Architectural and Algorithmic Optimizations for Down-Scale Transcoding of Compressed Video

Bo Shen, Sumit Roy
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2001-258 (R.1)
October 31st , 2003*

E-mail: {boshen, sumit}@hpl.hp.com

video transcoding, MMX, SSE, compressed domain processing, MPEG

With the introduction of the next generation wireless network, mobile devices access increasingly more media-rich content. However, a key factor that prevents a mobile device from accessing multimedia content is that it does not have enough display real estate to render the content that is traditionally created for the desktop web client. Moreover, the wireless network typically has lower bandwidth compared to a wired network of the same time frame. Therefore, a transcoder is needed somewhere in the network to transfer multimedia content to the appropriate form factor and bandwidth requirement. This paper introduces an extremely fast transcoder of such kind. Comparing to the previous methods, additional 40% of computing power is saved while the quality of transcoding is maintained. This saving in computing resources maps into the benefit that more concurrent sessions can be supported on one transcoding device. This scalability is crucial for the wireless network to handle user requests that might be very intensive at times.

* Internal Accession Date Only
Approved for External Publication

Architectural and Algorithmic Optimizations for Down-Scale Transcoding of Compressed Video

Bo Shen, Sumit Roy
Mobile & Media System Lab
Hewlett-Packard Labs
{boshen, sumit}@hpl.hp.com

Abstract

With the introduction of the next generation wireless network, mobile devices access increasingly more media-rich content. However, a key factor that prevents a mobile device from accessing multimedia content is that it does not have enough display real estate to render the content that is traditionally created for the desktop web client. Moreover, the wireless network typically has lower bandwidth compared to a wired network of the same time frame. Therefore, a transcoder is needed somewhere in the network to transfer multimedia content to the appropriate form factor and bandwidth requirement. This paper introduces an extremely fast transcoder of such kind. Comparing to the previous methods, additional 40% of computing power is saved while the quality of transcoding is maintained. This saving in computing resources maps into the benefit that more concurrent sessions can be supported on one transcoding device. This scalability is crucial for the wireless network to handle user requests that might be very intensive at times.

1. Introduction

The Internet brings heterogeneous devices together. For rich media transmission over the wireless Internet, content adaptation is a key issue. The original content may have been coded at higher resolution and higher bit rate, say 720x480 at 2 to 8Mbps for DVD quality, or 320x240 at 1.5 Mbps for desktop clients connected to the Internet through T1 line. However, due to the characteristics of the mobile communication – low bandwidth channel and limited display real estate, a 100kbps video at a lower resolution is desired. Current 3G wireless communication is targeting at providing 128~384kpbs communication channel. Therefore, a transcoder is needed somewhere in the network to adapt the content to the appropriate size and bit rate.

A straightforward method to perform this transcoding is to decode the original stream, down sample the decoded frames to a smaller size and re-encode to a lower bit rate. Considering a typical CCIR601 MPEG-2 video, it takes the full house power of a 400Mhz CPU to perform a real-time decoding. Encoding is even more expensive, which makes the straightforward method non-practical. Furthermore, if the transcoding is provided as a network service between content provider and content consumer, one transcoding unit is expected to able to support as many concurrent sessions as possible. This scalability is crucial for the wireless network to handle user requests that might be very intensive at times. Therefore, it is extremely worthwhile to develop fast algorithms to reduce the CPU load for such kind of transcoding session.

In a previous work [1], we used a compressed-domain approach, where the motion information in the original video is reused. The approach significantly improved the performance since the costly motion estimation process is eliminated. However, all the frames in the original video need to be

reconstructed, or otherwise, a frequency domain intra version of the inter frames needs to be constructed based on the algorithms proposed in [4]. These processes still consume large amount of CPU cycles.

This paper lays out a further optimized video transcoding algorithm. A macroblock-aware approach is proposed to take advantage of compressed domain processing techniques. In this approach, the transcoder performs transcoding at the macroblock level therefore is able to take advantage of different combination of macroblock types to avoid reconstruction of the original frame. In addition, we choose to perform motion compensation in the pixel domain because it is easily optimized either by hardware DSP design or MMX-enabled coding on Pentium platforms.

The paper is organized as follows. In Section 2, we present a compressed-domain video transcoding algorithm and provide an optimized implementation based on MMX and SSE instructions. In Section 3, we propose a new algorithm that further improves the computation efficiency. Performance analysis and testing results are presented in Section 4. And finally, we conclude in Section 5.

2. Compressed domain video transcoding and architectural optimization

2.1. Hybrid transcoding system

A simplified version similar to the system proposed in [1] is illustrated in Figure 1. In this system, the original video is decoded, down sampled and subsequently re-encoded. However, the motion information in the down-sampled video is obtained directly from the original video. Therefore, a costly motion estimation process is saved.

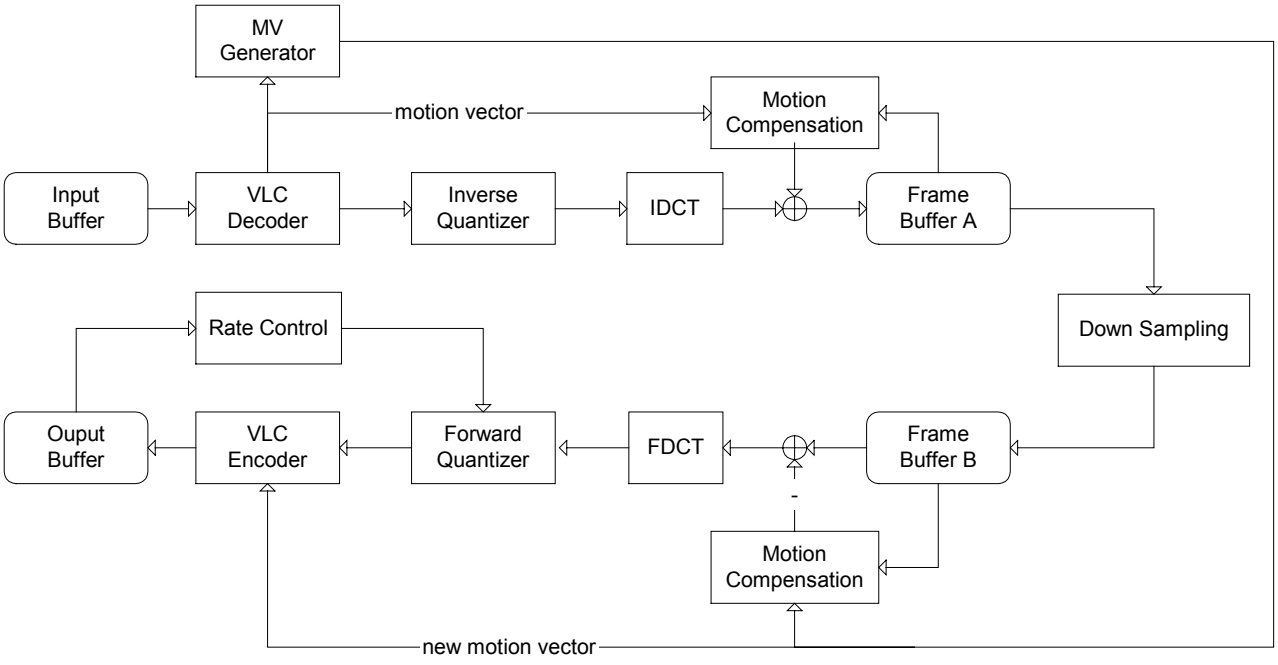

Figure 1 Compressed video transcoding system

This system represents a simplified version in which an open-loop approach is used. The open-loop approach may cause the error propagation problem on the receiver. However, the simplified system

performs much faster due to the elimination of one inverse quantization process and one IDCT process per loop.

2.2 MMX and SSE optimization

Based on the system described above, this section covers some architectural optimizations introduced via MMX and SSE instructions available in the Intel Pentium family of processors [9].

Intel introduced the MMX instructions in the Pentium family of processors. The technology is based on the Single-Instruction, Multiple-Data concept from parallel processing. Multiple small data items are packed into a 64-bit register and all processed in a single instruction. The initial set of 57 instructions operates on integers that are 8, 16, or 32 bits wide, and are known as the MMX extensions. Early implementations of this architecture multiplexed the MMX registers on the Floating Point registers and state.

The SSE instructions are introduced with the Pentium III family. They allow packed IEEE compliant single precision floating point arithmetic on four operands at a time. Some additional packed integer operations are also introduced.

The transcoder code is initially compiled without any optimization and with profiling enabled, that is with compilation flags *–O0 -pg*. The profiling data is then collected for two different input sequences, namely *flower-garden* and *football.* In each case, the output sequence is the result of reducing the input sequence by a factor of two in the horizontal and vertical dimensions, and a factor of four in the bit-rate. Table 1 shows the top ten procedures that take the most time as output by *gprof.*

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
17.77	13.44	13.44	1188000	0.01	0.01	fdct
17.73	26.85	13.41	5537394	0.00	0.00	recon_comp
14.28	37.65	10.80	991488	0.01	0.01	quant_non_intra
6.90	42.87	5.22	600	8.70	8.70	down2PixelPic
6.13	47.51	4.64	17843632	0.00	0.00	idctcol
5.02	51.31	3.80	2230454	0.00	0.00	addblock
3.98	54.32	3.01	1188000	0.00	0.00	sub_pred
3.62	57.06	2.74	17843632	0.00	0.00	idctrow
3.25	59.52	2.46	751416	0.00	0.00	pred_comp
2.93	61.74	2.22	31550362	0.00	0.00	flushbits

Table 1. Flat profile of transcoder, optimization *–O0*. `Each sample counts as 0.01 seconds.`

Before investigating the use of MMX and SSE instructions, the program was re-profiled with higher levels of compiler optimizations, *-O1, -O2*, and *–O3*. The results for *–O1* and *–O3* are shown in Tables 2 and 3 respectively.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
19.47	9.69	9.69	1188000	0.01	0.01	fdct
16.95	18.13	8.44	991488	0.01	0.01	quant_non_intra
15.75	25.97	7.84	5537394	0.00	0.00	recon_comp
6.01	28.96	2.99	17843632	0.00	0.00	idctcol
5.04	31.47	2.51	600	4.18	4.18	down2PixelPic
3.90	33.41	1.94	2230454	0.00	0.00	addblock

3.23	35.02	1.61	196512	0.01	0.01	quant_intra
3.07	36.55	1.53	600	2.55	36.12	getMBs
2.95	38.02	1.47	17843632	0.00	0.00	idctrow
2.91	39.47	1.45	31550362	0.00	0.00	flushbits

Table 2. Flat profile of transcoder, optimization *–O1*.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
19.62	9.51	9.51	1188000	0.01	0.01	fdct
17.71	18.09	8.58	991488	0.01	0.01	quant_non_intra
14.14	24.94	6.85	5537394	0.00	0.00	recon_comp
7.45	28.55	3.61	17843632	0.00	0.00	idctcol
5.30	31.12	2.57	600	4.28	4.28	down2PixelPic
3.88	33.00	1.88	17843632	0.00	0.00	idctrow
3.14	34.52	1.52	2230454	0.00	0.00	addblock
3.05	36.00	1.48	196512	0.01	0.01	quant_intra
2.91	37.41	1.41	751416	0.00	0.00	pred_comp
2.87	38.80	1.39	31550362	0.00	0.00	flushbits

Table 3. Flat profile of transcoder, optimization *–O3*.

It is to be noted that the top ten routines remain virtually identical across the levels of optimization, except for some internal reordering. Also, the compiler is able to reduce the execution time approximately 10~30% for these routines. The overall reduction in execution time is about 40% compared to the *–O0* case.

Referring to Figure 1, the procedures can be attributed to the following functional blocks:

1. FDCT: `fdct.`
2. Forward Quantizer: `quant_non_intra, quant_intra.`
3. Motion Compensation: `recon_comp, pred_comp, clear_ld_block, addblock.`
4. IDCT: `idctcol, idctrow.`
5. Down Sampling: `down2PixelPic.`
6. VLC Decoder: `getMBs, flushbits.`

Based on these profiles, it is decided to investigate the effect of MMX or SSE optimizations on various parts of the code. After optimizing the most expensive routine, the PSNR of the resulting sequence is compared to that of the original, un-optimized code. The execution time for transcoding each of the sequences by a factor of two and four is also measured. Since the MMX and SSE code consists of hand written assembly code, the optimization flag to the compiler has to be at least *–O1*. For the results shown, the optimization level is set to *–O3*.

Based on the profiling data collected at optimization levels *–O1* and *–O3,* the following code is replaced with an MMX/SSE optimized version:

1. Forward DCT (FDCT) - The MMX optimized FDCT code adapted from the mjpeg_beta1a package [6].
2. Quantization (quant) - The SSE optimized *quant_non_intra* function is from the mjpegtools-1.3b3 [7].
3. Motion Compensation - This uses the optimized motion compensation code from Aaron Holtzman's mpeg2dec library [8].

4. Inverse DCT (IDCT) - The MMX version uses the IDCT code from the optimized mpeg2dec.
5. Downsampling (down) - This optimization was done in-house.
6. Prediction (predict) - This optimization was done in-house.

Table 4 shows the profiling results after the FDCT and quantization optimizations have been applied.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
20.44	6.77	6.77	5537394	0.00	0.00	recon_comp
10.66	10.30	3.53	17843632	0.00	0.00	idctcol
8.24	13.03	2.73	600	4.55	4.55	down2PixelPic
5.22	14.76	1.73	17843632	0.00	0.00	idctrow
5.22	16.49	1.73	196512	0.01	0.01	quant_intra
4.74	18.06	1.57	31550362	0.00	0.00	flushbits
4.62	19.59	1.53	2230454	0.00	0.00	addblock
4.47	21.07	1.48	600	2.47	34.07	getMBs
3.47	22.22	1.15	991488	0.00	0.00	quant_non_intra
3.41	23.35	1.13				lp_mmx_fdct_row1

Table 4. Flat profile of transcoder, optimization *–O3, FDCT and Quantization*.

Note that the top two functional blocks in Table 3 have been dropped to 10th and 9th respectively. The other optimizations were applied similarly. The overall results are discussed in Section 4.

3. Algorithmic Optimization

3.1 Main Idea

Some previous work [2, 3] illustrated how to perform a down-sampling operation on compressed images using discrete cosine transform (DCT). They usually yields 40~80% computational saving. On the other hand, resolution-reduction of compressed video is much more complicated due to the inter-frame dependency. Specifically, there are two tracks of dependencies during the transcoding. The first is the dependency among frames in the original video, the second dependency among frames in the down-sampled version. Even when the motion information in the original video is reused, it is necessary to reconstruct the reference frames for the decoding of future frames of the original video [1]. However, if a reconstructed version of the original video is needed anyway, we cannot take advantage of the optimized DCT domain algorithms.

The next logical approach is try to form an intra DCT version of the inter frames so that the DCT domain down-sampling algorithms [2, 3] can be employed. However, this approach requires the motion compensation to be performed in the DCT domain. Such methods have been described in [4], and they do provide computational advantages in terms of the number of basic CPU operations required. To their disadvantage on the other hand, these methods tend to have irregular data access pattern that is not very easy to optimize. Moreover, constructing an intra DCT version of the inter frames consumes similar computation resources as reconstructing a pixel domain version.

In summary, both of the above mentioned transcoding approaches require the reconstruction of the frames on the picture level, i.e., each frame is reconstructed entirely either to pixel domain version or intra DCT version. Therefore the reconstruction still poses as a bottleneck.

To solve this problem, we have to consider the transcoding session at the macroblock level, that is, the transcoder has to be macroblock-aware. For example, in down-sample-by-two operation where both horizontal and vertical size of the original video is reduced by a factor of two, four original macroblocks generate one output macroblock. If all four of the original macroblocks are intra-coded, the DCT domain down sampling methods [2, 3] can be directly applied to generate the output macroblock in the down-sampled version. However, we still need the original size reference frame for the decoding of future frames in the original video. The solution is that we *approximate* the original frame by up sampling the down-sampled version. As a result, instead of reconstructing four original macroblocks, we reconstruct one output macroblock and up sample it. Considering the fact that we can utilize the optimized DCT domain down-sampling methods plus that the up-sampling operation is an easily optimizable operation either in hardware DSP design or MMX implementation, the computational saving is significant. In addition, all macroblock in an I-picture can take advantage of the method. For inter-frames, similar algorithms can be used if enough number of the involved macroblocks is intra macroblocks. The approximation by up sampling is based on the rationale that, to generate a down-sampled inter-frame, the quality loss is negligible when performing motion compensation based on down-sampled reference frames.

3.2 Macroblock-Aware Transcoding Algorithm

To explain in more detail how the macroblock-aware transcoding algorithm works, this section goes through a down-sampling-by-two operation of MPEG [5] video. Figure 2 shows the process for each type of pictures as well as different combinations of macroblocks in an MPEG video down-sampling application. The relative size of blocks and frames is also illustrated to reflect the change before and after the processes. For I-picture, a DCT domain down sampling methods can be used to generate a down sampled version of the I-picture. A pixel domain version has to be generated afterwards as a reference for the future P- or B-pictures in the group of picture (see the motion compensation arrow). Subsequently, an up sampling is performed to approximate the pixel domain version in the original size. The original size picture is needed for the reconstruction of the future P- or B-picture (see the reconstruction arrow). In this figure, the blue boxes correspond to the frame buffer for the original size video, and red boxes to the frame buffer for the down-sampled version.

For P- and B-picture, a mode-decision module decides whether the output macroblock is coded as inter or intra macroblock. If there are one intra block and three forward-predicted ones in the input macroblocks, the mode-decision module may decide to code the output block as a forward-predicted one. In general, assuming k macroblocks are involved in generating one output macroblock, the mode decision module decides that if at least m out of k macroblocks are intra macroblocks, the output macroblock will be coded as intra. Otherwise, the output macroblock is coded as inter. If the output macroblock is coded as intra, we further define a number n, where $n \in (m, k)$. Depending on the actual number of input intra macroblocks, the selection of n will decide what procedure the transcoder takes to generate the output intra macroblock. In summary, we have the following four scenarios that the macroblock-aware transcoder will handle differently.

- Case 1: the output MB is an intra MB, and m to n input macroblocks are intra
- Case 2: the output MB is an intra MB, and n to $(k-1)$ input macroblocks are intra
- Case 3: the output MB is an intra MB, and all k input macroblocks are intra
- Case 4: the output MB is an inter MB.

Using the down-sample-by-two operation as an example, four macroblocks are involved in generating one output macroblock, i.e. *k=4*. If we further select *(m, n)=(2, 3)*, we can use IIFF-I, IIIF-I, IIII-I and IFFF-F to map the aforementioned four scenarios. These symbols represent different compositions of the four original macroblocks. For example, if there are one intra block and three forward-predicted ones in the input macroblocks, a mode-decision module may decide to code the output block as a forward-predicted one, hence the symbol IFFF-F. The other three symbols can be interpolated similarly. Note that each symbol does not literately represent one exact combination of input blocks, other possible combinations can be solved based on the understanding of these four scenarios.

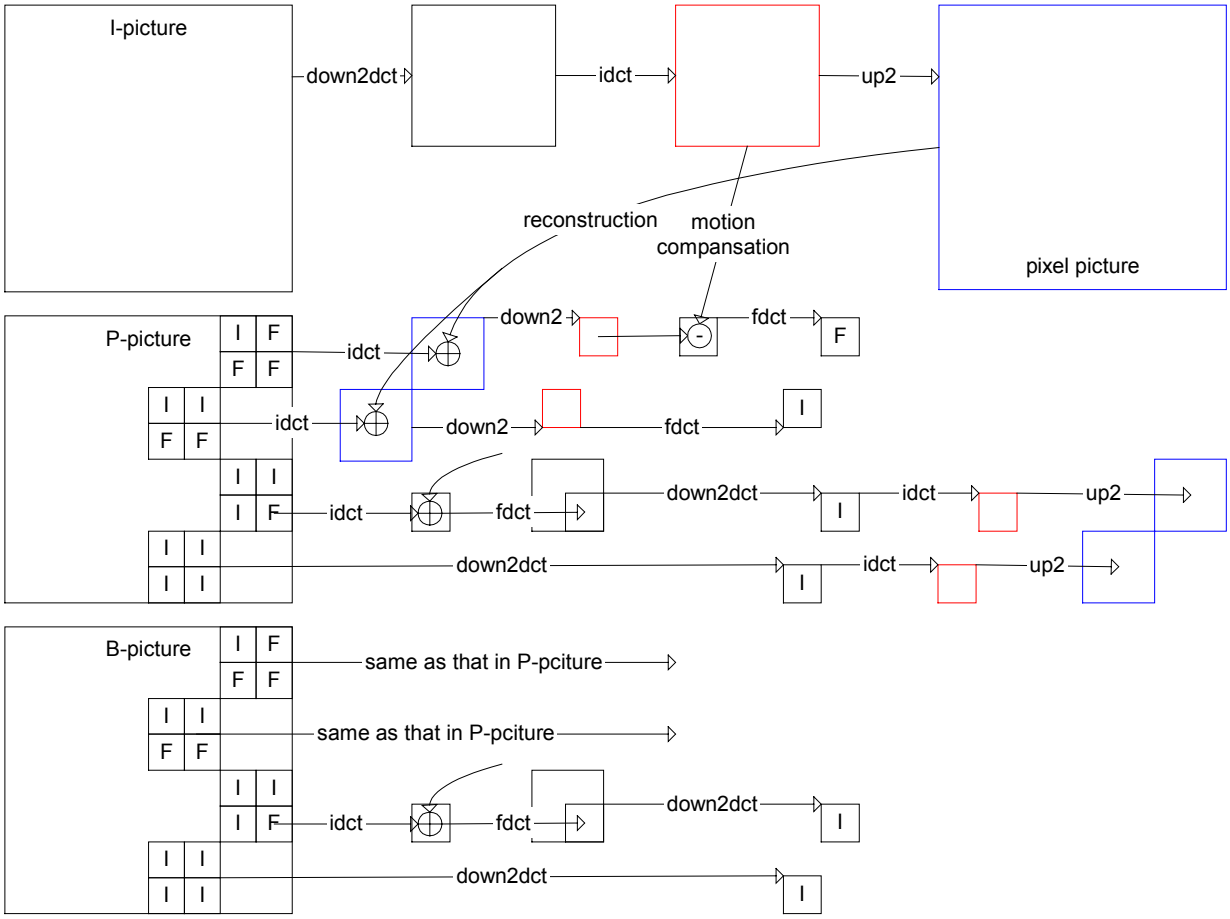

Figure 2 Data flow of the transcoder

To begin with, let us consider Case 4, the IFFF-F case. If the output macroblock is decided to be a predicted one, the original involved macroblocks are reconstructed regardless of whether the macroblocks are intra or non-intra. A spatial domain down sampling is performed on the reconstructed macroblock to generate a pixel domain output macroblock. Subsequently, the residual is generated based on the motion vector obtained from the original MPEG video. In this sequence of procedures, the reconstructed macroblocks along with subsequently down sampled version are kept as reference for the future P- or B-pictures. This scenario has no difference than a regular decoding and re-encoding scheme except that the motion information is reused (from MV Generator) instead of obtained through a costly motion estimation process.

If the output macroblock is decided to be an intra block, three different schemes are depicted in Figure 1. Case 3 represents the case when all four input macroblocks are intra (IIII-I). In this case, the DCT domain down sampling methods can be directly employed. Subsequently, the output macroblock is inverse discrete cosine transformed (IDCTed) to the pixel domain and the up-sampled version is also obtained for possible use as reference frames for future pictures.

If only one of the four original is a non-intra macroblock (e.g., the IIIF-I scenario for Case 2), only the non-intra macroblock is reconstructed and a DCT version is generated through the forward discrete cosine transform (FDCT) process (this process can also be performed through a DCT domain motion compensation [4]). Then the four DCT macroblocks are used to construct one DCT macroblock using a DCT domain downscaling process [2, 3] (see the down2DCT arrow). Subsequently, the output macroblock is IDCTed to the pixel domain and up-sampled version is also obtained for possible use as reference frames for future pictures.

For Case 1, we use IIFF-I as an example. In this case, we reconstruct every involved macroblock regardless of whether it is intra or not. Subsequently, a spatial domain down sampling is performed on the reconstructed macroblocks to generate a pixel domain output macroblock. The macroblock is then FDCTed to generate the output intra macroblock. In this case, we cannot take advantage of the DCT domain down-sampling method. The reason is that it is more costly to reconstruct the intra DCT version of the two input inter macroblocks.

For B-pictures, the four different scenarios described for P-pictures apply as well except that the reconstruction of some macroblocks may take reference from past and/or future pictures. In addition, as shown in the case of IIIF-I and IIII-I, we do not have to go through the processes of IDCT and up sampling for obtaining the pixel frame with original size. This is due to the fact that B-pictures are not used as reference frames.

The process for down sampling by a factor of three or four can be easily derived from the above. In down-sample-by-four case, k is equal to 16. If we select m as 9, we found that it is optimal to select n as 12. Similar scenarios from Case 1 to 4 can be identified and handled accordingly by the macroblock-aware transcoder. Note that since the down-sample-by-four operation brings in many more possible combinations of macroblocks, further optimization can be done for some cases depending on the locations of the involved intra macroblocks.

3.3 Transcoding system

Figure 3 illustrates the data flow of the transcoding process. From the input buffer, variable length code (VLC) decoder parses the input video stream. Motion vectors are passed to motion compensation module and MV Generator for generating new motion vectors for the downscaled version. The DCT data is sent to inverse quantizer and subsequently, a mode decision module decides whether the output macroblock is coded as inter or intra.

- If it is Case 2 and 3, the input DCT data is sent to the DCT domain down-sampling module to generate down sampled DCT data. For the macroblocks directly generated by DCT domain down sampling, IDCT is performed and the result is saved in Frame Buffer B. Furthermore, a up sampling is performed to generate a reconstructed version with original size that is saved in Frame Buffer A.

- Otherwise, the macroblock needs to be reconstructed. The DCT data is IDCTed and motion compensated if the macroblock is an inter macroblocks. The result is saved in Frame Buffer A. Therefore, the data in Frame Buffer A is the reconstructed macroblock with original size. It is then down sampled and put into Frame Buffer B.

The data in Frame Buffer A is used to reconstruct future frames in the original video. The data in Frame Buffer B is used to generate the new residual based on the new motion vector. The residual is subsequently FDCTed and sent along with the DCT data generated directly by DCT domain down-sampling module to the forward quantizer. The rate control module controls the step size of the quantizer in order to achieve specified output bit rate. Finally, VLC encoder generates the output binary bit stream.

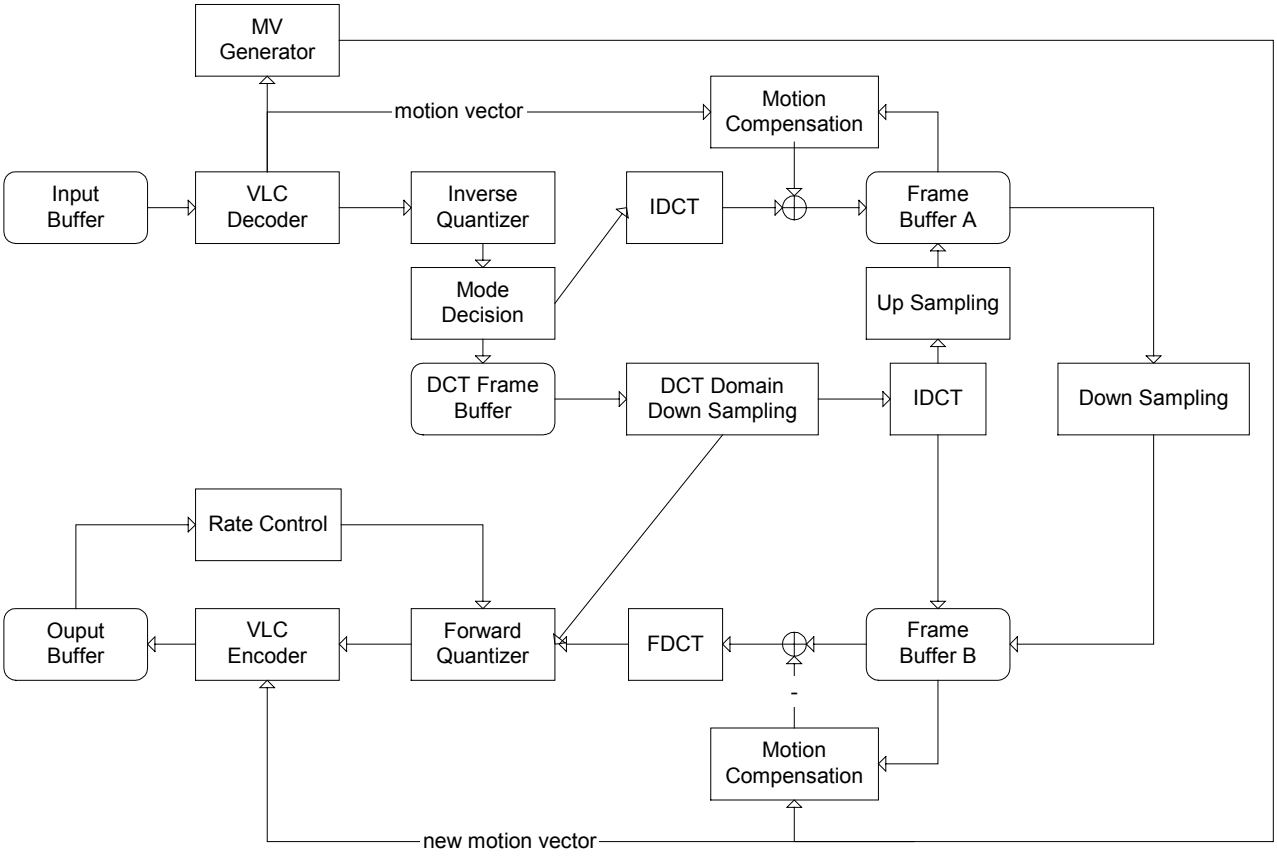

Figure 3 Processing flow of the transcoder

MV Generator is responsible to generate new motion vector based on the motion vectors from the original stream. The new motion vector could be a scaled version of the some weighted average of the original ones. A previous work [1] presents some analysis regarding the optimal way to generating the new motion vector.

Note that the original frames are reconstructed in Frame Buffer A, either by a full decoding – IDCT and then motion compensation, or by up sampling the down sampled DCT version. The up-sampling process is only performed when the original pixel domain frame is needed as a reference. Similarly, the output video constructed out of the output buffer is either encoded from the down sampled pixel domain version through motion compensation using new motion vectors generated from MV Generator, or directly from the down-sampled DCT version. The mode-decision module dictates how

the data in the frame buffers is generated. In general, Frame buffer A stores the reconstructed frame of the original size (blue boxes in Figure 1), Frame buffer B stores the down sampled version (red boxes in Figure 1). In practice, Frame Buffer A and B can share the same memory space.

4. Performance Analysis & Testing Result

Tests are conducted on Intel Pentium III platform with 733MHz CPU. Two test streams are used. The football test stream is of size 704x480 and is coded at 6 Mbps. It has 150 frames. The flower-garden test stream contains 450 frames with the size of 704x480 and is coded at 4Mpbs. Both streams are coded using IBBPBB… structure in groups of pictures that have 15 frames. In the experiments, down-sample-by-two and down-sample-by-four operations also reduce the bit rate by a factor of 4 and 16 respectively.

4.1. Testing results for MMX and SSE optimization

For MMX and SSE optimization, we test the computation performance and PSNR differences after optimizations on different modules are implemented. Figure 4 shows the computational time for both down-sample-by-two operation and down-sample-by-four operation on two test streams. Note that the processing time keeps dropping when more functional procedures are optimized. Specifically, "O1" to "O3" indicate three levels of compiler optimization. "F" indicates optimization on FDCT. "Q" indicates optimization on quantization modules. "M" represents optimization on motion compensation. "I" represents optimization on IDCT module. "D" represents optimization on down sampling module. "A" represents optimization on one sub-module (addblock) of motion compensation. "P" represents optimization on the prediction module. In each case, real time transcoding is achieved when all the optimizations go in place.

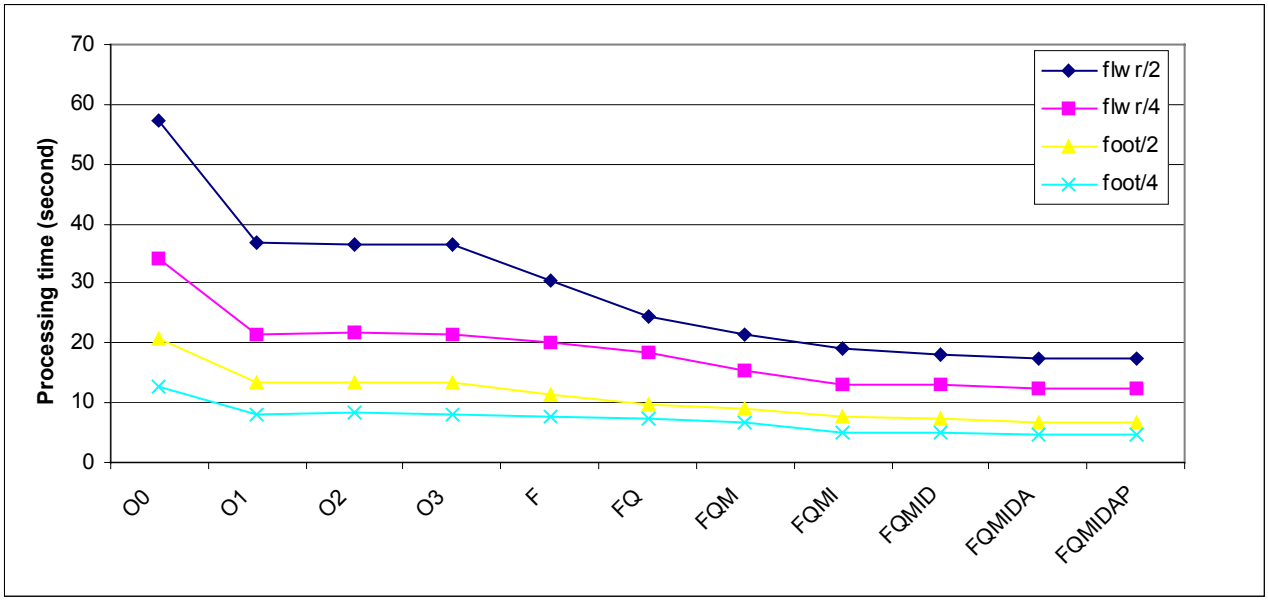

Figure 4 Computation time for different optimizations

Figure 5 and Figure 6 compare the PSNR differences after a cascade of optimizations are added into the transcoder. In this test, IDCT operation on the I-pictures is not MMX optimized. Note that the

optimization on IDCT modules introduces the majority of the distortion. Overall, the optimizations introduce negligible quality loss.

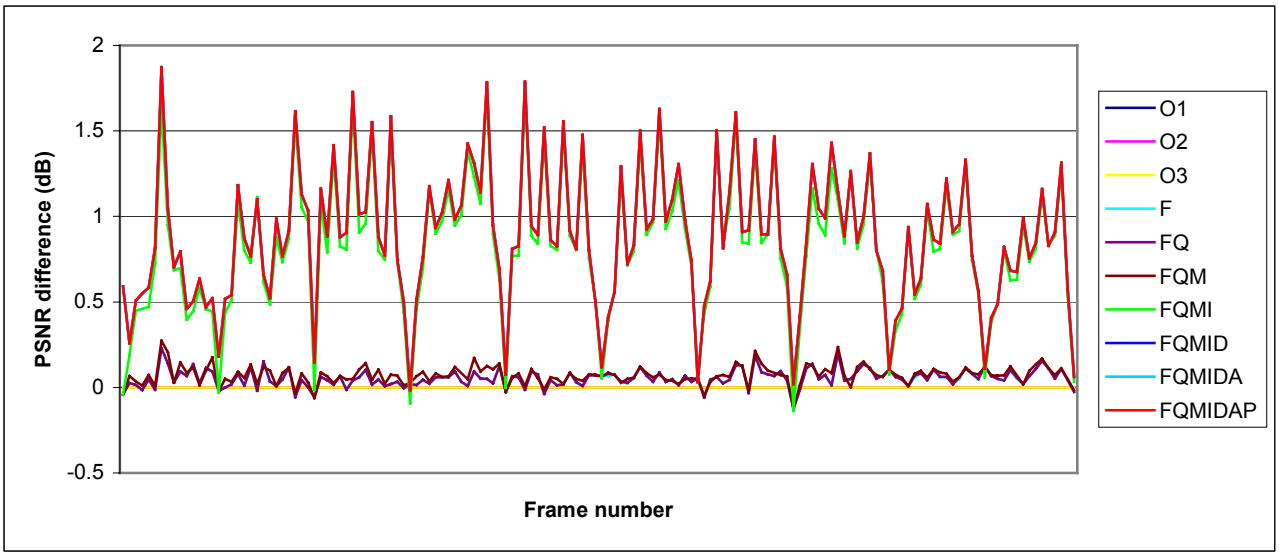

Figure 5 PSNR difference for different levels of optimization (football sequence)

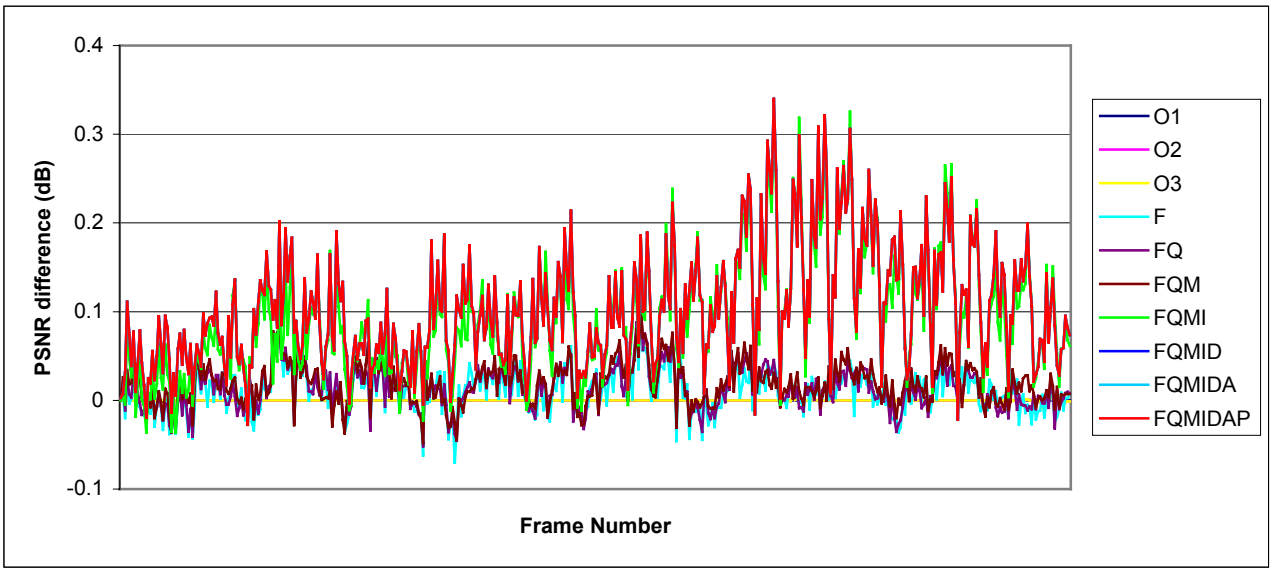

Figure 6 PSNR difference for different levels of optimization (flower-garden sequence)

4.2. Analysis and testing result for macroblock-aware transcoding

Primarily, the computational saving comes from the elimination of a number of IDCT processes. Here, one IDCT process represents the IDCT operations needed for the reconstruction of one macroblock. For 420 chrominance compression [5], six 8x8 IDCT operations are invoked for the reconstruction of one macroblock. For I-pictures, the number of IDCT operations required is $1/N^2$ times originally required, where N is the down-sampling factor. For the down-sample-by-two operation, it originally needs four IDCT operations for one output macroblock, now only one IDCT is needed. Considering an additional DCT domain down sampling process, two IDCT is saved, which is about 50% saving. The same applies to the IIII-I case in P- and B-pictures. For the IIIF-F case in P- and B-pictures, two IDCTs

are required instead of four. Considering an additional DCT domain down sampling process, one IDCT is saved, which is about 25% saving. Additional saving comes from the elimination of the reconstruction of pixel domain version of B-pictures for IIIF-I and IIII-I cases. Overall, the computational saving introduced by the proposed method is about 40% for the down-sample-by-two case. The saving is more significant for down-sample-by-three (N=3) and down-sample-by-four (N=4) cases.

The computational saving is video sequence dependent. For example, if a P-picture in the original stream contains mostly predicted macroblocks, the advantages brought by the smart handling of IIII-I or IIIF-I case are limited. However, since this algorithm is targeting at transcoding high bit-rate and high-resolution video to low bit-rate and low-resolution video, statistics show that many macroblocks are coded in intra mode in the original high bit-rate and high-resolution video. Figure 7 shows a P-picture in a test MPEG stream (coded at 6Mbps), the green boxes indicate that all four macroblocks are intra macroblocks (the IIII-I case). Figure 8 shows the percentage of different cases for each frame in the original MPEG test stream.

Figure 7 IIII-I case in a P-picture of the football sequence

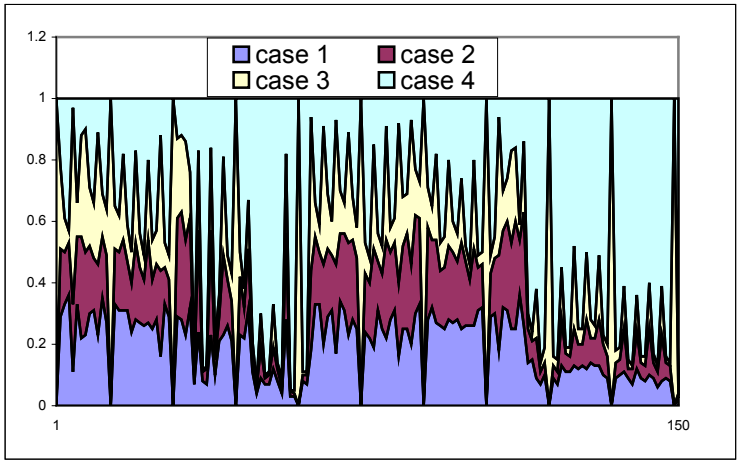

Figure 8 Stacked percentage of different cases for each frame of the football sequence

To test the efficiency of the macroblock-aware transcoder, we perform a down-sample-by-two operation on the football test stream. The transcoder generates an output video that is coded at 1.5 Mbps with the size of 352x240. Two PSNR curves are shown in Figure 9. The old PSNR curve is obtained by using picture-level reconstruction in the transcoding; the new PSNR curve is obtained by using macroblock-aware transcoding proposed in this paper. Both PSNR are computed against the frames generated by decoding the original video and down sampling the result by two. A frame-by-frame quality comparison is given in Figure 10.

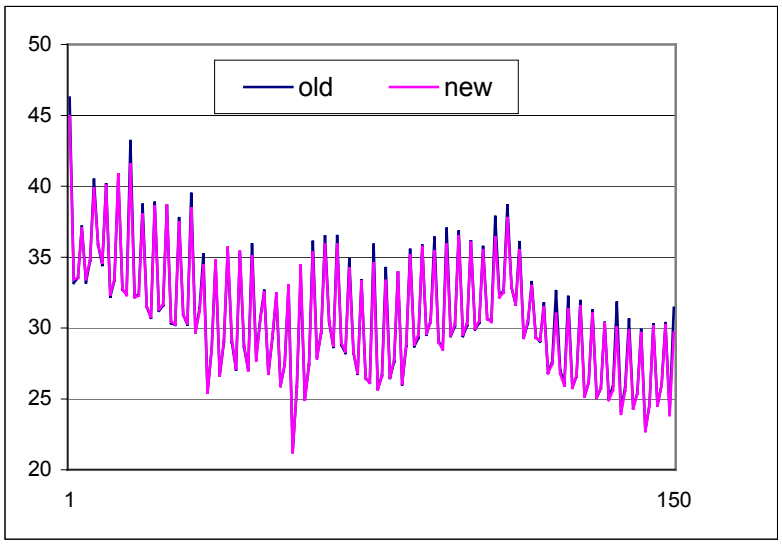

Figure 9 PSNR of each frame generated by the old transcoder and the new transcoder

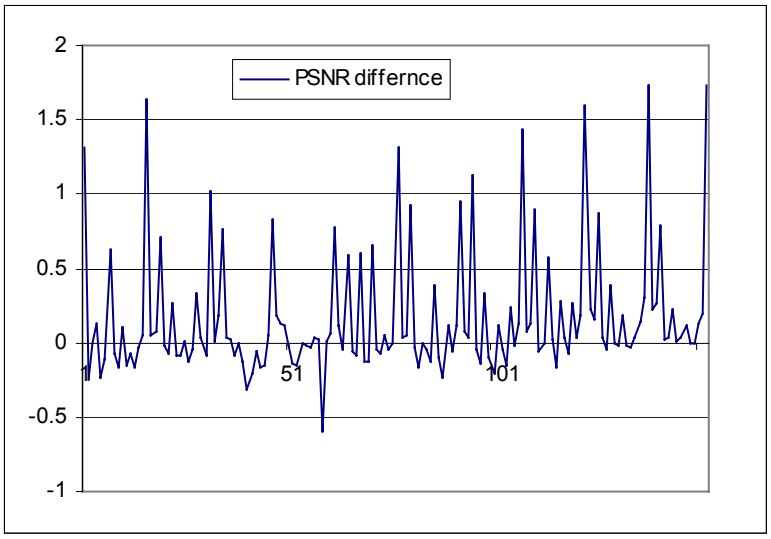

Figure 10 PSNR difference

The experiment indicates that the macroblock-aware transcoding renders similar quality output. Note that the PSNR drop for I-pictures is relatively high. This is exclusively due to the approximation nature of the DCT domain down sampling algorithm that is used in the implementation.

5. Conclusion

An architectural optimization to a video transcoding system using MMX and SSE is presented in this paper. The transcoding system performs a spatial resolution reduction on a compressed video by reusing the original motion information. However, the transcoding speed is limited by the necessity of the reconstruction of original frames. Therefore, this paper further presents an optimized algorithm that avoids the reconstruction of original frames as much as possible. Moreover, the proposed algorithm takes advantage of fast DCT domain down sampling methods as much as possible without the reconstruction of intra DCT version of original frames. As a result, additional 40% of saving is achieved with negligible loss of quality. The algorithm applies to other types of compressed video as well, as long as motion-compensated motion estimation and DCT-based frequency domain compression techniques are used in the compression.

References

[1] B. Shen, I. Sethi and V. Bhaskaran, "Adaptive motion-vector resampling for compressed video downscaling," *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 9, no. 6, pp. 926-936, Sept. 1999.

[2] N. Merhav and V. Bhaskaran, "A transform domain approach to spatial domain image," *HP Laboratories Technical Report*, HPL-94-116, Dec. 1994.

[3] B. Natarajan and V. Bhaskaran, "A fast approximate algorithm for scaling down digital images in the DCT domain," *Proc. IEEE int. Conf. On Image Processing (ICIP*), Washington DC. Oct. 1995.

[4] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT-domain inverse motion compensation," *Proc. ICASSP'96*, Atlanta, GA, pp. IV.2307-2310, May 1996.

[5] J. L. Mitchell, W. B. Pannebaker, C. E. Fogg and D. J. LeGall, MPEG Video Compression Standard, Chapman& Hall, 1995.

[6] http://download.sourceforge.net/mjpeg/mjpeg_beta_1a.tar.gz.

[7] http://download.sourceforge.net/mjpeg/mjpegtools-1.3b3.tar.gz.

[8] Aaron Holtzman, http://dara.notbsd.org/~aholtzma/ac3/mpeg2dec.php.

[9] Intel, Intel Architecture MMX Technology Developer's Manual.