VLIW DSP VS. SUPERSCALAR IMPLEMENTATION OF A BASELINE H.263 VIDEO ENCODER

Serene Banerjee, Hamid R. Sheikh, Lizy K. John, Brian L. Evans, and Alan C. Bovik

Dept. of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712-1084 USA
{serene,sheikh,ljohn,bevans,bovik}@ece.utexas.edu

ABSTRACT

A Very Long Instruction Word (VLIW) processor and a superscalar processor can execute multiple instructions simultaneously. A VLIW processor depends on the compiler and programmer to find the parallelism in the instructions, whereas a superscaler processor determines the parallelism at runtime. This paper compares TI TMS320C6700 VLIW digital signal processor (DSP) and SimpleScalar superscalar implementations of a baseline H.263 video encoder in C. With level two C compiler optimization, a one-way issue superscalar processor is 7.5 times faster than the VLIW DSP for the same processor clock speed. The superscalar speedup from one-way to four-way issue is 2.88:1, and from four-way to 256-way issue is 2.43:1. To reduce the execution time on the C6700, we write assembly routines for sum-of-absolute-difference, interpolation, and reconstruction, and place frequently used code and data into on-chip memory. We use TI's discrete cosine transform assembly routines. The hand optimized VLIW DSP implementation is 61x faster than the C version compiled with level two optimization. Most of the improvement was due to the efficient placement of data and programs in memory. The hand optimized VLIW implementation is 14% faster than a 256-way superscalar implementation without hand optimizations.

1. INTRODUCTION

Two factors limit the use of real-time video communications: network bandwidth and processing resources. The ITU-T H.263 standard [1, 2, 3] for video communication over wireless and wireline networks has high computational complexity. An H.263 encoder includes the H.263 decoder (minus the variable length decoding) in a feedback path. The H.263 encoder is roughly three times more complex than an H.263 decoder.

In the H.263 encoder, the most computational complex operation is motion estimation, even when using efficient search algorithms. The bottleneck is in the sum-of-absolute differences (SAD) calculations. SAD provides a measure of the closeness between a 16×16 macroblock in the current frame and a 16×16 macroblock in the previous frame. Other computational complex operations are image interpolation, image reconstruction, and forward discrete cosine transform (DCT).

In this paper, we evaluate the performance of the C source code for a research H.263 codec developed at the University of British Columbia (UBC) [4] on two processor architectures. The first architecture is a very long instruction word (VLIW) digital signal processor (DSP) represented by the TI TMS320C6701 [5, 6]. The second is an out-of-order superscalar architecture represented by the SimpleScalar simulator [7].

We evaluate the performance of the superscalar processor vs. the maximum number of simultaneous instructions executed. We compare the performance of the C source code on both processor architectures. For the VLIW DSP implementation, we demonstrate that manual placement of frequently used data and code into on-chip memory provides a 29x speedup, whereas hand coding five of the most computational complex routines gives a 4x speedup. Our hand-coded SAD, clipping, interpolation, and fill data C6700 assembly routines and our SimpleScalar settings are available from

http://www.ece.utexas.edu/~sheikh/h263/

2. BACKGROUND

2.1. TMS320C6700 VLIW DSP Family

The TMS320C6700 is a floating-point subfamily within the C6000 VLIW DSP family. The C6000 family has 32-bit data words and 256-bit instructions. The 256-bit instruction consists of eight 32-bit instructions that may be executed at the same time. The C6000 depends on the compiler to find the parallelism in the code.

The C6000 family has two parallel 32-bit data paths. Each data path has 16 32-bit registers, and can only communicate one 32-bit word to the other data path per instruction cycle. Each data path has four 32-bit RISC units: one adder, one 16×16 multiplier, one shifter, and one load/store unit. Each 32-bit RISC unit has a throughput of one cycle, but its result is delayed by one cycle for 16×16 multiplication, zero cycles for logical and arithmetic operations, four cycles for load/store instructions, and five cycles for branch instructions. Every instruction may be conditionally executed. Conditional execution avoids the delay of a branch instruction and avoids interrupts from being disabled because a branch is in the pipeline.

The C6000 family has one bank of program memory and multiple banks of data memory. Each data memory bank is single-ported and two bytes wide, and has one cycle data access throughput. Accessing one data memory bank twice in the same cycle results in a pipeline stall and increased cycle counts. The C6701 has 64 kbytes of program memory and 64 kbytes of data memory. The program memory bank has 2k of 256-bit instruction words, and may be operated as an instruction cache. Each of the 16 data memory banks has 2k of 16-bit halfwords.

On the C6700, the pipeline is 11–17 stages, depending on the instruction. The clock speed varies from 100 to 200 MHz. Power dissipation is typically 1.5–2.0 W, and the transistor count is less than 0.5 million. All simulation results for the C6700 were obtained by running the code on a 100-MHz C6701 Evaluation Module board. The C6700 simulator does not report pipeline stalls or memory bank conflicts, so running the code on the board gives the true performance.

2.2. SimpleScalar Superscalar Processor

The SimpleScalar has a superscalar architecture derived from the MIPS-IV. The architecture prefetches at least four times the number of instructions to be executed and finds the parallelism between instructions at run time. All instructions have a fixed length of 64 bits and fixed format, which promotes fast instruction decoding. It supports out-of-order issue and execution, based on the Register Update Unit (RUU) [8]. The RUU uses a reorder buffer for register renaming and holds the results of pending instructions. The memory system employs a load/store queue to support speculative execution. The results of a speculative store are placed in the store queue. If the address of all previous stores are known, then loads are dispatched to the memory. The six pipeline stages are *issue*, *dispatch*, *fetch*, *commit*, *writeback*, and *load/store queue refresh*.

For the SimpleScalar simulator, the configuration of the processor core, memory hierarchy and branch predictor can be specified by using command-line arguments. The clock speed of typical superscalar processors is as high as 1 GHz. Power dissipation is a few tens of Watts, and the transistor count may vary from a few tens to a hundreds of millions.

2.3. UBC's H.263 Video Encoder

The H.263 Version 2 (H.263+) video encoder [1] contains 23,000 lines (720 kbytes) of C code. It was written for desktop PC applications and increases memory usage for faster execution. It incorporates the baseline H.263 encoder with many optional H.263+ modes. Our goal was to optimize the baseline H.263 encoder only for embedded applications, where the amount of available memory is much lower. This encoder irregularly uses floating point variables and arithmetic. So, we choose the floating-point TMS320C6701 instead of the fixed-point TMS320C6201 as the VLIW DSP representative. In the performance evaluation, we encode sub-QCIF (128×96) frames. A related paper optimizes the MPEG-2 decoder on the TMS320C6000 [9].

3. VLIW DSP IMPLEMENTATION

The most demanding operations in a typical video encoder are motion estimation, motion compensation, discrete cosine transform, half-pixel interpolation, and reconstruction, as shown in Table 1. We hand optimize these routines to speed up the encoder. The entire H.263+ encoder, which needs 340 kB of program memory and 2.2 MB of data memory, does not fit in internal RAM on the C6701. We used memory placement and code optimizations to reduce cycle count.

3.1. Memory Placement

Placing the code and data in slow external RAM wastes many cycles, especially for frequently accessed data and code. We therefore performed manual placement of code and data into fast on-chip program and data memory respectively. Computationally intensive routines for motion estimation, interpolation, DCT, SAD, and reconstruction routines were placed in the on-chip program memory. Commonly used runtime functions from TI's libraries such as *memcpy*, *memcmp* and *memset* were linked into internal program memory. A total of 43 kB out of the 340 kB of needed program memory was placed in the internal program memory.

Of the 2.2 MB of data memory, the lookup table for quantization consumed 1.6 MB. The following were placed in internal data memory: (1) 16×16 macroblocks and their corresponding search area for the

Time (%)	Routine	-o2 Optimization	-o2 and Memory Optimization	-o2 and Code Optimization	All Optimizations
91.9%	Motion_Estimation	1356000	33400	325000	13000
1.8%	Motion_Compensation	26200	4200	6400	3400
1.6%	DCT (2-D DCT)	17200	670	6000	130
0.52%	Quantization	7700	660	3300	660
1.1%	Interpolation	16900	4200	2200	750
2.1%	Reconstruction	31000	4000	9100	2260
3.5%	Rest	52000	8300	31200	6270
100.0%	Entire encoder	1476000	51400	374000	24200

Table 1: Cycle counts (x1000) of H.263 routines (per frame) on the TMS320C6700 VLIW DSP.

Program	Usage	-o2 and Memory Optimization	-o2 and Code Optimization	All Optimizations
SAD	Motion Estimation	54 x	4 x	264 x
Clip_MB	Reconstruction	7.5 x	13 x	228 x
DCT	DCT of Frames	26 x	3 x	138 x
Interpolate	Bilinear Interpolation	4 x	8 x	22 x
FillMBData	Reconstruction	7 x	8 x	22 x
Encoder	—	29 x	4 x	61 x

Table 2: Speedup of hand optimized H.263 routines on the TMS320C6700 VLIW DSP per call.

Program	1-way	4-way	8-way	16-way	32-way	256-way	VLIW
SAD	12723	7281	6775	6017	6031	6422	290
DCT	56409	22770	16381	14069	13193	12885	384
Clip_MB	24545	9604	7023	6250	6438	6438	1173
FillMBData	12692	4693	3082	3425	3438	3438	8740
Interpolate	2011397	640799	425703	456605	456480	456480	750000
Encoder	196 M	68 M	59 M	30 M	30 M	28 M	24 M

Table 3: SimpleScalar (with increasing issue widths) and hand optimized VLIW DSP cycle counts for H.263 routines.

motion estimation routines; (2) 16×16 macroblocks for the DCT, quantization, coding and reconstruction routines; (3) local data of computationally intensive routines; and (4) the stack. These choices give an overall speedup of 29x over the unoptimized version alone.

3.2. Hand coding assembly language routines

Compiler intrinsics give little performance improvement for critical routines. To reduce cycle counts further, we write parallel assembly routines for SAD and Clip_MB and linear assembly routines for InterpolateImage and FillMBData. We use TI's DCT assembly routines.

SAD consists of two read, one subtract, one absolute value, and one accumulate operation per pixel of both blocks. We wrote a parallelized pipelined assembly kernel to compute the SAD between two 16×16 macroblocks that avoided memory bank conflicts. Of the 16 instructions in the fully unrolled inner loop, 11 instructions use five of the six arithmetic units and five instructions use all six arithmetic units. The SAD is computed in 290 cycles per macroblock in the worst case. Since the SAD routine aborts computation if the accumulated difference exceeds the current minimum for the search window, the hand-optimized SAD routine takes on average 110 cycles per macroblock for full-search motion estimation. The speedup over the level two C compiler optimization is 264 times.

The Clip_MB routine clips overflowing values to 255 and underflowing values to zero for every pixel in a macroblock. We unroll the loops and pipeline the comparisons. The cycle counts reduce to 1173 per macroblock, which is an improvement of 228 times over the compiler's best optimization.

The TI DCT routine runs 138 times faster than the C version with level two optimization only. However, the encoder uses 32-bit integers for coefficients, whereas TI's DCT routine uses 16-bit short integers. The conversion is an additional overhead.

We write FillMBData and Interpolate routines in linear assembly. Both these routines use packed data access to external memory. The speedup is about 22 times each over the corresponding C compiled version with level two optimization.

Table 1 summarizes improving cycle counts and Table 2 summarizes speedups of the optimized routines. Overall speedup for the encoder is 61 times over the version with C compiler optimizations enabled. The encoder requires 24 M cycles per frame, or equivalently, 4 sub-QCIF (128×96) frames/s on a 100-MHz C6701 EVM board. Additional improvement may be obtained by optimizing the variable length encoding. Our hand optimized VLIW routines can also be used in the MPEG standards.

4. SUPERSCALAR IMPLEMENTATION

For a fair comparison while comparing the individual H.263 routines, code and data were placed in the internal RAM for both the VLIW and the superscalar processor. We vary issue widths of the superscalar processor from 1-way to 256-way. The other parameters, i.e. the fetch and decode widths and the load/store queue and RUU sizes, were chosen to minimize the cycle counts. Typically, the fetch queue and the decode queue were twice the issue bandwidth and the load/store queue and the RUU size were four times the issue bandwidth. The 4 kbyte G-share branch predictor [10] was used, which gives good performance over a variety of applications. For memory latencies, the default values were chosen, which are equal to typical values in modern general-purpose processors.

For the SAD, DCT and Clip_MB routines on a 256-way issue superscalar processor, the best cycle counts were 6017, 12885 and 6250, respectively, which exceeds the cycle counts for corresponding VLIW DSP parallel assembly routines, as shown in Table 4. In Table 4, the superscalar cycle counts for FillMBData and Interpolate routines are lower than the cycle counts for the corresponding VLIW DSP linear assembly routines.

For the entire encoder, the cycle count was 28 M cycles, or equivalently, 3.5 sub-QCIF (128×96) frames/s on a 100 MHz processor. The cycle counts for the subroutines and the entire encoder for a 1-, 4-, 8-, 16-, 32-, and 256-issue processor are summarized in Table 3. With increasing issue widths, the speedup shows diminishing returns. Increased issue widths may sometimes lead to more missed speculations. This results in an increased the cycle counts, as observed in Table 3.

Routine	VLIW DSP cycles	Simple Scalar cycles	Ratio
SAD	290	6017	1:20.7
DCT	384	12885	1:33.5
Clip_MB	1173	6250	1:5.3
FillMBData	8740	3082	2.8:1
Interpolate	750000	425703	1.8:1
Encoder	24 M	28 M	1.2:1

Table 4: Comparison of 256-way issue superscalar and hand optimized VLIW DSP implementations.

Measure	-o2 Optimizations	-o2 and Memory Optimizations	-o2 and Code Optimizations	All Optimizations	Superscalar Performance
Cycles per frame	1476 M	51 M	374 M	24 M	28 M
Frames per second	0.14	2	0.54	4	3.5
Speedup	—	29	4	61	53

Table 5: Comparison of a VLIW DSP and 256-way issue Superscalar implementations at a clock speed of 100 MHz.

5. CONCLUSION

This paper evaluates performance of a baseline H.263 encoder in C on a VLIW DSP TMS320C6700 processor and on an out-of-order SimpleScalar superscalar processor to encode 128×96 frames. The H.263 encoder was originally written for PC applications and trades memory off for speed. We validate that the VLIW DSP and superscalar implementations generated identical H.263 bitstreams to the PC version of the encoder.

When using level two C compiler optimization only, we find that for the same processor clock speeds

1. one-way issue superscalar implementation is 7.5x faster than the VLIW DSP implementation,
2. four-way to one-way issue speedup is 2.88:1, and
3. 256-way to four-way issue speedup is 2.4:1.

In analyzing the performance of the VLIW DSP implementation, external memory access was the key bottleneck. The amount of on-chip memory was only a fraction of the total memory needed by the application. A less significant but still important problem was that the compiler cannot always find the full parallelism in C code to take advantage of the parallel functional units in a VLIW processor. Once we placed the data and code that were most often used into on-chip memory, we hand coded SAD, image interpolation, and image reconstruction routines on the C6700.

After all of the manual optimizations, the VLIW DSP implementation was 14% faster than the 256-way superscalar implementation with level two compiler optimization and with all data and code on-chip. This reflects a key difference in the architectures. On an out-of-order superscalar processor, the parallelism is detected and exploited at run time. On a VLIW DSP, the parallelism must be detected and exploited at compile time. Optimizing a VLIW DSP implementation requires careful manual placement of data and code into on-chip memory, which is not required on a superscalar processor. Optimizing both superscalar and VLIW DSP implementations generally requires hand coding the most frequently called routines.

6. REFERENCES

[1] G. Cote, B. Erol, M. Gallant, and F. Kossentini, "H.263+: Video coding at low bit rates," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 8, pp. 849–866, Nov. 1998.

[2] T. Gardos, "H.263+: The new ITU-T recommendation for video coding at low bit rates," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, vol. 6, pp. 3793–3796, May 1998.

[3] ITU Telecom Standardization Sector, "Video coding for low bit rate communication," *ITU-T Recommendation H.263 Version 2*, Jan. 1998.

[4] B. Erol, F. Kossentini, and H. Alnuweiri, "Implementation of a fast H.263+ encoder/decoder," in *Proc. IEEE Asilomar Conf. on Signals, Systems and Comp.*, vol. 1, pp. 462–466, Nov. 1998.

[5] *TMS320C6000 Programmer's Guide*. No. SPRU 198D, Texas Instruments, Inc., Mar. 2000.

[6] *TMS320C6000 CPU and Instruction Set*. No. SPRU 189E, Texas Instruments, Inc., Jan. 2000.

[7] T. Austin and D. Burger, "The SimpleScalar tool set, Version 2.0," in *Tech. Rep., Univ. of Wisconsin Comp. Sciences Dept.*, no. TR-1342, June 1997.

[8] G. S. Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers," *IEEE Trans. on Computers*, vol. 39, pp. 349–359, Mar. 1990.

[9] S. Sriram and C.-Y. Hung, "MPEG-2 video decoding on the TMS320C6x DSP architecture," in *Proc. IEEE Asilomar Conf. on Signals, Systems and Comp.*, vol. 2, pp. 1735–1739, Nov. 1998.

[10] S. Onder, X. Jun, and R. Gupta, "Caching and predicting branch sequences for improved fetch effectiveness," in *Proc. Int. Conf. on Parallel Architectures and Compilation Tech.*, vol. 1, pp. 294–302, Nov. 1999.