Implementation of Real-Time MPEG-4 FGS Encoder

Yen-Kuang Chen and Wen-Hsiao Peng

Microprocessor Research Labs, Intel Corporation
yen-kuang.chen@intel.com

Abstract. While computers become faster than they used to be, software implementation of the latest codec in real time is still a challenging topic. This paper presents our techniques in optimizing MPEG-4 Fine Granularity Scalability (FGS) video encoders. First, zigzag scan is a slow process in video encoding and decoding. While state-of-the-art processors utilize hardware prefetchers to reduce memory latency, non-sequential addresses in zigzag scan may destroy the trackability of hardware prefetching. The problem is even more serious in MPEG-4 FGS where we need multiple scans in bit-plane coding. More than 30% of CPU time is for bit-plane encoding in an MPEG-4 FGS encoder (including base layer and enhancement layer). In this work, we rearrange the layout of the image structure so as to have zigzag scan in sequential memory locations. After the rearrangement, there are prefetch reads and we see 80% speed-up in bit-plane encoding. Second, variable length encoder (VLC) incurs a huge number of unpredictable conditional branches. While modern processors can execute tens of instructions in the pipeline, a mis-predicted branch will decrease the efficiency of the pipeline. The problem is severer in MPEG-4 FGS where we need multiple bit-plane VLC's. More than half of the CPU time for MPEG-4 FGS enhancement layer encoder is on bit-plane VLC's. In this work, we also design a bit-plane VLC algorithm, which has fewer unpredictable branches. The new design reduces mis-predicted branches by 2.4x. After these changes, overall speed-up in our MPEG-4 FGS software encoder is 1.4x without any assembly and MMX technology optimization.

1. Introduction

Due to bandwidth constraints of communication channels, video data are often compressed prior to transmission on a communication channel. Encoding and decoding video signals are computationally intensive processes. While the state-of-the-art CPU offers us more computations per second than it used to be, (1) memory latency is not matching, and (2) branch mis-prediction penalty is larger than it used to be. In this work, we first rearrange the storage layout of the video/image data before compression so as to utilize the hardware data prefetch, which takes some pressure off longer memory latency. Second, we re-design the run-length encoder so as to reduce the branch mis-prediction rate.

The structure of an MPEG-4 FGS encoder [2, 3] is shown in Figure 1. Similar to many image or video encoding processes, which have discrete cosine transform

(DCT), quantization, zigzag scan, and variable length coding VLC, the FGS enhancement layer encoding process has the following:

1. DCT
2. (Optional shifting operation of selective enhancement & frequency weighting [2])
3. Zigzag scan
4. Multiple bit-plane extraction and VLC

One uniqueness of the enhancement layer encoder is bit-plane extraction and VLC of the bit-planes, as shown in Figure 2.

One critical performance factor, which appears in FGS codec, is the large amount of randomized zigzag data addressing.

Zig-zag scan maps the 8x8 matrix to a 1x64 vector which groups low frequency coefficients in top of vector. Figure 3(a) shows a normal storage of a block in raster scan order, where we put horizontal data together first. When we need to scan the data in the zigzag scan order (as shown in Figure 3(b)), we will access data in the order at

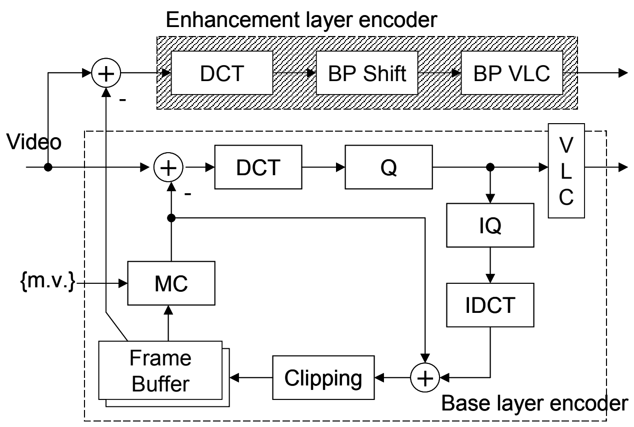


Figure 1: The structure of an MPEG-4 FGS encoder.

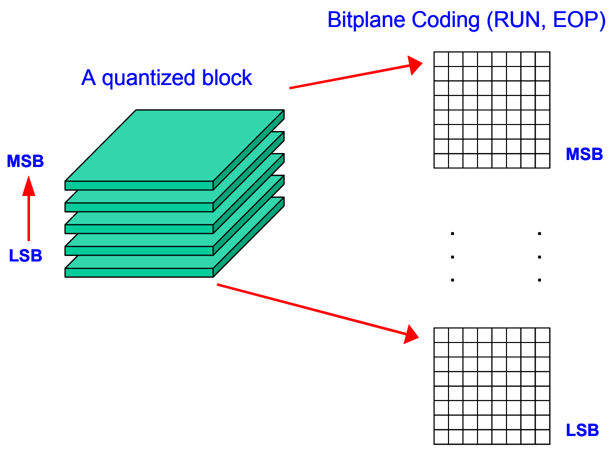


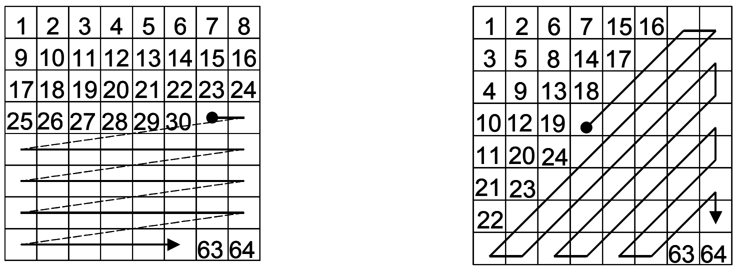
Figure 2: Bit-plane coding in MPEG-4 FGS. A quantized block is coded using bit-plane coding from MSB to LSB.

location 1, 2, 9, 17, 10, 3, 4, 11, 18, 25, etc.

Figure 4 shows a straightforward implementation of the bit-plane extraction and encoding. In this implementation, for each bit-plane encoding, we need a zigzag scan, which causes performance problems.

The first problem is that we need extra steps to generate the addresses because the addresses are not sequential. If we scan the data only once, one extra step is acceptable. On the other hand, encoding MPEG-4 FGS enhancement layers takes multiple scans. In this case, multiple extra steps become a bigger overhead.

Another problem of storing data in raster scan while processing data in zigzag scan is that random accesses of data make it harder for hardware to prefetch data. Intel Pentium 4 processors introduced hardware prefetchers, which load the data into the second-level cache before the application needs the data [1]. More precisely, the prefetchers check whether there are linear data access patterns and retrieve data if a linear access pattern is detected. If data are prefetched into the cache, then it takes less time for applications to get the data. In this case, larger memory latency is not as critical as it used to be. Unfortunately, hardware prefetchers less work effectively if the application processes data in zigzag scan order while data are stored in raster scan order. In this work, to make hardware prefetchers work efficiently, we rearrange the data.



(a) (b)

Figure 3: (a) Normal raster scan order, where horizontal data are stored together. (b) Zigzag scan pattern.

```
for (different bit-plane) {
    bit_mask = 1<<bit-plane;
    for (different blocks) {
        while (position<=last_position) {
            if (block_buffer[zigzag[position]] & bit_mask) {
                eop=(position==last_position)?1:0;
                Coding <run, eop> Symbol
                run=0;
            }
            else
                run++;
            position++;
        }
    }
}
```

Figure 4: A straightforward implementation of bit-plane extraction and encoding. This implementation needs to translate the array address for each bit-plane extraction.

As shown in Table 2, more than half of the CPU time was in bit-plane extraction during the encoding.

Another critical performance factor is the VLC. Normally, VLC takes about 10% of the CPU time in the entire encoding process (wherein a fast motion estimation algorithm is used). Nonetheless, FGS have multiple bit-plane VLC's and takes more than half of the CPU time in the enhancement layer encoding process.

In run length encoding, we first locate the non-zero coefficients. Then, we count the 0's between non-zero data, and encode the number of 0's with the non-zero data. Currently available instructions do not permit efficient determination of this run length. The value of each coefficient must be tested individually with a conditional operation that results in a mispredicted branch each time a non-zero data is extracted. Branch miss predictions will decrease the pipeline efficiency and slow down Pentium 4. While a normal video encoder has one VLC, an MPEG-4 FGS encoder has multiple bit-plane VLC's. In this case, a VLC algorithm with fewer conditional branches becomes important.

Figure 5 shows the original approach of the bit-plane VLC (the algorithm used by the reference codec). This implementation encodes RUN's between non-zero bits. For each non-zero bits, we need a branch. That is, bit-plane extraction will introduce conditional branches. Currently, 11.2% of the conditional branches in the VLC are mispredicted. Branch misprediction penalty would be huge in future architectures with deeper pipeline. Thus, it is important to design a bit-plane extraction and VLC of runs without conditional branches.

2. Zigzag Scan In-Order Approach

We should perform only one zigzag scan for all bit-plane extraction and VLC's. Because the outputs of DCT are usually in raster-scan fashion, the inputs of VLC are normally in raster-scan manner as well. In order to access coefficients in the zigzag scan order, there are some address calculations required. Currently, in the reference

```
while(AddressCounter<last_position) {
    if(FrameBuffer[AddressCounter] & And_op ) {
        //Coding the symbol here, Huffman Coding
        _encode_fgs_symbol(RUN, YUVsign[AddressCounter]);
        YUVsign[AddressCounter]=2;                  //The sign is transmitted
        RUN=0;
    }
    else {
        RUN++;
    }
    AddressCounter++;
}
```

Figure 5: Original algorithm with conditional branches. The number of mispredicted branches in this algorithm depends on the number of ones in the bit-plane. For example, if there are 6 ones in the bit-plane, then we have 7 mispredicted conditional branches.

software, zigzag scan is combined with bit-plane VLC. Thus, for a 6-bit-plane FGS encoding, we need 6 zigzag scans and 6-bit-plane VLC's. Figure 6(a) shows the situation described above. After DCT, we store DCT coefficients in the raster scan order at point [a]. Then, we convert them into 32-bit integers at point [b]. When we extract bit-planes for VLC, we work on the raster-scanned buffer. Thus, we need a zigzag scan for each bit-plane extraction.

To reduce extra zigzag scans, our implementation pre-scans the data in zigzag scan order and stores the data in sequential order. So, only one address calculation for each coefficient is needed. Figure 7 shows the pseudo code of the proposed implementation, which works the same as that in. While this approach can significantly reduce the number of calculations needed, additional memory buffer is needed. Hence, we combine the pre-zigzag scan into other buffer-copy operations (between [a] and [b] in Figure 6(a)). Figure 6(b) shows our implementation that has only one internal buffer copy.

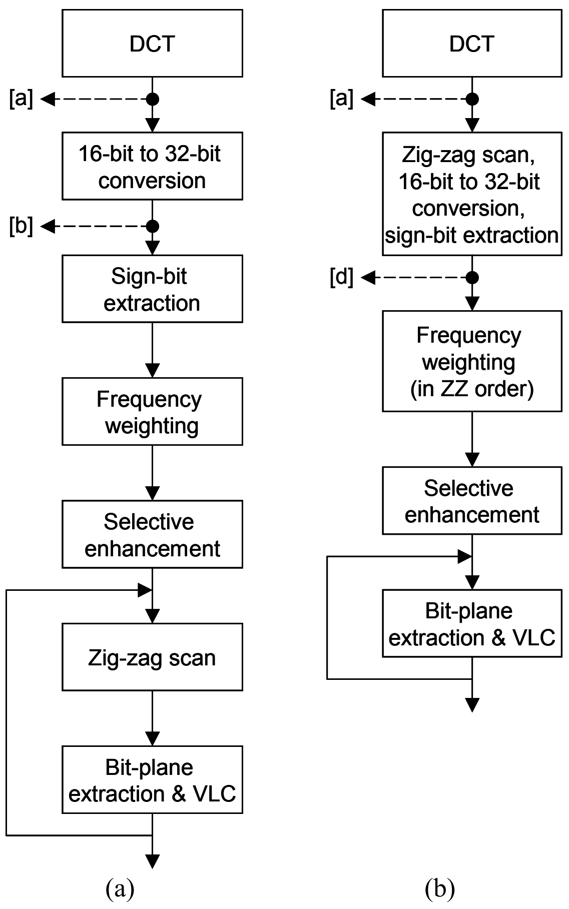

DCT
[a]
16-bit to 32-bit conversion
[b]
Sign-bit extraction
Frequency weighting
Selective enhancement
Zig-zag scan
Bit-plane extraction & VLC
(a)
DCT
[a]
Zig-zag scan, 16-bit to 32-bit conversion, sign-bit extraction
[d]
Frequency weighting (in ZZ order)
Selective enhancement
Bit-plane extraction & VLC
(b)

Figure 6: FGS enhancement layer encoding processes.

3. Brnchless VLC

To reduce conditional branches, we should perform logic operations. There are two main stages in the new approach. Instead of counting the runs of zero bits, we first extract the positions of non-zero bits into a buffer. Then, we encode the runs using the difference of the positions in the buffer.

During the first stage, we transform conditional branches into logic operations. As shown in Figure 8, we won't write the correct values into *ones_position* nor increase the *ones_counter* unless the Boolean variable *oneorzero* is equal to one. In this case, the condition branches only happen at the end of the both loops.

4. Results

Besides reducing the address calculations in zigzag scan, making memory accesses predictable improves the performance as well. This is because current hardware prefetchers have limited capabilities. Originally, we zigzag scan DCT blocks when we extract the bit planes. When the data stored in the raster-scan order, zigzag scan of a DCT block will access memory randomly. After we re-organize our memory layout, accesses to the DCT coefficients are sequential. Thus, it is easier for hardware to predict our memory access pattern and prefetch data effectively. Using special hardware monitors, we found that the number of "reads non-prefetch" drops by 50%.[1] Table 1 shows that our FGS encoder is 40% faster after the new memory organization. Table 2 shows the CPU time breakdown of each module before & after our changes (on 1.6GHz Pentium 4 systems). The bit-plane extraction module is 80% faster. In this case, our code is about 1.4x faster than the MPEG-4 reference software without any assembly and MMX technology optimization.

Moreover, the new VLC algorithm greatly reduces mispredicted conditional branches by 2.4x in the VLC module, as shown in Table 3. Thus, the overall algorithm has 3.9% misprediction rate instead of 7.7%. While currently we see only 1.6% speed-up, in a CPU with 50-pipeline stages, the new algorithm will have 15% speed-up. This is because the number of mispredicted conditional branches drops by more than 50%.

5. Summary

In this paper, we analyze the performance bottleneck of the software implementation of MPEG-4 FGS encoders and present four basic optimization techniques:

[1] “Reads non-prefetch" counts read requests to the main memory, but the subset that are not due to prefetches. Seeing 50% fewer counts indicates that the hardware prefetchers have launched these other cases. That is, our goal of getting hardware prefetchers to trigger more was achieved.

```
for (different blocks)
        for (different positions) {
         block_buffer'[position] = block_buffer[zig_zag[position]];
        }
for (different bit-plane)
{
        bit_mask = 1<<bit-plane;
        for (different blocks) {
                while (position<=last_position) {
                        if ( block_buffer'[position] & bit_mask ) {
                                eop=(position==last_position)?1:0;
                                Coding <run, eop> Symbol
                                run=0;
                        }
                        else
                                run++;
                        position++;
                }
        }
}
```

Figure 7: The proposed implementation of bit-plane VLC. The zigzag scan is performed first before the bit-plane extraction.

```
// First stage: bit-plane extraction
ones_position[0]=-1;
ones_counter=1;
for(i=0; i <end_counter; i++) {
      ones_position[ones_counter]=i;                          // conditional
      oneorzero=(FrameBuffer[i] & And_op)||0;           // logic
      ones_counter+=oneorzero;                                   // conditional
}
ones_position[ones_counter]=i;

// Second stage: VLC the bit-plane
for(i=1;i<ones_counter;i++) {
      RUN=ones_position[i]-ones_position[i-1]-1;
      AddressCounter+=RUN;
      _encode_fgs_symbol(RUN, YUVsign[AddressCounter]);
      YUVsign[AddressCounter]=2;                             //The sign is
transmitted

      AddressCounter++;
}
RUN=ones_position[i]-ones_position[i-1]-1;
AddressCounter+=RUN;
_encode_fgs_symbol(RUN, YUVsign[AddressCounter]);
```

Figure 8: New algorithm with fewer conditional branches. We use logic operations to put ones' positions into an array. Then, we just encode the runs without conditional branches. The number of mispredicted branches in the new algorithm is independent of the number of ones in the bit-plane. There are two mispredicted branches in the new algorithm.

1. Take redundant operations out of the loops.
2. Combine multiple slow, data-dependent operations together in one step.
3. Make memory accesses predictable.
4. Reduce conditional branches

Mostly, putting data in zigzag scan order makes memory accesses predictable and improves performance. This is because we can utilize hardware prefetchers to reduce memory latency.

Our technique to have the zigzag scan immediately after the DCT not only can be applicable to encoders, but also to decoders. In decoders, we should not perform the zigzag scan until we accumulate all bit-planes together.

6. References

[1] Intel Corp., Intel Pentium 4 Processor Optimization Reference Manual, Order Number: 248966-04, 1999-2001.

[2] “Streaming video profile---Proposed Draft Amendment,” ISO/IEC JTC1/SC29/WG11, N3315, Mar. 2000.

[3] W. Li, “Overview of Fine Granularity Scalability in MPEG-4 Video Standard,” IEEE Trans. on CSVT, vol. 11, no. 3, Mar. 2001, pp. 301 –317.

[4] H. Jiang, “Using Frequency Weighting in Fine-Granularity-Scalability Bit-plane Coding for Natural Video”, ISO/IEC JTC1/SC29/WG11, M5489, 1999.

Table 1: Speed of the FGS encoder (enhancement layer only) of CIF images at 1.6GHz Pentium 4.

Sequences	Raster-scan	New memory	Speed up
Akiyo	30.9 (fps)	43.5 (fps)	1.41x
Foreman	23.4 (fps)	33.7 (fps)	1.44x

Table 2: CPU time of each module in the FGS encoder (enhancement layer only).

	Akiyo		Foreman	
	Raster-scan	New memory	Raster-scan	New memory
Bit-plane extract	54%	40%	59%	44%
Put bits	11%	19%	12%	20%
Encode symbols	5%	13%	6%	12%
Sign-bit extract	13%	12%	10%	10%
Forward DCT	8%	11%	6%	10%

Table 3: Performance comparisons of the FGS VLC algorithms.

		Old algorithm	New algorithm	Speedup
Overall Application	Cycles (millions)	9,923	9,772	1.02x
	Misprediction rate	7.7%	3.9%	
	Mispredicted cycles	12.9%	6.1%	
VLC Module Only	Mispred. branches (millions)	51.5	21.3	
	Misprediction rate	11.2%	5.2%	
Future Projection	M Cycles (50 stage pipeline)	11,851	10,317	1.15x