# OPTIMIZATION OF A BASELINE H.263 VIDEO ENCODER ON THE TMS320C6000

*Hamid R. Sheikh, Serene Banerjee, Brian L. Evans, and Alan C. Bovik*

Laboratory for Image and Video Engineering
Dept. of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX78712-1084 USA
{sheikh, serene, bevans, bovik}@ece.utexas.edu
http://signal.ece.utexas.edu

## ABSTRACT

Most implementations of H.263 available today target PC or workstation environments where there are plenty of memory and processing resources available. Such coders use high-level languages, typically C or C++, with extravagant memory usage to achieve speed of execution. In this paper we present our work on optimization of a baseline H.263 encoder for the Texas Instrument's (TI) TMS320C6000 platform. In particular, we present the techniques we used to optimize the University of British Columbia's (UBC) H.263 encoder implementation. The UBC's encoder is written in the C language for a desktop environment. Our optimizations resulted in an over all speedup of 61 times over the unoptimized version. Our implementation uses the TI's TMS320C6701 Evaluation Module.

**Keywords:** H.263 encoder, VLIW architecture, TMS320C6000, optimization techniques.

## 1. INTRODUCTION

Two factors limit the use of real-time video communications: network bandwidth and processing resources. H.263 is an ITU-T standard for video communication over wireless and wireline networks. In this paper, we attempt to minimize the processing resources required for the H.263 video encoder [1] written by the University of British Columbia [2, 3] on the TMS320C6000 [4]. We report figures from our tests on the Texas Instrument's TMS320C6701 (C6701) Evaluation Module running the encoder with full-search algorithm for motion vector estimation on a sub-QCIF resolution video sequence. The C6701 comes with 64 kilobytes of program and data memory each. Like other video compression algorithms based on motion compensation and block transform, the H.263 encoder has a high computational complexity. A related paper optimizes an MPEG II video decoder on the TMS320C6000 [5].

## 2. BACKGROUND

Computationally complex operations in a typical video encoder include block matching, motion compensation, discrete cosine transform (DCT) and interpolation. However, for embedded digital signal processors, such as the TMS320C6000 family of processors, a significant portion of the complexity of video encoding consists of accessing huge amounts of picture data in slow off-chip memory and performing arithmetic operations on them.

Table 1 (a) shows the cycle counts for various routines in the H.263's encoding step when the C code was compiled with -o2 option. The resulting code and data memory requirements demanded that the entire program and its associated data be placed in external SDRAM. With the encoder taking one and a half billion cycles per frame on a 100 MHz processor, we were far from achieving real-time performance at a reasonable frame rate.

Table 1 (a) shows that motion estimation is the most computationally intensive task in an H.263 encoder. It indeed is when considered in terms of the required number of arithmetic operations as well as the number of data fetches from slow off-chip RAM. The complexity of motion estimation lies predominantly in the computation of the sum of absolute difference (SAD): the measure of closeness of a macroblock in the current frame with another in the previous frame. The unoptimized encoder spends 67% of the encoding time in the computation of SAD.

The TMS320C6000 family is a VLIW RISC processor that has two parallel 32-bit data paths, each data path consisting of a 16 32-bit register file and four 32-bit computational units: an adder, a 16 x 16 multiplier, a shifter, and a load/store unit. The units in each data path can read from and write to the register files of the other data path, subject to certain constraints.

As we proceed through different stages of optimization, other operations such as motion compensated prediction, DCT, quantization, and interpolation begin to take a bigger share of complexity. The C6000 family of VLIW processors is well suited for such algorithms where functional parallelism inherent in these algorithms can be exploited to give major improvements in execution speed.

## 3. EFFICIENT IMPLEMENTATION

We performed optimization of the encoder in two steps: efficient use of on-chip data and program memory, and code optimizations of computationally intensive routines in C as well as in assembly language.

### 3.1. Memory Optimizations

Table 1 (b) shows the results of performing memory optimizations of the encoder only. The code for computationally intensive routines was placed in internal on-chip program memory. Some runtime support functions from the TI's library that were called repeatedly by the encoder were also linked into internal memory. For the motion estimation routine, the macroblock and the corresponding search window were copied into internal data memory before the routine was called. Local data of some complex routines was placed in internal RAM. Also, each macroblock was copied into internal data memory before the DCT, quantization, coding, transmission and reconstruction routines were called. The computational overhead of copying each macroblock in internal RAM was about 10% of the savings.

As apparent from Table 1 (b), tremendous improvements in execution speed can be achieved by intelligent placement of data and code in internal memory. The total speedup obtained by this step alone is about 29 times.

### 3.2. Code Optimizations

Several methods for optimizing code were employed, ranging from the use of compiler intrinsics to linear assembly to fully hand-coded parallel assembly of certain sections of the code. In particular, the routine for calculation of the SAD was hand-coded in assembly and is five times more efficient than the C version with fully unrolled inner loop. Our SAD routine aborts computation if the accumulated difference increases beyond the current minimum for the search window. In full search motion estimation, our SAD routine performs at an average of 110 cycles per macroblock. For fast-search motion estimation algorithm however, the routine performs at an average 270 cycles per macroblock.

The interpolation routine was written in linear assembly using packed fetches from and writes to off-chip memory, thereby giving an improvement of about 10 times over the C version. Packed fetches and writes were used in some other parts of the code as well. Assembly routine for the computation of DCT from TI's web site was incorporated into the encoder.

Table 1 (c) shows the cycle counts with the code optimizations only (with code and data in external RAM). With the code optimizations alone, an improvement by a factor of four is achieved. Table 1 (d) shows the cycle counts with code and memory optimizations. Both the optimizations combined give an improvement of 61 times. The figures reported are for full search motion estimation. Fast search algorithms reduce the cycle counts even further.

## 4. SIMULATION RESULTS

Figure 1 summarizes the improvements in speed obtained by the optimization techniques described above. A profile of the optimized code shows that most of the time is still spent in data access to external RAM. Simulations with fast motion vector search algorithm also suggest that the complexity bottleneck is accesses to frame data stored off-chip. Figure 2 shows the share of the computation of the SAD relative to the share of motion estimation for different levels of optimizations. The effect of slow off-chip memory accesses becomes the dominant bottleneck as the code becomes more efficient in performing computations.

## 5. CONCLUSION

We have demonstrated optimization of UBC's H.263 encoder that was originally written for a desktop environment. We have demonstrated that memory access to external memory are a significant bottleneck in the implementation of real-time embedded video systems that have large memory requirements. Compiler's ability to understand generic C code written for desktops is also limited. However, we conclude that for low-resource embedded video applications, it is best to start with a careful and implementation-dependent design rather than to start with a generic high-level language code originally written for desktop environments.

## 6. REFERENCES


[1] ITU Telecom Standardization Sector, "Video Coding for Low Bit Rate Communication," *Draft ITU-T Recommendation H.263 Version 2,* Sept. 1997.

[2] B. Erol and 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.

[3] G. Cote and B. Erol and M. Galland 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.

[4] "TMS320C6000 CPU and Instruction Set," in *Digital Signal Processing Solutions 2000, Texas Instruments, Inc.*, http://www.ti.com, no. SPRU189E, Jan. 2000.

[5] 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.

| Operation | (a) Before Optimizations | (b) Memory Optimizations only | (c) Code Optimizations only | (d) Code and Memory Optimizations |
|---|---|---|---|---|
| Motion Estimation | 1356000 | 33400 | 325000 | 13000 |
| Motion Compensation | 26200 | 4200 | 6400 | 3400 |
| DCT | 17200 | 670 | 6000 | 130 |
| Quantization | 7700 | 660 | 3300 | 660 |
| Interpolation | 16900 | 4200 | 2200 | 750 |
| Reconstruction | 31000 | 4000 | 9100 | 2260 |
| Others | 52000 | 8300 | 31200 | 6270 |
| Total | 1476000 | 51400 | 374000 | 24200 |

*Table 1*: **Cycle counts (x1000) for various H.263 routines on the TMS320C6701 Evaluation Module.**

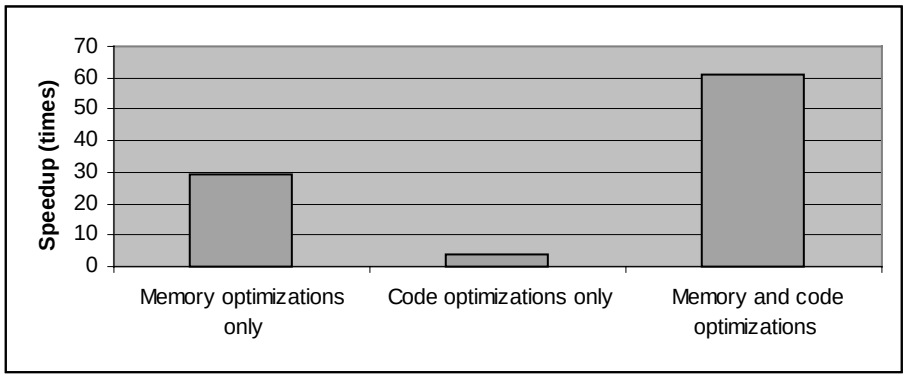


*Figure 1*: **Speedup of the H.263 encoder with different optimizations**

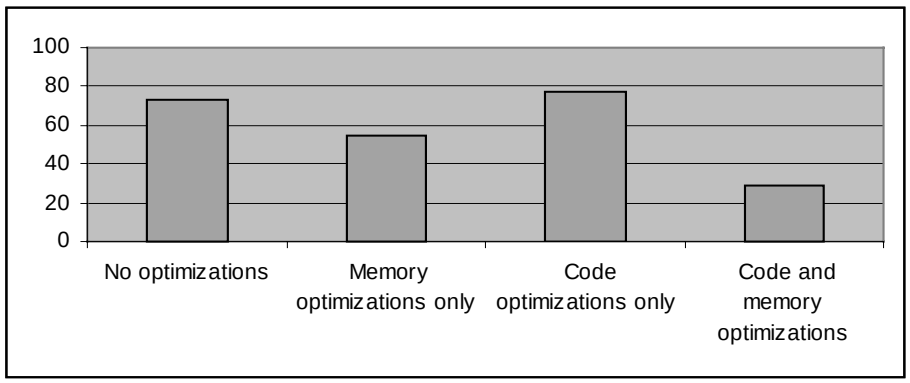


*Figure 2*: **Percent share of computation of the SAD between two macroblocks in full-search motion estimation algorithm at different stages of optimizations**