Deeply Pipelined DSP Solution to Deblocking Filter for H.264/AVC

Zhigang Yang, Wen Gao, Yan Liu, and Debin Zhao

**Abstract — ***The in-loop deblocking filter in H.264/AVC 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 quite difficult to efficiently implement the filter on digital signal processor (DSP) platform. In this paper, deeply pipelined DSP solutions to both edge filter and boundary strength decision are presented. To avoid conditional jumping when performing edge filter, DSP first calculate all possible filtered outputs, then combine them with corresponding masks to get single output for each sample, and last conditionally store the combined output based on the final content activity check result. Moreover, on the basis of the symmetry of filtering on both sides of the edge, two symmetrical samples are packed into one register for "pair processing" to increase the parallelism of the pipeline. While for boundary strength decision, two-pass pipelines are designed. In the first pass, strengths are created by using the inter-relative information on block level, and then in the second pass, these strengths are refined based on the intra mode checks on macroblock level. To cooperate with global filter control and also to keep pipelining, a two-level internal memory organization is presented as well. The simulated results indicate that this efficient implementation can support real-time filtering for high resolution videos[1].*

Index Terms — DSP, software pipeline, memory organization, deblocking filter, H.264/AVC.

I. INTRODUCTION

H.264/AVC [1] is an international standard for the compression of natural video images. The framework of H.264/AVC is block transform/block prediction hybrid based scheme, which can introduce the blocking artifact [2], [3], especially in highly compressed or low bit rate video environment. H.264/AVC adopts in-loop deblocking filter to reduce this subjective artifact and to improve the objective PSNR as well [4].

According to the H.264/AVC baseline profile decoder complexity analysis by M. Horowitz et al., loop filter is the largest component which can account for 33% of the decoder complexity [5]. P. List et al. explained the main reason for its high complexity is that conditional branches inevitably appear in the inner loops of the filter algorithm. And they also pointed out that such highly adaptive processing is quite a challenge for parallel processing on DSP [4]. Another reason for the high complexity is the wide data access. In the worst case, samples in a macroblock need to be loaded into machine registers four times and be stored back to memory two times, either to be modified or to determine if neighboring samples will be modified.

Since loop filter is a so heave computing step that several VLSI architectures, e.g. [6], [7], were presented and successfully solved the two problems in the past three years. Among these architectures, M. Sima et al. first introduced multiple parallel pipelines running at same time to calculate all possible filtered outputs to overcome the conditional processing on the block edge [6], and B. Sheng et al. first introduced the advanced 2-D filter order to make full data reuse and reduce the data access [7].

While on DSP platform, the current state of optimization is only done on high-level control. For instance, S. Wang et al. unrolled the filter code into several versions to reduce the overhead of checking the filter direction, and they also take frame type into consideration to rearrange some conditional check [8]. Till now, there is still no efficient pipeline solution to loop filter. This is mainly because DSP can only support one pipeline at a time. Due to a large amount of filter conditions and limited DSP register number, normal algorithm improvement like what VLSI architecture [6] and [7] have adopted, is usually inefficient for DSP pipeline and always leads to worse DSP performance.

The purpose of this paper is to provide a deeply pipelined DSP solution to accelerate the deblocking filter process for H.264/AVC. The rest of this paper is organized as follows. In Section II, an overall DSP solution on algorithm level is introduced. In Section III, pipeline design for edge filter is described in detail. In Section IV, two-pass pipelines for boundary strength are designed. In Section V, two-level internal memory organization to keep pipelining is presented. And the simulated results are demonstrated in Section VI to show the capability of the implementation. Finally, the paper is concluded in Section VII.

II. OVERALL DSP SOLUTION ON ALGORITHM LEVEL

Three are three parts in deblocking filter in H.264/AVC, including filter control, boundary strength (Bs) decision, and edge filter. The detailed description of adaptive deblocking filter can be found in [4]. A typical time proportion of each part is shown in Fig. 1. How to improve the structure of the three parts from the DSP-oriented viewpoint will be introduced in this section.

[1] Z. Yang, W. Gao, Y. Liu, and D. Zhao are with Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. W. Gao is also with Institute of Computing Technology, Chinese Academy of Science, China. (e-mail: {zgyang, wgao, liuyan, dbzhao}@jdl.ac.cn).

Contributed Paper
Manuscript received July 16, 2006

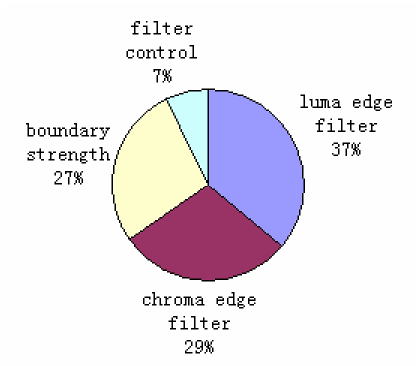


Fig. 1. The profiling of deblocking filter under the original processing flow, testing on QCIF sequence bus when QP=32

A. 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. So we set one macroblock (MB) line as a macro process unit. Once all Bs within the MB line have been created, the luminance edges can be consecutively filtered, so do the chrominance edges. The improved filter control is shown below:

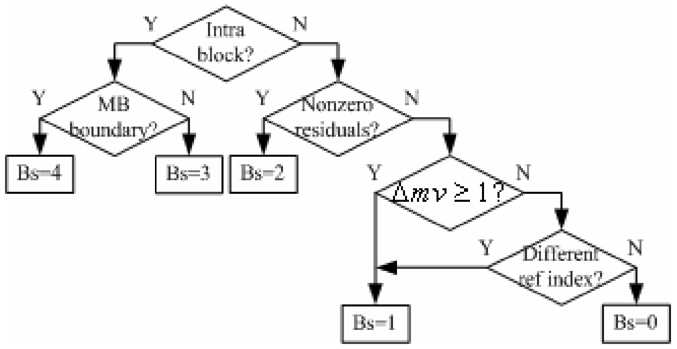
```
for (each MB line in the picture){
    Get all Bs within current MB line;
    for (each luminance MB in MB line) {
        for (each luminance edge in current MB) {
            if (four Bs of the edge are not all zero)
                Filter the luminance edge;
        }
    }
    for (each chrominance (UV) MB pair in MB line) {
        for (each chrominance edge in current MB) {
            if (four Bs of the edge are not all zero)
                Filter two chrominance (UV) edges;
        }
    }
}
```

The edge filter cannot be performed on picture or slice boundaries, so extra slice boundary check is required in the original filter control. In fact, slice boundary check can be put into the pipeline of Bs decision, that is to say, Bs of the slice boundary can be set to zero. Therefore, only Bs is used to determine whether perform edge filter or not. Such improvement can reduce the overhead of slice boundary check.

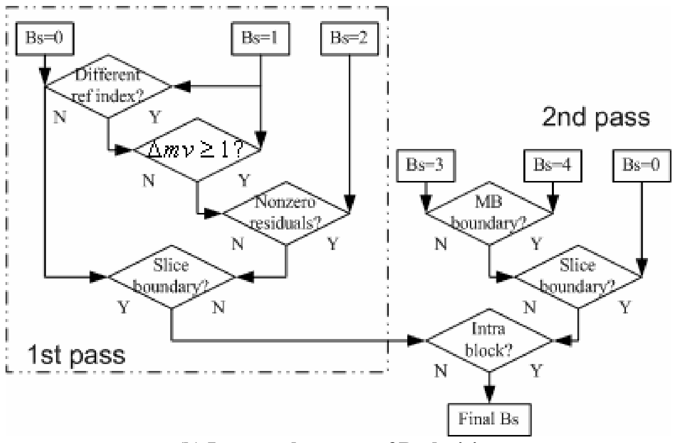
This improved filter control is also helpful for two-level internal memory organization design, which is going to be introduced in Section V.

B. Boundary Strength Decision

Bs parameter which is assigned an integer value from 0 to 4, determines the filter strength performed on edge level. Bs is derived from block modes and coding conditions of the two adjacent blocks, such as motion vector (MV), reference index, and nonzero residuals. The original Bs decision order is from strong (Bs=4) to weak (Bs=0), and a brief demonstration is shown in Fig. 2(a). This check order results in conditional jumping which prevents DSP building pipelines. So we inverse the check order and translate the conditional jumping to conditional storing, which allow being used in pipeline. The improved Bs decision process is shown in Fig. 2(b), including the mentioned slice boundary check in Section II-A.



(a) Original process of Bs decision



(b) Improved process of Bs decision

Fig. 2. Comparison between original and improved process of Bs decision

In order to exert the efficiency of DSP, two-pass pipelines are designed for Bs decision. The first pipeline corresponds to the left part of Fig. 2(b) before the "intra block" check. The inter-relative information is used to calculate Bs within one MB line regardless of block mode. Then, on the basis of the output of first pass, the second pipeline checks all block modes to create final Bs within the MB line. We adopt two-pass pipelines, mainly because that the block mode check is on 16x16 MB level, while the conditions of MV, reference index and nonzero residuals are on 4x4 block level. If all these conditions are designed together in one-pass pipeline, "intra block" will be checked many redundant times. Furthermore, one-pass pipeline design requires so many general registers to keep temporary outputs and conditional registers to keep check results, that sometimes DSP cannot afford all these registers.

C. Edge Filter

To detect whether two adjacent blocks have blocking artifacts or not, eight samples across either a vertical or horizontal boundary (shown in Fig. 3) are analyzed. For convenience, the related conditions are listed in Table I. Two

TABLE I
CONDITIONS IN EDGE FILTER AND CORRESPONDING VARIABLES

Condition	Variable
Bs>0 && |q0-p0|<α && |p1-p0|<β && |q1-q0|<β	ftag
Bs=4	s4
!s4	ns4
|q2-q0|<β	aq
|p2-p0|<β	ap
aq && |p0-q0|<(α>>2)+2	aq4
ap && |p0-q0|<(α>>2)+2	ap4
!aq4	naq4
!ap4	nap4

threshold values α and β are used to control sample-level filter. Two filtering modes are selected based on the Bs parameter. A brief description is shown below, where P2, P1, P0, Q0, Q1, and Q2 denote the filtered outputs.

- When Bs=1, 2, or 3, a 4-tap filter and clipping operation are applied to produce P0 and Q0. For luminance component, if condition ap is true, a 4-tap filter and clipping operation are applied to produce P1; if condition aq is true, Q1 is similarly produced.
- When Bs=4, for p values, if condition ap4 is true, strong 4-tag and 5-tap filter is applied to produce P0, P1, and P2 (P2 only for luminance). Otherwise, a weaker 3-tap filter is applied to only produce P0. The q values are modified in a similar manner.

To solve the problem of so many complex operations and conditional branches, we first divide the edge filter process into several sub-processes, such as luminance/chrominance edge filter and vertical/horizontal edge filter, and each processes corresponds to one pipeline. Such divisions are based on the conditions which are not necessary in the inner loops of the filter algorithm, to decrease the required register number. Then, the improved edge filter process is shown below:

Step1: Load all required samples and get the results of all conditions.

Step2: Calculate all possible filtered outputs.

Step3: Produce all possible sample masks, and then combine the filtered outputs with masks to get single output for each sample. If conditions ftag holds true, store the combined results.

In the last step, we do not adopt fully conditional storing, because there are so many conditions require keeping their values that this amount exceeds the maximum of conditional registers in DSP. Instead, we choose logical mask operations to achieve the same task. Moreover, since the loop filter has symmetrical operations on both sides of the edge, symmetrical samples from p and q can be packed into one register for "pair processing" to increase the parallelism of the pipeline. At the end of this section, a brief process flow of (Q0, P0) is shown in Fig.4 to provide a more clear description of the improved edge filter process.

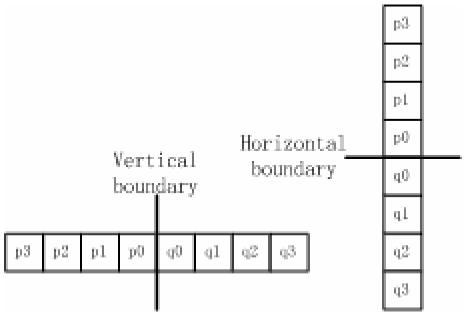


Fig. 3. Samples across vertical or horizontal boundary

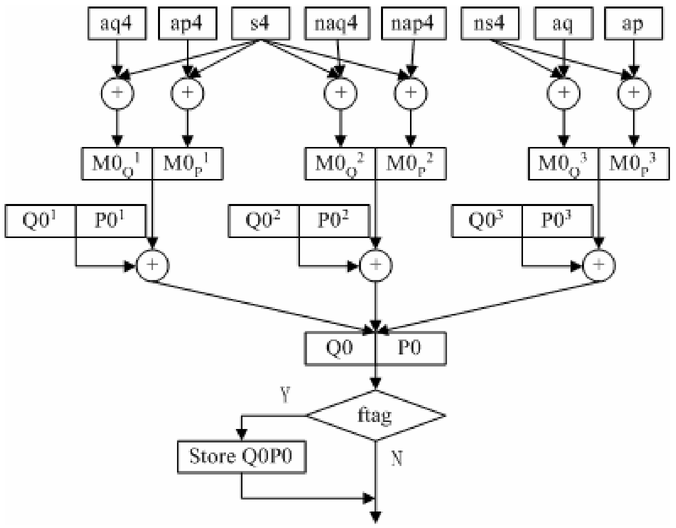


Fig.4. Improved process flow of (Q0, P0)

In the next section, we are going to elaborate on the detailed implementation of edge filter.

III. PIPELINE DESIGN FOR EDGE FILTER

Following the three processing steps presented in Section II-C, a highly parallel software pipeline is designed for luminance vertical edge filter in this section. Our work is based on Very Long Instruction Word (VLIW) DSP [9], so a brief introduction to the DSP instruction set and the central processing unit (CPU) data paths would help to ease the understanding the specific design.

A. 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 detailed description of the DSP instruction set and CPU data paths can be found in [10] and [11].

The CPU has two data paths, A and B (shown in Fig. 5). The main components consist of:

- Two general-purpose register files (A and B).
- Eight functional units (.L1, .L2, .S1, .S2, .M1, .M2, .D1, and .D2).
- Two register file data cross paths (1X and 2X).
- Two load-from-memory data paths (LD1 and LD2)
- Two store-to-memory data paths (ST1 and ST2)

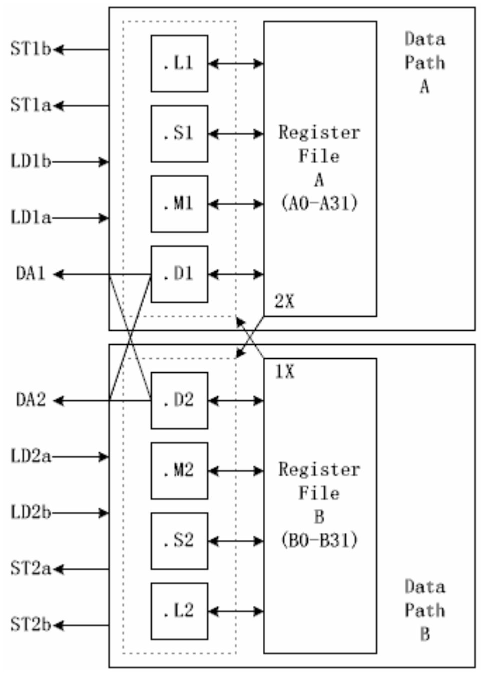


Fig. 5. CPU Data Paths

Parallelism is the key to extremely high performance. 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. To design high parallel software pipeline is one of the most important parts in DSP applications.

Dependency graph is a useful tool for efficient DSP software pipeline design. Each node in the dependency graph denotes an operand, and edges connecting the nodes denote instructions. The dependency graph can also indicate how to allocate data paths. So in the rest of this section, we will show the dependency graph of each step and give the key point of each design.

B. Conditions (Step1)

Because the filter process contains symmetrical operations on both sides of the edge, two symmetrical conditions listed in Table I can be put into one register to form a condition pair, such as aqap, aq4ap4, and naq4nap4. While for s4 and ns4, two condition pairs, s4s4 and ns4ns4 are also created to cooperate with other pairs.

(1) Instruction selection. LDNDW instruction can load eight samples from both sides of a vertical edge into a register pair, q3q2q1q0:p0p1p2p3. Then SHLMB, SWAP4, and a set of PACK instructions can pack the loaded samples into q2p2q1p1, q0p0q0p0 and p0q0p0q0, which can be processed by SUBABS4 and CMPGTU4 instructions, where SUBABS4 instruction can do subtraction with absolute value for four 8-bit data, and CMPGTU4 instruction can do four 8-bit data comparison. At the same time, Bs is loaded to calculate the rest conditions.

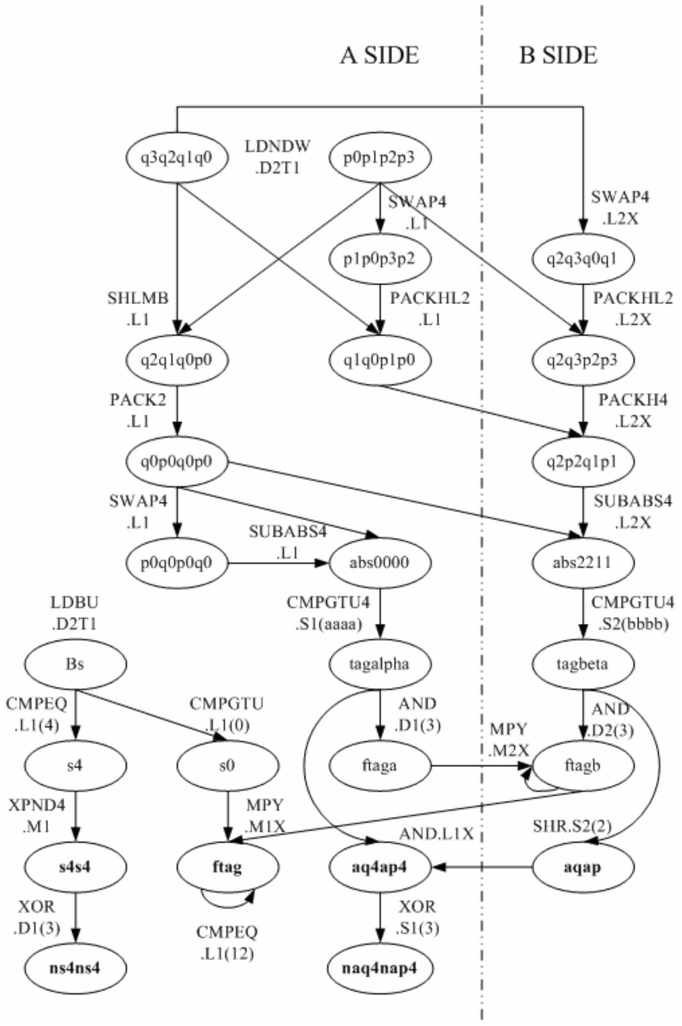


Fig. 6. Dependency graph of Step1 for luminance vertical edge filter, Note that in order to make the whole graph clear, we omit some nodes, which denote 5-bit constants or whose values saved in source register are constant during the loop, instead we put these constants and operands into parenthesis following the instruction and functional unit, e.g. CMPEQ.L1(4). Another notice, "aaaa" and "bbbb" are the packed α and β values for four 8-bit data comparison.

(2) Data path allocation. Most of the instructions belong to logical, compare and pack operations, which must be assigned to .D, .L or .S functional unit. As a consequence, .M is seldom used. To fill the pipeline, we translate some logical operations to multiply operations when calculating ftag. In this step, side A has more operations than side B. This is because all the samples are loaded into side A at the every beginning of Step1. However, the two data path can be balanced in the following two steps.

Fig. 6 shows the dependency graph of Step1 for luminance vertical edge filter. At the end of this step, condition ftag and five condition pairs, s4s4, ns4ns4, aqap, aq4ap4, and naq4nap4 are produced.

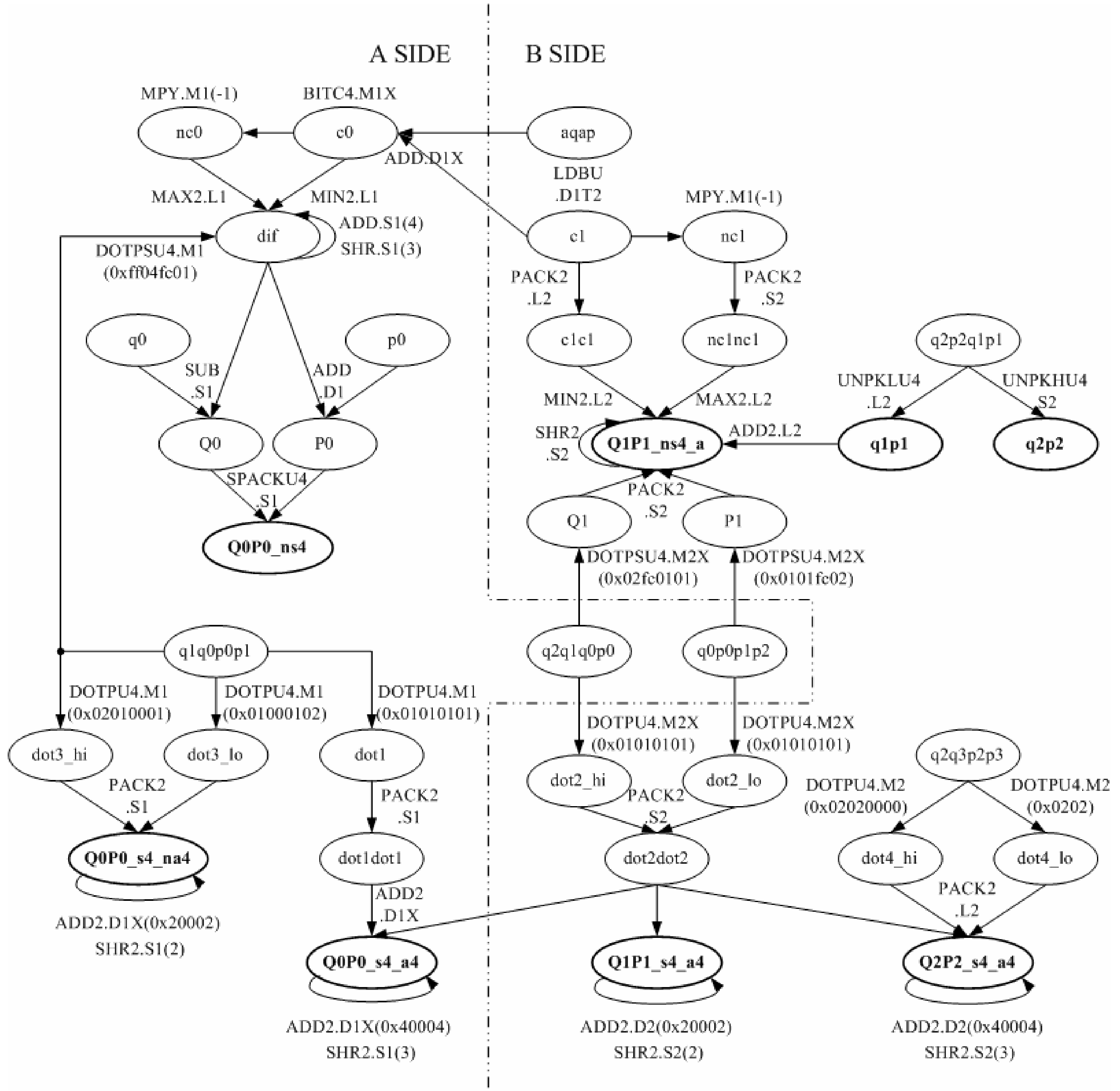


Fig. 7. Dependency graph of Step2, all possible filtered outputs calculation

C. All Possible Filtered Outputs (Step2)

To produce all possible filtered outputs is the most complex step in edge filter process. In this step, there are lots of operations, so functional unit should be carefully assigned and proper instruction should be elaborately chosen.

(1) Instruction selection. DOTPSU4 instruction can compute the dot product of the four 8-bit samples, so it is used for x-tap filter. ADD2 and SHR2 instruction can deal with two 16-bit data, so they are used to do sample pair's addition and shift. MIN2 and MAX2 instruction are used together to do clipping operations. SPACKU4 instruction is used for threshold-decision.

(2) Data path allocation. All the resources on each side of CPU data path should be balanced. These resources mainly include the used register files, all kinds of functional units, and data cross paths. Under such rules, side A is in charge of the filter operations for (Q0, P0), while side B is in charge of the filter operations for (Q1, P1) and (Q2, P2).

The dependency graph of Step2 is shown in Fig. 7. The upper part of the graph associates with normal strength filter process and the lower part of the graph represents strong filter process. At the end of this step, eight sample pairs are produced. They are Q0P0_s4_a4, Q0P0_s4_na4, Q0P0_ns4, Q1P1_s4_a4, Q1P1_ns4_a, q1p1, Q2P2_s4_a4, and q2p2.

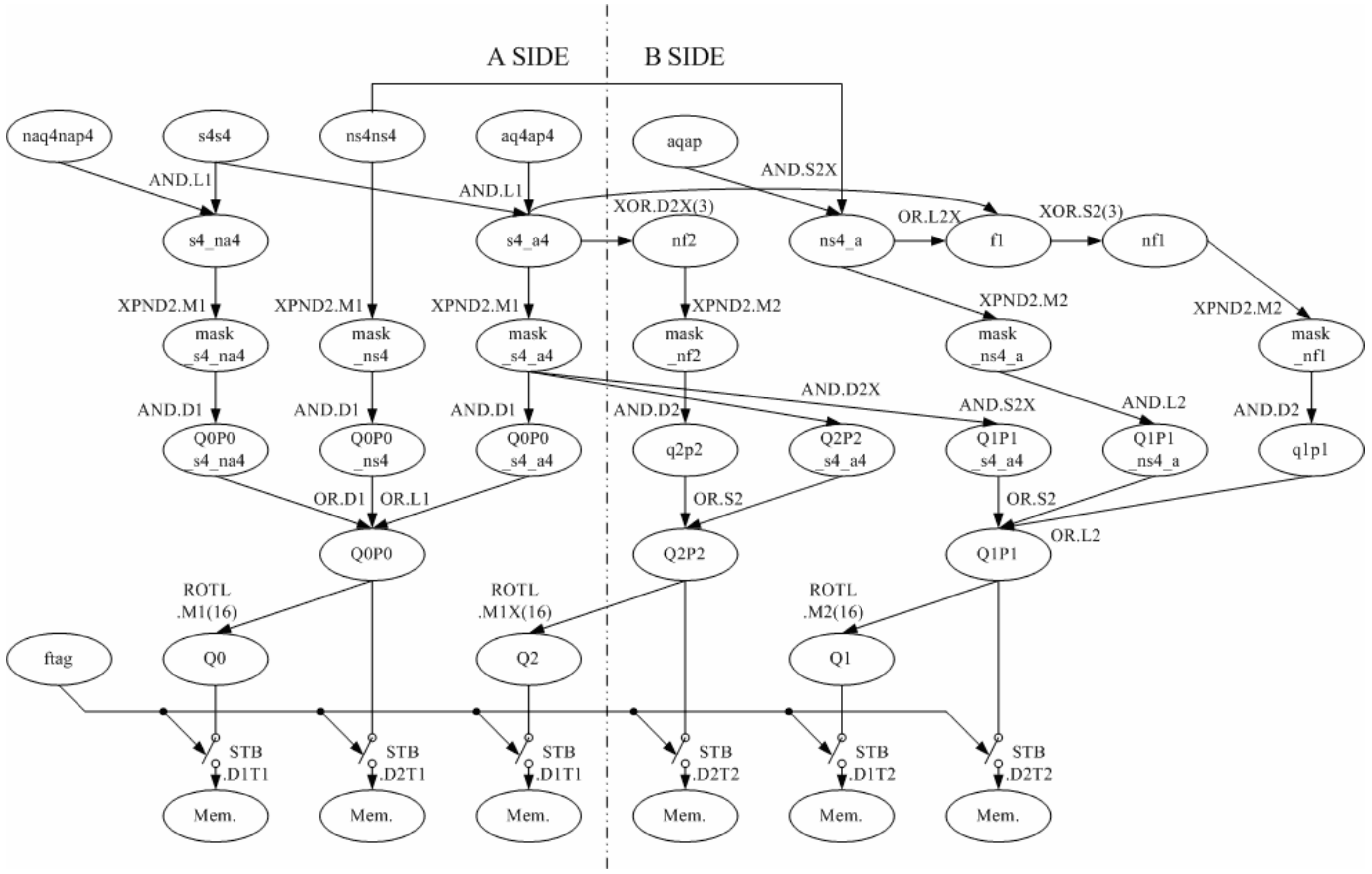


Fig. 8. Dependency graph of Step3, mask and conditional storing

D. Mask and Conditional Storing (Step3)

The condition pairs produced in Step1 and the sample pairs produced in Step2 are the inputs of Step3. According to pair process, the masks are also required to be produced in pair.

(1) Instruction selection. XPND2 instruction can expand bits into two 16-bit masks, and respectively write the two masks to the upper and lower half word of the destination register. So XPND2 instruction is used to create mask pair which is derived from the condition pair. Then some common logical instructions like AND, OR combine the sample pairs with the mask pairs to get single result for each sample pair.

(2) Data path allocation. Like Step2, side A is in charge of the operations for (Q0, P0), and side B is in charge of the operations for (Q1, P1) and (Q2, P2).

The dependency graph of Step3 is shown in Fig. 8. At the end of this step, the combined outputs are conditionally stored based on the value of ftag.

E. Cycle Statistics

Fig. 6, 7, 8 compose the kernel of pipeline for luminance vertical edge filter. Pipeline for chrominance or horizontal edge filter can be similarly designed. The cycle statistics for each pipeline are listed in Table II. The samples across the horizontal edge are discretely stored, so more instructions are required to load sample in turn from the memory. This is the main reason that horizontal edge filter always spends more cycles than vertical edge filter.

TABLE II
CYCLE STATISTICS OF FOUR EDGE FILTER PROCESS

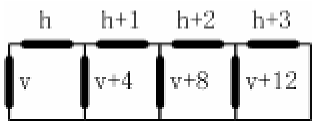
Pipeline	Loop Kernel	Loop Count	Prolog Epilog	Function Call	Tota l
luma/vet	15	16	22	6	268
luma/hor	16	16	31	6	293
chroma/vet	11	8	24	6	118
chroma/hor	12	8	26	6	128

IV. PIPELINE DESIGN FOR BOUNDARY STRENGTH

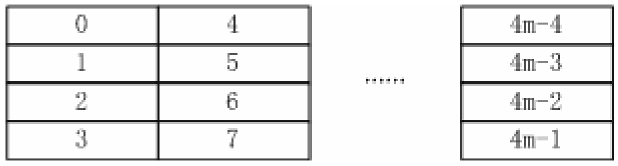
In this section, two-pass pipelines are designed for Bs. As mentioned in Section II-B, the two pass is on different block size level. They will be described one by one.

A. 1st Pass Pipeline Design

Fig. 2(b) has shown the DSP solution for single Bs decision from the algorithm level. In order to increase the parallelism of the pipeline under the DSP platform, four horizontal 4x4 blocks in one MB are processed together in the loop kernel, to get four discrete vertical Bs and four consecutive horizontal Bs, shown in Fig. 9(a). Operations for vertical Bs are assigned to side A, and operations for horizontal Bs are assigned to side B. And Fig. 9(b) shows the process order. When processing the top three block lines, the information of MV and reference



(a) 4 horizontal Bs and 4 vertical Bs got from a group of four 4x4 blocks



(b) Process order of a group of four 4x4 blocks

Fig. 9. Process flow of 1st pass Bs decision

index can be reused for the next line. When processing the bottom block lines, relocation is required.

B. 2nd Pass Pipeline Design

The Bs produced in the first pass is the input of the second pass. The MB mode is checked, if current MB is intra coded, the entire 32 Bs of current MB are rewritten with 4 or 3.

In fact, for I picture, the first pass is unnecessary, and the second pass can also be simpler without any mode check.

C. Cycle Statistics

The first loop repeats 4m times, and the second loop repeats m times, where m denotes the MB width of the image. The cycle statistics for Bs decision are listed in Table III. This statistics are based on H.264/AVC baseline profile.

TABLE III
CYCLE STATISTICS OF TWO-PASS BS DECISION WITHIN ONE MB LINE

Pipeline		Kernel	Count	Prolog Epilog	Function Call	Total
P picture	1	15	4m	44	6	64m+50
	2	4	m			
I picture	1	-	-	8	6	2m+14
	2	2	m			

Note: m denotes the MB width of the image

V. MEMORY ORGANIZATION

There exists two-level internal memory [9] between the CPU and extended memory in DSP architecture. A good memory organization can prevent pipeline from being interrupted by cache read miss, and the improved global filter control in Section 2-A also requires consecutive filtering, so we elaborate on L1/L2 data scheme in this section.

The cache size of L1D and L2 is 16KB and 256KB respectively. If single L1D read miss occurs, 6 additional cycles need to be cost. However, the cache architecture allows pipelining read misses, and multiple parallel and consecutive misses consume only 2 cycles [12]. The useful routine "touch" [13] can load data into L1D with minimum cycle penalty by such read miss pipeline, so we are able to control the data flow from L2 to L1D by "touch" to get better DSP performance.

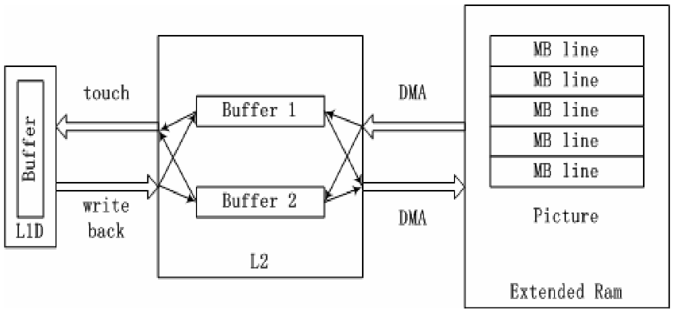


Fig. 10. Block diagram of two-level internal memory organization for deblocking filter

Set D1 (720x480) format picture as example, L1/L2 data scheme are described below:

(1) L1 data scheme. The 16KB L1D can keep a whole MB line in it. To perform deblocking filter on these MBs, extra four pixel lines are required due to the top horizontal boundary filter (shown in Fig. 2). So the total data size from L2 to L1D is 720×(16+4) = 14400bytes. When the deblocking filter process finishes, the filtered data can be write back to L2 by cache controller.

(2) L2 data scheme. In this scheme, L2 cache is used as on-chip memory. According to L1D data scheme, the buffer in L2 should have at least 14400bytes to cooperate with L1D. To fully exert DMA background transfer between L2 and extended memory, double buffers are usually adopted. So the total buffer size for L2 is at least 14400×2bytes.

The above data scheme is for luminance picture, and it can be easily extended for chrominance data scheme with the same memory organization. Fig. 10 summarizes the above two-level internal memory organization for deblocking filter.

VI. SIMULATED RESULT

In our experiments, D1 (720x480) format videos are tested with H.264/AVC baseline profile on fixed-point DSP running at 600MHz. Besides the pipeline cycles for the edge filter and Bs decision which have been mentioned in Section III-E (Table II) and in Section IV-C (Table III), we also consider the non-pipeline cycles for local MB filter control, global MB line filter control (including background DMA startup between L2 and extended memory) and foreground data transfer between L1D and L2. Table IV lists the detailed cycles for each part. The minimum and maximum cycles for one luminance (Y) MB are 87 and 167+268×4+293×4=2411 respectively. And the minimum and maximum cycles for two chrominance (UV) MBs are 77 and 126+118×4+128×4=1110 respectively.

In 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 125fps when filtering all intra coded 720x480 video. While filtering video with normal GOP structure like IPPP…, the capability depends on QP and video content, as shown in Table V. As QP increases, it is a general trend that the capability becomes better and better.

TABLE IV
CYCLE STATISTIC FOR PIPELINE/NON-PIPELINE PARTS

Pipeline	Cycles	Non-Pipeline	Cycles
luma vertical edge filter	268	luma (Y) MB filter control	87-167
luma horizontal edge filter	293	chroma (UV) MBs filter control	77-126
chroma vertical edge filter	118	luma MB line filter control	669
chroma horizontal edge filter	128	chroma MB line filter control	691
Bs decision within a MB line in I picture	104	luma MB line transfer between L1D and L2	258
Bs decision within a MB line in P picture	2930	chroma MB line transfer between L1D and L2	188

TABLE V
THE CAPABILITY (FPS) OF THE PROPOSED DSP STRATEGY TESTING ON 720X480 SEQUENCES (IPPP…)

Seq.\QP	20	24	28	32	36	40	44
football	128	141	170	216	270	344	415
mobile	134	147	168	205	282	434	665
news	352	483	609	733	847	945	1021

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 an amount decrement of the edges to be filtered. There is also another trend that the capability of filtering low motion video is much better than that of filtering high motion video, such as news vs. football. This is simply due to the difference of motion vector between two adjacent blocks in Bs decision.

VII. CONCLUSIONS

This paper provides a deeply pipelined DSP solution to adaptive deblocking filter for H.264/AVC. The structure of whole filter process is improved to meet the requirement of software pipeline. Mask and conditional storing are adopted to solve the problem of conditional jumping. "Pair processing" is introduced to increase the parallelism of pipelines for edge filter. And two-pass pipelines are designed to further accelerate boundary strength decision. To keep pipelining and assist global filter control, two-level internal memory organization is also described. This high parallel design can support real-time deblocking filter process for high resolution videos.

REFERENCES

[1] "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.
[2] 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. Circuits Syst. Video Technol.*, vol. 9, pp. 156-160, Feb. 1999.
[3] Y.-L. Lee and H. W. Park, "Loop filtering and post-filtering for low-bit-rates moving picture coding", *Signal Processing: Image Commun.*, vol. 16, pp. 871-890, 2001.
[4] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, and Marta Karczewicz, "Adaptive deblocking filter", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, No. 7, pp. 614-619, July 2003.
[5] Michael Horowitz, Anthony Joch, Faouzi Kossentini, and Antti Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, No. 7, pp. 704-716, July 2003.
[6] Miao Sima, Yuanhua Zhou, Wei Zhang, "An efficient architecture for adaptive deblocking filter of H.264/AVC video coding", *IEEE Trans. Consumer Electron.*, 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. IEEE Int. Conf. Image Processing*, vol. 1, pp. 665-668, Oct. 2004.
[8] Sung-Wen Wang, Ya-Ting Yang, Chia-Ying Li, Yi-Shin Tung, and Ja-Ling Wu, "The optimization of H.264/AVC baseline decoder on low-cost TriMedia DSP processor", in *Proc. SPIE Conf. Applications of Digital Image Processing*, vol. 5558, pp. 524-535, Aug. 2004.
[9] TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor Data Manual, SPRS200G, Aug. 2004, http://www.ti.com
[10] TMS320C6000 CPU and Instruction Set Reference Guide, SPRU189E, Jan. 2000, http://www.ti.com
[11] TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide, SPRU732A, Jun. 2005, http://www.ti.com
[12] TMS320C64x DSP Two-Level Internal Memory Reference Guide, SPRU610B, Aug. 2004, http://www.ti.com
[13] TMS320C6000 DSP Cache User's Guide, SPRU656A, May 2003, http://www.ti.com

Zhigang Yang was born in Harbin, Heilongjiang Province, P. R. China, in 1980. He received BS degrees in Computer Science from Harbin Institute of Technology in 2002. From then on, he is pursuing MS and PHD degrees at the same time for video signal processing in Harbin Institute of Technology. He became a Student Member of IEEE in 2005. His research interests are video compression, digital signal processing and real-time video coding.