H.264 DECODER OPTIMIZATION EXPLOITING SIMD INSTRUCTIONS

Juyup Lee, Sungkun Moon, Wonyong Sung

Seoul National University

ABSTRACT

The inherent nature of digital signal processing (DSP) and multimedia applications has been targeted for exploiting single-instruction, multiple-data (SIMD) extensions to instruction architectures for the most of the modern microprocessors. In particular, SIMD instructions can be the most effective in video applications which have simple operations on multiple and small data types, mostly 8-bit or 16-bit samples. In this paper, the newest video coding standard, H.264/AVC baseline profile decoder has been implemented and optimized exploiting Intel MMX technology to show the overall system speedup by the SIMD style coding.
The overall system speedup gain was 26% and the most of the gain came from the reduction of memory access time.

1. INTRODUCTION

SIMD extensions to general purpose processors (GPP) have been exploited pervasively to effectively support DSP functions and multimedia applications which usually have simple operations on multiple and small data types, recursively. In particular, Intel IA-32 instruction architecture has been evolved to maximize its MMX technology by extending to SSE, SSE2 and SSE3.

H.264/AVC is the newest video coding standard of ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. The main goal of the standard is to achieve enhanced compression performance and provision more network adaptiveness for efficient communications.

In this paper, H.264/AVC baseline profile decoder is implemented with Intel Pentium 4 processor with MMX extensions. SIMD instructions are aggressively applied to the kernels of the baseline subsystems including interpolation, inverse transform, and loop filtering. The speedup of each subsystem is measure in execution time and the overall system speedup was calculated to show Amdahl's law.

2. INTEL MMX TECHNOLOGY

MMX technology is the extension to Intel Architecture (IA-32) to exploit SIMD technique to maximize data-level parallelism (DLP) which is inherent in DSP applications. MMX instructions are targeting applications with small native data types, computation intensive and recursive operation on the data and algorithms with inherent data parallelism. By exploiting SIMD instructions "on registers" in the x87 floating point unit (FPU) it also satisfies backward compatibility with all application and operating system code. IA-32 architecture has been evolved to more actively include more data types and registers to support more aggressive instruction sets. Intel MMX technology has been extended to SSE, SSE2, and SSE3.

2.1 Intel IA-32 Architectural Support for MMX

IA-32 Intel Architecture introduced eight 64-bit registers (MM0–MM7) for MMX technology and Pentium 4 processor support additive eight 128-bit registers (XMM0-XMM7) to extend to Streaming SIMD Extensions (SSE), SSE2 and SSE3.

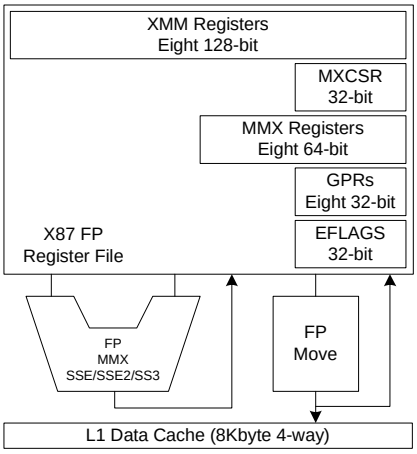

FIG. 1 IA-32 Architectural Support for MMX Extensions

FIG 1 shows the architectural support for MMX and the extensions. MMX technology exploits the existing x87 Floating Point Unit and adds 16 registers with the control registers to support more instructions and backward compatibility with existing programs.

2.1.1 MMX

MMX technology defines four data types, packed byte, packed word, packed doubleword, and quadword. Totally, 57 new instructions added to the IA-32. The instructions can be classified as add, subtract, multiply, compare, and shift. The packed and the saturating arithmetic instructions can be the most useful for video compression applications.

2.1.2 SSE/SSE2/SSE3

In brief, Intel SSE instructions provide the programmer with additive eight 128-bit registers (XMM0-XMM7) in which each register consists of four 32-bit single precision floating point numbers. SSE2 instruction sets added 128-bit double-precision floating point and 128-bit packed byte integers and provide some enhanced instructions such as cacheability-control, etc. SSE3 instructions support horizontal or asymmetric data movement whereas most SSE/SSE2 instructions support only vertical way.

MXCSR is the 32-bit register to provide control and status bits for SIMD floating-point operations and EFLAGS the 32-bit register to be used to store the results of compare operations [3].

3. H.264 DECODER OVERVIEW

H.264/AVC video coding standard has been introduced with significant enhancements in both video coding efficiency and flexibility over a variety of network domains. In video coding layer (VCL), some of important enhancements are the use of a small block-size (4x4) exact-match transform, adaptive in-loop deblocking filter and motion-prediction capability.

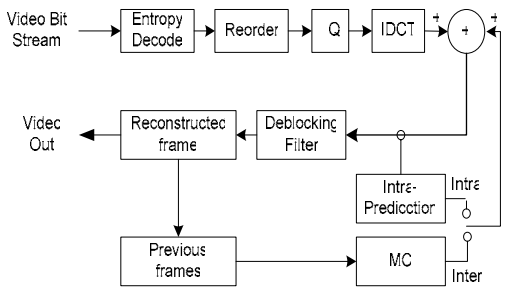

FIG. 2 H.264/AVC Decoder Block Diagram

As shown in FIG 2, after receiving the data from network abstraction layer (NAL), decoder entropy decodes and the inverse quantized and inverse transformed data is created and added to the predicted data from the previous frames depending upon the header information. Then, the original block can be reconstructed after deblocking filter [8].

3.1 H.264 Baseline Profile Decoder Features

H.264 defined three types of profiles. The baseline profile is the simplest and supports intra and inter-coding, and entropy coding with context-adaptive variable-length coding (CAVLC). The distribution of time complexity amongst major subsystems was analyzed and loop filtering, interpolation, inverse transform and bit stream parsing and entropy decoding are the order of the averaged time consuming parts [7].

In this paper, interpolation, inverse transform, and loop filter are implemented using SIMD instructions for speedup.

4. SIMD OPTIMIZATION

In general, there are two approaches to implement SIMD techniques for recursive operations on multiple data in digital signal processing.

```
#ifndef MMX                                                    (a)
        for (result = 0, x = -2; x < 4; x++)
        result += imgY[y_pos+j][x_pos+i+x]*COEF[x+2];

#else                                                          (b)
        a = max(0,min(maxold_y,y_pos+j));
        b= x_pos+i-2;
_asm{
        movq xmm1, [edx+esi]
        pxor xmm2,xmm2
        punpcklbw xmm1,xmm2
        :xmm1=|dc|dc|x5|x4|x3|x2|x1|x0|
 }
__asm{
movdqa xmm2, COEF;xmm2=|0|0|1|-5|20|20|-5|1|
movdqu xmm3, ONES;xmm3=|1|1|1|1|1|1|1|1|

pmaddwd xmm1,xmm2
//xmm1=|0|x5+-5*x4|20*x3+20*x2|-5*x1+x0|
packssdw xmm1,xmm1
//xmm1=|dc|dc|dc|dc|0|x5+-5*x4|20*x3+20*x2|-5*x1+x0|
pmaddwd xmm1,xmm3
//xmm1=|dc|dc|x5+-5*x4|20*x3+20*x2+-5*x1+x0|
packssdw xmm1,xmm1
//xmm1=|dc|dc|dc|dc|dc|dc|x5+-5*x4|20*x3+20*x2+-5*x1+x0|
pmaddwd xmm1,xmm3
//xmm1=|dc|dc|dc| x5+-5*x4+20*x3+20*x2+-5*x1+x0|
packssdw xmm1,xmm1
//xmm1=|dc|dc|dc|dc|dc|dc|dc| x5+-5*x4+20*x3+20*x2+-5*x1+x0|
pextrw eax,xmm1,0
mov result,eax
}
```

FIG. 3 (a) Original C Code (b) SIMD Optimized Code Using Fine-Grained Method

4.1 Fine-Grained Method

In FIR filter operation, for fixed number of data the result data can be obtained by multiplication with the coefficients and added in parallel using SIMD instructions. FIG. 3 shows an example of 6-tap FIR filter used in the interpolation kernel. C-code implementation

is shown in (a) and SIMD version is shown in (b). In (a) single filtering operation takes six additions and six multiplications and six memory accesses for pixel data. In (b) SIMD instructions use only three MAD and three PACK operations and three memory accesses.

4.2. Coarse-Grained Method

Different from the completion of a series of operation at once in fine-grained approach multiple data takes each operation so the latency is higher. However, for small number of data the latter is simpler and more efficient lowering execution time. FIG. 4 shows an example of inverse transform implemented in C-code in (a) and SIMD instructions in (b). In (a), it is required that 4x4 additions, 4x2 shifts and 4x8 memory access but in (b) four additions, four memory accesses and two shifts.

In the example, four data moved to 128-bit XMM registers and processed simultaneously. In nested loops, coarse-grained method can be shown as a parallelization of outer loops.

```
#ifndef MMX                                                    (a)
   for (j=0;j<BLOCK_SIZE;j++){
      for (i=0;i<BLOCK_SIZE;i++){
         m5[i]=img->cof[i0][j0][i][j];
      }
         m6[0]=(m5[0]+m5[2]);
         m6[1]=(m5[0]-m5[2]);
         m6[2]=(m5[1]>>1)-m5[3];
         m6[3]=m5[1]+(m5[3]>>1);
                        :
}

#else                                                          (b)
   mptr = & (img->cof[i0][j0][0][0] );
__asm{
   mov edx, mptr
   movdqu xmm1, [edx]
   packssdw xmm1,xmm1// read m50] from memory to xmm1
}
                        : // read m5[1],m5[6] from memory
__asm{
   movdqu xmm4, [edx +48]
   packssdw xmm4,xmm4// read m5[3] from memory
}
__asm{
   movq xmm5,xmm1
   psubw xmm1,xmm3 //m6[1]=(m5[0]-m5[2]);
   paddw xmm3,xmm5 //m6[0]=(m5[0]+m5[2]);
   movq xmm5, xmm2
   psraw xmm2,1
   psubw xmm2,xmm4 //m6[2]=(m5[1]>>1)-m5[3]
   psraw xmm4,1
   paddw xmm4,xmm5 //m6[3]=m5[1]+(m5[3]>>1)
                        :
}
```

FIG. 4 (a) Original C code (b) SIMD Optimized Code Using Coarse-Grained Method

4.2 Optimization Methods for the Kernels

Above mentioned three subsystem kernels are optimized using SIMD instructions with either of two implementation methods. Each function of the kernel is summarized in TABLE 1 with the fraction information in the system.

4.2.1 Interpolation

H.264/AVC codec supports integer, half-pixel and quarter-pixel interpolation. Except integer case, fine-grained method is exploited for 6-tap FIR filtering in half-pixel interpolation for each pixel. In quarter-pixel interpolation coarse-grained method is applied to process with 4x4 block data simultaneously.

4.2.2 Inverse Transform

For 4x4 block integer matrix operation the standard recommends to minimize single multiplication and addition on the data with the coefficients and redundancy for the matrix operations. So, coarse-grained method is applied on 4x4 block and optimized.

4.2.3 In-Loop Filter

Even there are quite a few conditional loops in loop filtering algorithm, based on the fact that each pixel in the 4x4 block has the same filtering strength coarse-grained approach is fit well. After the completion of different modes of operations the results can be aggregated to form resulting values.

Primary routine	Functional Description	Fraction
Interpolation	Used in motion compensation to calculate half-pixel or quarter-pixel values	27.5%
Inverse Transform	Calculation of inverse DCT followed by inverse quantization for 4x4 integer block	7.0%
Loop Filter	Filtering at the boundary of each block to reduce blocking artifacts	5.3%
Memory	Memory allocation and free operation	10.0%
VLD	Support CAVLC	12.2%
I/O	Read and Write from the disc	4.5%
Misc.	-	33.5%

TABLE 1 Functional Description and fraction of H.264/AVC Baseline Profile Decoder Kernels

5. EXPERMENTAL RESULTS

To implement, JM-7.2 version is used as a source code and the blocks of interest are modified using in-line assembly to program SIMD instructions. VTune analyzer is used for performance profiling and the execution time is monitored as a metric.

5.1 VTune Analyzer Support

VTune analyzer is a performance estimation tool provided by Intel for IA-32 and Itanium architecture. It can be very useful in analyzing system performance and

optimizing time consuming sections by powerful functions such as call graph, execution time measurement and events (instruction retirement, cache miss).

In the experiments, total execution time and the execution time of each kernel are measured using VTune. Also, the execution time is searched in the measure of arithmetic and memory R/W operations.

5.2 Speedup Measurement

Each SIMD optimized block is executed and the execution time is measured. With the speedup ratio and the fraction of optimization, the overall system speedup is calculated by applying Amdahl's law.

5.3 Analysis

The performance enhancement by SIMD instructions is not very significant compared to previous papers since CPI for Pentium 4 is 0.5 for many operations. Though the actual operations are increased the total execution time is reduced in that the number of memory accesses is much less.

Routine	Estimated Speedup	Actual Speedup
Interpolation	2.00	1.93
Inverse Transform	2.46	5.08
Loop Filter	1.4	1.91
Total	1.24	1.26

TABLE 2 System Speedup Results

Routine	Memory R/W		Arithmetic Operation	
	ratio	speedup	ratio	speedup
Interpolation	87.2%	2.18	12.8%	1.08
Inverse Transform	76.8%	5.09	23.2%	4.40
Loop Filter	80.0%	1.56	20.0%	5.30

TABLE 3 System Speedup Results in Separation of Memory and Arithmetic Operations

Table 2 shows the speedup of each subsystem by executing each algorithm separately. Then, applying Amdahl's law the overall system speedup is calculated with respect to the fraction data in Table 1. The performance gain was 26% and inverse transform showed the most significant enhancement by the effective use of unpack instructions. In Table 3, Memory R/W and arithmetic operations are separated and to show in which parts the speedup is achieved. Interpolation showed the most significant enhancement by minimizing memory access time.

The results clearly show that memory access is the most dominating and SIMD techniques are exploited to minimize the total memory access time.

6. CONCLUSION

In this paper, the newest video coding standard, H.264/AVC baseline profile decoder is implemented exploiting Intel SIMD instructions. The speedup is achieved mainly by minimizing memory access time during the execution of the kernel subsystems. In arithmetic operations, the speedup is not significant at all. The overall system speedup was 26% by applying Amdahl's law.

There is still enough headroom for more optimization by aggressively applying memory pre-fetch operation and caching unused registers to further minimize expensive memory access.

7. REFERENCES

[1] A. Peleg and U. Weiser, "MMX Technology Extension to Intel Architecture," IEEE Micro, 16(4):42-50, Aug. 1996.

[2] A. Peleg, S. Wilkie and U. Weiser, "Intel MMX for Multimedia PCs," Comm. ACM, vol. 40, no. I, pp. 25-38, Jan. 1997.

[3] IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture, http://developer.intel.com/design/pentium-4/manuals/

[4] IA-32 Intel Architecture Optimization Reference Manual, http://developer.intel.com/design/pentium4/manuals/

[5] D. Talla, L. K. John and D. Burger, "Bottlenecks in Multimedia Processing with SIMD Style Extensions amd Architectural Enhancements," IEEE Trans. Computers, vol. 52, no. 8, Aug. 2003.

[6] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Lutha, "Overview of the H.264/AVC Video Coding Standard," IEEE Trans. Circuits Syst. Video Technol, vol. 13, no. 7, pp. 560-576, July 2003.

[7] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis," IEEE Trans. Circuits Syst. Video Technol, vol. 13, no. 7, pp. 704-716, July 2003.

[8] I. E. G Richardson, *H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia*, Wiley, 2003.

[9] V. Lappalainen, A. Hallapuro, and T. D. Hamalainen, "Optimization of Emerging H.26L Video Encoder," IEEE Workshop on Signal Processing Systems, pp. 26-28 Sep 2001.