.ac.cn

ABSTRACT

The in-loop deblocking filter contains highly adaptive processing on both sample level and block edge level, which inevitably appears in the loop kernel of the algorithm. Therefore it is a challenge for parallel processing on a digital signal processor (DSP) platform. In this paper, pipelined DSP solutions to the in-loop deblocking filter in AVS1-P2 are presented. First, the whole filter process is divided into six sub-processes, so that the global filter structure can be improved to achieve regular processing flow. Then software pipelines are designed for these sub-processes, with elaborately allocating functional units and carefully choosing enhanced assembly instructions based on the DSP platform. The simulated results show that this efficient implementation can easily support real-time filter processing for high resolution videos.

Index Terms— Pipelines, digital signal processors, deblocking filter, AVS

1. INTRODUCTION

AVS standard is the latest video/audio coding standard established by China Audio Video Coding Standard Working Group [1]. AVS1-P2 video coding standard [2], as one of the most important parts of AVS standard, mainly targets to high resolution, high bit rate coding applications, such as broadcasting. Compared with another popular video coding standards H.264 [3], AVS1-P2 can achieve similar performance while with lower complexity. The framework of AVS1-P2 is also block transform/block prediction hybrid based scheme, which can introduce blocking artifacts [4], especially in low bit rate or highly compressed video environment. So AVS1-P2 adopts an in-loop deblocking filter to reduce the artifact and to improve the objective PSNR as well.

The in-loop deblocking filter can account for nearly 30% of the decoder complexity [5]. One reason for its high complexity is that lots of conditional branches inevitably appear in the most inner loops of the filter algorithm. Another reason is the wide and irregular data accesses. Almost every sample in a picture need be loaded from memory, either to be modified or used to determine if neighboring samples will be modified. Since the loop filter is a so heavy computing step that several VLSI architectures were presented to solve these problems in the past three years. For example, multiple parallel pipelines with same stages [6] were first introduced to calculate all possible filtered results to overcome the conditional branches, and the advanced 2-D filter order [7] was first introduced to make full data reuse and reduce data access. DSP is another powerful tool for video applications, while conditional branches are very time consuming and are also quite a challenge for parallel processing in DSP hardware [5]. Since DSP can only support one pipeline at a time, mask operation was used to avoid conditional jumping and pair processing was adopted to increase the parallelism of a software pipeline [8].

All the above mentioned methods are targeted for H.264, so the purpose of this paper is to provide pipelined DSP solutions to accelerate the in-loop deblocking filter for AVS1-P2. The rest of this paper is organized as follows. In Section 2, global filter flow is improved and divided into six sub-processes. Then software pipelines are designed for edge filter in Section 3. Simulated results are demonstrated in Section 4 to show the capability of the implementation. Finally, the paper is concluded in Section 5.

2. GLOBAL FILTER CONTROL

Three are mainly three parts in the loop filter, including boundary strength (Bs) decision, edge filter and filter control. In this section, global filter structure is to be improved from the viewpoint of pipeline processing.

2.1. Original Filter Flow

The minimum block partition adopted by AVS1-P2 is 8x8 block. Accordingly, the loop filter is performed on the edges of 8x8 blocks. Vertical filter is first performed then horizontal filter. The original detailed process of edge filter in AVS1-P2 is shown in Fig. 1. Bs is used to control the filter strength performed on edge level. Six pixels (p2, p1, p0, q0, q1, and q2) across a vertical or horizontal boundary and two threshold values (α, β) are used to control sample-level filter. P1, P0, Q0, and Q1 are the filtered outputs. For further information, please refer to the AVS1-P2 standard [2].

2.2. Process Division

As shown in Fig. 1, the whole process contains so many conditional operations, such as jumping, clipping, and threshold-decision that DSP cannot build up an efficient pipeline for the whole process. So we divide the whole process into six parts, and then design pipelines for each part.

First, one macroblock (MB) line is set as a macro process unit. Once all the Bs within the MB line have been determined, the luminance edges can be consecutively filtered, so do the

* Supported by the National Natural Science Foundation of China under Grant No. 60672088.

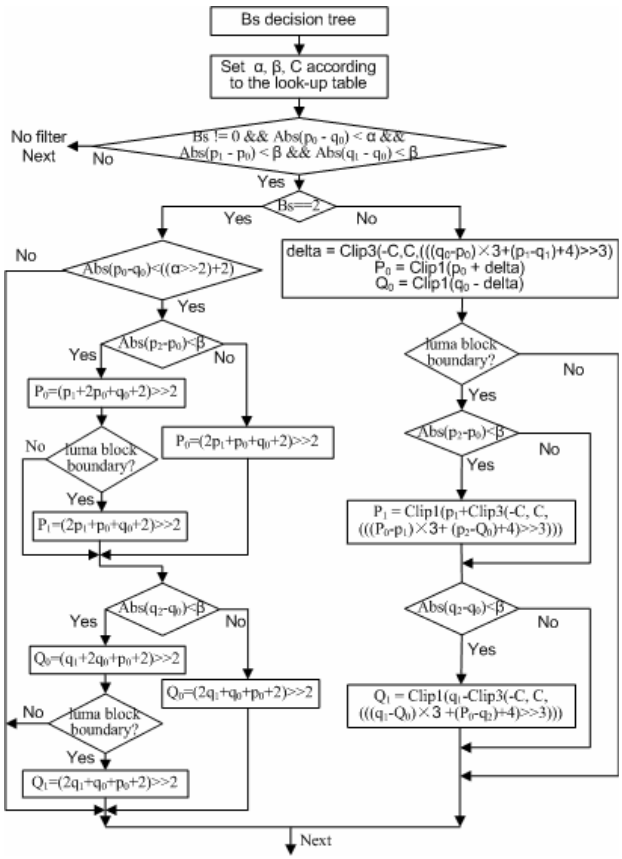

Fig. 1. Edge filter in AVS1-P2

chrominance edges. Thus, the process of Bs decision can be divided into three sub-processes according to the picture type, because the calculations for Bs are regular under different type. For instance, in I picture, all the Bs are equal to 2 except for picture or slice boundaries, and in P or B picture, the motion vector array and reference index array can be regularly accessed to decide each Bs.

Second, the edge filter should be divided into multiple sub-processes with fewer conditional operations, and each sub-process should be independent with others. Luminance/chrominance is the most obvious division. When filtering chrominance pixels, only p0 and q0 are filtered, so chroma-MB edge filter can be a simple sub-process and suitable for pipeline design. While for luminance pixels, Bs can be a further division. When Bs=2 (Bs=1), the left (right) branch in Fig. 1 denotes the relevant sub-process.

As a summarization, Table I lists the above six sub-processes and their main functions.

TABLE I

SUB-PROCESSES IN THE LOOP FILTER

Sub-process		Function
Bs decision	1	Bs decision within a MB line in I picture
	2	Bs decision within a MB line in P picture
	3	Bs decision within a MB line in B picture
edge filter	1	luma-block edge filter with Bs=1
	2	luma-block edge filter with Bs=2
	3	chroma-MB edge filter

2.3. Improved Filter Control

Good data flow and global function control are the basis of efficient DSP implementation. In order to achieve better data reuse, regular data access to memory is a key point. Based on the macro process unit mentioned in Section 2.2 and the six sub-processes listed in Table I, the improved global filter control is shown below:

```
for (each MB line in the picture){
   Perform one sub-process of Bs decision according to
      picture type;
   Load one luminance MB line to on-chip memory;
   for (each luminance MB in the MB line) {
      for (each luma-block edge in current MB)
         Perform one sub-process of luma-block edge filter
            according to Bs;
   }
   Load two chrominance (UV) MB lines to on-chip memory;
   for (each chrominance (UV) MB pair in the MB line) {
      for (each chroma-MB edge in current MB)
         if (two Bs of the edge are not all zero)
            Perform the sub-process of chroma-MB edge filter;
   }
}
```

Since the edge filter cannot be performed on slice boundaries, extra slice boundary check is required. In fact, slice boundary check can be put into the pipeline of Bs decision, i.e. Bs of the slice boundary can be set to zero. Therefore, only Bs is used to determine whether to perform edge filter or not. Such improvement can reduce the overhead of slice boundary check.

3. PIPELINES DESIGN

Since edge filter is the most complex part in the loop filter, pipelines design for this part is to shown in this section. Our work is based on TI TMS320C64x DSP [9]. A brief introduction to the DSP would help to ease the understanding of the specific design.

3.1. Instruction Set and CPU Data Paths

DSP has a powerful instruction set. Many enhanced instructions in assembly language can deal with multiple data in parallel. Properly choosing the enhanced instructions can bring a great increment for pipeline efficiency. The CPU in the DSP has two similar data paths (A/B), and each data path mainly consists of 32 register files and 4 functional units. As for this architecture, high parallelism is achieved by software pipeline. There are total eight functional units in CPU, so in a single clock cycle, CPU can execute a maximum of eight instructions in parallel, reaching its peak performance. More detailed information of this kind of DSP can be found in [9].

3.2. Pipelines Design for Edge Filter

In the sub-processes of edge filter, there are still conditional operations which do not meet the requirement of software pipeline, so the processing flow should be rearranged. An efficient solution is that translate the conditional jumping to conditional storing. This process can be implemented in a pipeline.

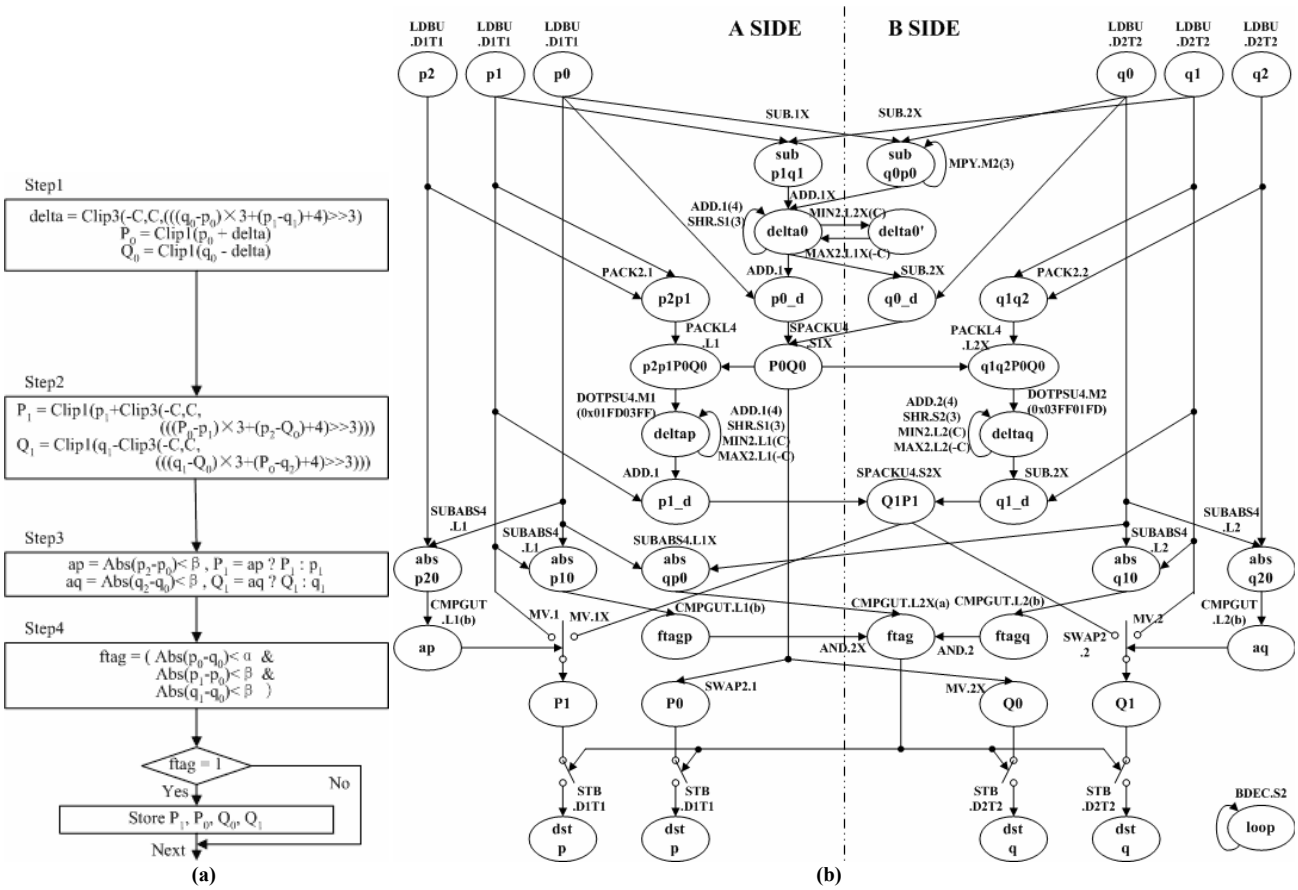

Fig. 2. Software pipeline design for luma-block edge filter with Bs=1, (a) Improved processing flow, (b) The inner loop dependency graph corresponding to Fig. 2(a). Note: in order to make the whole graph clear, we omit some nodes, which denote 5-bit constants or whose values saved in source register are constant during the loop, instead we put these constants and operands into parenthesis following the instruction and functional unit, e.g. SHR.S1(3) and MIN2.L2(C).

For further explanation, we set the sub-process of luma-block edge filter with Bs=1 as an illustrative instance, which is the most complex among the three sub-processes. From the pipeline-oriented viewpoint, Fig. 2(a) shows the improved processing flow of this sub-process. In the first two steps in Fig. 2(a), P1, P0, Q0, and Q1 are calculated despite any condition check. Since there are two additional spatial activity checks $|p2-p0|<\beta$ and $|q2-q0|<\beta$, conditional storing has to be performed on P1 and Q1 first, shown in step3 in Fig. 2(a). In the last step in Fig. 2(a), P1, P0, Q0, and Q1 are conditionally stored back to memory according to the result of sample-level content activity check. The flow of the other two filter sub-process can also be rearranged in the same way.

Next, a software pipeline is designed based on the improved processing flow. Fig. 2(b) shows the dependency graph of the loop kernel in the sub-process of luma-block edge filter with Bs=1. Each node in the dependency graph denotes an operand, and an edge connecting the nodes denotes an instruction. Moreover, Fig. 2(b) also indicates how to allocate functional units and what instructions have been used.

(1) Functional unit allocation. All the resources on each side of CPU data path should be balanced. These resources mainly include register files and all kinds of functional units. Under such rules, side A is in charge of most operations for p2, p1, p0, while side B is in charge of most operations for q2, q1, q0. For example, calculations for deltap and deltaq, shown in Fig. 2(b), contain same operations that can be easily assigned to side A/B; but for delta0, there are no similar calculations, so the operations for delta0 need to be carefully separated into each side.

(2) Instruction selection. Properly choosing the enhanced instructions can achieve high parallelism. For example, in Fig. 2(b), the PACK2 and PACKL4 instructions are used together to pack four pixels into a 32-bit register, then the DOTPSU4 instruction can compute the dot product of the four pixels; and the MIN2, MAX2, and SPACKU4 instructions can implement the clipping and threshold-decision operations.

Thus, a pipeline can be built up following the guide line from the dependency graph. Software pipelines for the other two sub-processes can also be designed in the same way. The detailed cycle statistic for each pipeline is listed in Table II.

4. SIMULATED RESULTS

The comparison between the proposed DSP strategy and another strategy in [8] is presented in Table II. The table shows that the proposed strategy produces smaller loop kernels and requires less cycles to perform edge filter than the reference. This is because the techniques adopted in the reference introduce additional operations to the loop kernel, and they would be more efficient only if there are more conditional operations in the loop kernel.

TABLE II
CYCLE STATISTIC COMPARED WITH ANOTHER DSP STRATEGY

Proposed						Mask operation + Pair processing [8]					
Pipeline	Loop Kernel	Loop Count	Prolog Epilog	Function Call	Total	Pipeline	Loop Kernel	Loop Count	Prolog Epilog	Function Call	Total
luma/Bs=1	10	8	25	6	111	luma/Bs=1	11	8	27	6	121
luma/Bs=2	8	8	20	6	90	luma/Bs=2	9	8	25	6	103
Chroma	9	8	24	6	102	Chroma	10	8	27	6	113

TABLE III
CYCLE STATISTIC FOR PIPELINE/NON-PIPELINE PARTS

Pipeline	Cycles	Non-Pipeline	Cycles
luma-block edge filter with Bs=1	111	local filter control per luma MB	177-393
luma-block edge filter with Bs=2	90	local filter control per chroma MB	52-106
chroma-MB edge filter	102	global filter control per luma MB line	622
Bs decision within a MB line in I picture	61	global filter control per chroma MB line	331
Bs decision within a MB line in P picture	990	L1D cache read miss per luma MB line	123
Bs decision within a MB line in B picture	1446	L1D cache read miss per chroma MB line	94

TABLE IV
CAPABILITY (FPS) OF THE PROPOSED DSP STRATEGY TESTING ON 720X480 SEQUENCES (IBBPBBP)

Seq.\QP	20	24	28	32	36	40	44
football	340	353	377	398	438	513	592
mobile	549	569	613	653	713	787	877
news	802	876	947	991	1048	1117	1166

In the following experiments, D1 (720x480) format videos are tested with AVS1-P2 on TMS320C64x DSP [9] running at 600MHz. Besides the cycles spent on the six sub-processes, we also consider the cycles for non-pipeline parts, including local filter control per MB, global filter control per MB line and L1D cache read miss. Table III lists the detailed cycle statistic for each part. Since Bs is pre-calculated, filter control on edge level could be achieved. So the minimum and maximum cycles for one luminance MB are 177 and 393+111×8=1281 respectively, and the minimum and maximum cycles for two chrominance (UV) MBs are 52 and 106+102×4=514 respectively.

For the I picture, strong filter is performed on every edge except for the picture or slice boundary, so the capability of the proposed DSP strategy is steady at 276fps when filtering all intra coded 720x480 video. While filtering videos with normal GOP structure like IBBPBBP, the capability is dependent on QP and the content of videos, as shown in Table IV. As QP increases, it is a general trend that the capability becomes better and better. This is because when increasing QP, more and more MBs are coded with skip mode, and fewer and fewer MBs are intra coded, resulting in a decrement of both normal and strong filter operations. There is also another trend that the capability of filtering low motion video (like news) is much better than that of filtering high motion video (like football). This is simply due to the difference of motion vector between two adjacent blocks in Bs decision.

Finally, the capability of the proposed DSP strategy is 16 to 20 times faster than the original processing flow. And this strategy can satisfy real-time processing for high resolution videos.

5. CONCLUSIONS

This paper provides some pipelined DSP solutions to accelerate the in-loop deblocking filter in AVS1-P2 on the DSP platform. In order to design efficient software pipelines, the processes of edge filter and Bs decision are respectively divided into three sub-processes. And based on these sub-processes, the whole flow is regularly improved. Since the sub-processes contain fewer conditional operations, more efficient software pipelines are achieved with elaborately allocating functional units and carefully choosing instructions. The proposed techniques have been adopted in a developed DSP-based real-time AVS1-P2 decoder. The future work is to support multi-channel decoding or to integrate these techniques into a DSP-based encoder for pursuing high coding speed.

6. REFERENCES

[1] AVS Working Group Website: http://www.avs.org.cn

[2] "Final draft of information technology – advanced coding of audio and video – part 2: video," in AVS workgroup Doc. N1214, Shanghai, China, Sep. 2005.

[3] "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC)," in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050, 2003.

[4] S. D. Kim, J. Yi, H. M. Kim, and J. B. Ra, "A deblocking filter with two separate modes in block-based video coding," *IEEE Trans. CSVT*, vol. 9, pp. 156-160, Feb. 1999.

[5] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, and Marta Karczewicz, "Adaptive deblocking filter", *IEEE Trans. CSVT*, vol. 13, No. 7, pp. 614-619, July 2003.

[6] Miao Sima, Yuanhua Zhou, Wei Zhang, "An efficient architecture for adaptive deblocking filter of H.264/AVC video coding", *IEEE Trans. CE*, vol.50, No. 1, pp. 292-296, Feb. 2004.

[7] Bin Sheng, Wen Gao, and Di Wu, "An implemented architecture of deblocking filter for H.264/AVC," in *Proc. ICIP*, vol. 1, pp. 665-668, Oct. 2004.

[8] Zhigang Yang, Wen Gao, Yan Liu, and Debin Zhao, "Deeply pipeline solution to deblocking filter for H.264/AVC", *IEEE Trans. CE*, Vol. 52, No. 4, pp. 1267-1274, Nov. 2006.

[9] TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide, SPRU732A, Jun. 2005, http://www.ti.com