Implementations of H.264/AVC Baseline Decoder on Different Digital Signal Processors

Yair Moshe and Nimrod Peleg
Signal and Image Processing Laboratory (SIPL)
Department of Electrical Engineering, Technion – Israel Institute of Technology
Technion City, Haifa, 32000, Israel
E-mails: yair@siglab.technion.ac.il, nimrod@siglab.technion.ac.il

Abstract - *H.264/AVC is the newest video coding standard developed by the joint effort of ITU-T VCEG and ISO/IEC MPEG. This standard achieves a significant improvement in coding efficiency relative to former standards at the cost of increased complexity, thus gaining a lot of attention by industry, but creating a big challenge for efficient hardware and software implementations. In this paper three implementations of an H.264/AVC baseline decoder using different leading high-end Digitals Signal Processors are described. The implementation has been done, using the JVT reference software, by undergraduate seniors under the supervision of the authors, in the course of one year. This study is beneficial in several aspects. First, the complexity of an H.264/AVC decoder is measured and analyzed using "real world" hardware and software. Second, the difficulties that programmers could face when implementing an H.264/AVC decoder scheme are examined. And lastly, different Digital Signal Processors are examined under a very demanding algorithm.*

Keywords – *H.264, MPEG-4 AVC, video coding, Digital Signal Processor implementation*

1. INTRODUCTION

H.264/MPEG-4 AVC [1] is the latest international video coding standard. VLSI technology has advanced significantly since the development of previous standards (e.g. MPEG1/2, H.261/3), resulting in a significant reduction in the implementation cost of some coding tools that were excluded from those standards. H.264/AVC includes some of these tools as well as a variety of innovations. Hence, its complexity is higher, especially on the encoder side. Even on the decoder side, its complexity is estimated to be two to three times higher than an H.263 decoder, for the same bitrate [5]. DSPs (Digital Signal Processors) are specialized microprocessors designed to efficiently perform digital signal processing algorithms. They are efficient in terms of size, power consumption and price. In addition, DSPs offer great flexibility since they can be easily programmed for different applications. This cost-efficiency makes DSP programming an important and challenging subject. Therefore, it is useful to examine H.264/AVC performance on various DSP platforms.

Several studies examined H.264/AVC decoder performance on a GPP (General Purpose Processor). Fig. 1 shows a time breakdown of a non-optimized decoder as reported in [4]-[7]. These profiling results were all measured for GPPs with different implementations, test sequences, and coding parameters. In spite of this fact, examining Fig. 1 is interesting, in the sense of getting a rough estimate of the expected results using a specific configuration, and acknowledging the fact that great variability in running time is possible depending on different parameters.

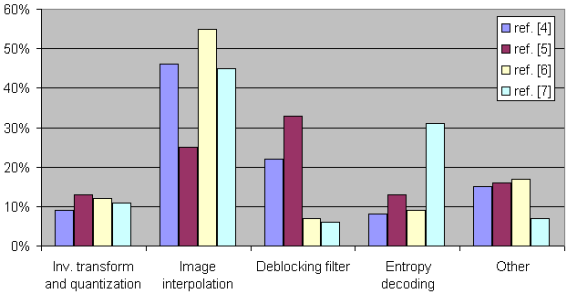

Fig. 1. Time breakdown of an H.264/AVC decoder on General Purpose Processors as reported in [4]-[7]

In [5] a way is given to compute a lower time-complexity bound of an H.264/AVC decoder for a given hardware. This lower bound provides an insight on the decoder complexity, but could be two to six times lower than experimental results, even for a highly optimized decoder. Therefore, it is much more practical to estimate H.264/AVC time-complexity by establishing "real life" experimental benchmarks on specific hardware platforms.

This paper describes a comparative work that examines H.264/AVC decoder performance on three different DSPs. This work has been performed in our lab by three senior undergraduate student groups, under the supervision of the authors. A baseline profile version (after removing some irrelevant code) of the reference software from JVT is used [3]. The purpose of this paper is to examine the special difficulties that programmers could face when using an H.264/AVC reference software decoder for DSP implementation, and suggest

optional solutions. Moreover, it enables a comparison between three different modern DSP architectures.

It should be noted that the work has been performed during the students' studies and so is limited in time and scope. This affected the amount of optimizations performed on the code but does not prevent us from gaining some important insights.

The paper is organized as follows. Section 2 gives a brief overview of the H.264/AVC video coding standard. Section 3 discusses modern DSPs and briefly describes the three DSPs under examination. Results, as well as some insights gained during the work, are given in section 4. Finally, section 5 concludes our work.

2. H.264/AVC OVERVIEW

An overview of H.264/AVC and its development stages can be found in [2], as well as in other sources. In this section, the main innovations of H.264/AVC in comparison to previous standards, which affect its complexity, are briefly described.

• Transform and quantization: Instead of the traditional 8x8 DCT, which is a real transform and hence needs real number arithmetic, H.264/AVC uses an integer approximation to the DCT with a smaller block size of 4x4 pixels. In addition, no quantization tables are needed and quantization is done according to a logarithmically-controlled quantization parameter.

• Inter multi-frame prediction: In previous video coding standards, motion compensation was performed using 16x16 macroblocks only. H.264/AVC allows also the use of smaller block sizes – 16x8, 8x16, 8x8, 4x8, 8x4 and 4x4. In addition, motion compensation is done in ¼-pixel accuracy and can be based on up to five reference frames.

• Intra prediction: Performed in the spatial domain, by referring to neighboring pixels of previously-coded blocks that are to the left and/or above the block to be predicted. Two block sizes for intra prediction are supported – 4x4 with nine prediction modes, and 16x16 with four prediction modes.

• Deblocking filter: A well known problem of block-based coding is the production of visible block artifacts due to block edge discontinuities, especially at low bit rates. H.264/AVC defines an adaptive in-loop deblocking filter which reduces blockiness while retaining the sharpness of the edges in the scene.

All aforementioned changes affect both the encoder and decoder's complexity. Except for the new transform and quantization scheme that lower the complexity, all changes raise the time and space complexity. Other new features are not mentioned here because this paper deals with the baseline profile only.

3. DSP OVERVIEW

Since DSPs are tailored for executing signal processing algorithms with very limited resources, their internal architecture is different from the one found in GPPs. Some basic distinctions are the use of fixed-point arithmetic instead of floating-point arithmetic and the use of a flavor of Harvard architecture instead of Von Neumann architecture. Other characteristics of DSP architectures are, e.g., specialized multiply-accumulate units and zero-overhead loops. Recently, DSP architectures have undergone fundamental changes so different architectures are available.

For this study, three leading high-end DSPs with different architectures have been chosen: TMS320DM642 from Texas Instruments, MSC8101 StarCore from Freescale (formerly Motorola SPS) and ADSP-BF533 Blackfin from Analog Devices. All three DSPs are targeted towards low cost real-time video codec implementation as one of their main markets. The StarCore is also highly targeted to communication applications.

Table 1 shows some properties of these three DSPs. In this table, BDTI benchmark is a speed benchmark on a relative linear scale interpolated from the BDTImark2000 results given in [8][1]. More information about the different processors could be found in the relevant company websites.

Table 1. DSPs under examination

	DM642	StarCore	Blackfin
Num. Rep.	Fixed Pt.	Fixed Pt.	Fixed Pt.
Data Width	8/16 bits	16 bits	16 bits
Instr. Width	32 bits	16 bits	16/32 bits
Clock Speed	600 MHz	300 MHz	600 MHz
Intern. Mem.	288 KB	512 KB	148 KB
BDTI Bench.	5480	2700	3350

There are many factors that one has to take into account when examining different DSPs. For example: performance (e.g., speed, memory signature size), cost, size, power consumption, ease of development and integration. In this paper we are by no means trying to compare all these factors or to recommend a specific DSP. The only aspects that will be considered from now on are performance in terms of speed, and the ease of development.

4. IMPLEMENTATIONS AND RESULS

Adapting the decoder to the different DSPs consisted of two phases. The first phase involved making the code work on the DSP and replacing file

[1] The results given in [8] are for the same processors used in this study but with different clock speeds, so the results were scaled accordingly.

access by external memory access, while the second phase involved profiling and optimizing the code.

The baseline profile decoder used takes approximately 13,000 lines of code. Even though modern development environments for DSPs are able to compile ANSI C code, adapting such a long and complex code to such an environment is not an easy task. Things usually break down because of limited resources and because of incompatibilities and instabilities in the DSP development environment.

In this paragraph we'll give two examples (out of many) of bugs that showed up only on specific DSP compilers. A hard to trace bug was that nesting of loops for more than four levels resulted in incorrect code. Another hard to trace compiler bug, which caused the code to break, happened while changing a loop counter inside a loop, in addition to changing it in the loop header. In order to resolve these bugs, appropriate workarounds were applied.

Another problem encountered is the very long time it takes to profile the code on different DSPs. This might take days or weeks even for short image sequences, so we've collected as much profiling data as was feasible in the students' limited time-table. QCIF (176x144) and CIF (352x288) *Carphone*, *Forman* and *Vectra* sequences were used with a frame rate of 30 frames/sec. Only the first frame in each sequence was an I-frame and the rest (usually few dozens of frames) were P-frames.

Profiling results of the non-optimized code showed for all DSPs that memory access is very inefficient and takes a large portion of the total running time. For example, on the StarCore, memory initialization, using the ANSI C routine malloc(), took 50% of the decoder's time. The profiling results for other DSPs were not very different. These results are explained by the fact that the reference code is very inefficient and by the fact that the use of up to five previous reference frames for motion prediction in H.264/AVC incurs large memory requirements.

With this profiling in mind, some code-wide optimizations were performed. The highest compiler optimization level was used and memory initialization and access were greatly optimized by, e.g., changing dynamic memory allocations to static ones, removing unnecessary duplications of intermediate results to temporary memory buffers, avoiding unnecessary memory zeroing, and using the DSP's DMA (Direct Memory Access) controller. These code-wide optimizations had great impact on the performance. For example, the DM642 decoder ran 29% faster after changing most of the dynamic memory allocations to static ones. On the same code, moving from the lowest compiler optimization level to a higher one incurred additional speedup of 38%.

A second phase of optimizations involved local optimizations. An example of such an optimization is for the decoder's inverse transform function. The function that performs an inverse transform on a given block made 16 accesses to the block pixels. Moving the block from the slow external memory to the faster internal memory and making some rearrangement in the function's code resulted in a speedup of 80% for this function on the DM642.

Another example of local optimization is in the code that checks whether motion vectors point outside frame boundaries. In the non-optimized code, every pixel is examined whether it is inside or outside of the frame boundaries. This was replaced by checking if the block is out of the frame boundaries and only if so, making the check for each specific pixel. This optimization, although quite simple, gave a significant speed improvement, especially for the StarCore implementation since this DSP has no branch prediction, and hence its performance is sensitive to conditional statements.

Fig. 2 shows the time distribution among different parts of the DM642 decoder for the non-optimized and optimized codes. The parts of the decoder that were optimized more extensively take a smaller percentage of the total time in the optimized code, compared to the non-optimized code. In both cases, image interpolation (reconstruction of the image based on motion compensation and intra prediction) takes most of the time and takes a larger time share than in all benchmark scores described in [4]-[7]. This is explained by the fact that DSPs have smaller amount of fast internal memory comparing to GPPs and therefore are very sensitive to the non-sequential memory access performed by image interpolation.

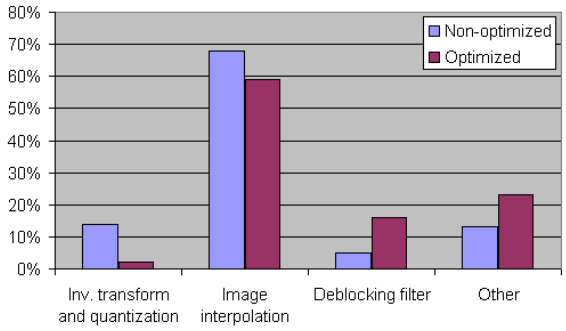

Fig. 2. Time breakdown of H.264/AVC non-optimized and optimized decoder on DM642 DSP.

Table 2 summarizes the decoder time breakdown for the non-optimized and the optimized code after both code-wide and local optimizations. The results are given in QCIF frames/sec.

Even though the BDTI benchmark showed quite different performance for the different DSPs, Table 2 indicates that all three of them had a similar decoding speed for the non-optimized code. For the optimized code, the performance is very optimization-specific but better performance was achieved on the DM642 than for the StarCore. This is because the DM642 was easier to use, so better

optimizations were possible for the same amount of work.

Table 2. Decoder performce on different DSPs[2]. Results are given in QCIF frames/sec.

	DM642	StarCore	Blackfin
Non-optimized	2.1	1.9	1.9
Optimized	9	7.1	-

Looking for more optimizations, two main inefficiencies were detected in the optimized code: 1. Memory access is still inefficient. It does not exploit the hardware to its limits and causes many memory stalls while accessing the slow external memory. 2. The JVT code is very inefficient and this inefficiency is not concentrated in few major bottlenecks but is distributed all over the code.

In addition to the profiling results in Table 2, a theoretical simulation was performed for the StarCore decoder. This simulation calculated the performance while neglecting external memory stalls. The result was 58.7 QCIF frames/sec. This simulation shows the theoretical limit of optimizing external memory access with the current code on this hardware.

5. CONCLUSION

In this study, three pairs of senior undergraduate students have implemented an H.264/AVC decoder on different high-end DSPs, using a baseline version of the JVT reference code. A comparison of the three implementations showed similar performance for a non-optimized version of the code. On all DSPs, the first-to-handle bottlenecks were the same. In addition, second-to-handle optimizations were located based on the specific hardware architecture of every DSP. Ease of use was best for the DM642, so it was possible to achieve better results with this DSP.

The optimized code is still far from real-time performance. This is partly due to the students' limited time for optimizing the code, and is mainly due to the fact that the reference code is very inefficient and has not been designed with a constrained system in mind. It seems like a realistic implementation should start from scratch, or from a much more efficient code, since the reference code had almost been squeezed to its limits in this study.

Currently, a similar work is performed in our lab for an H.264/AVC encoder. Until now, similar memory and other inefficiencies to the ones in the decoder were detected, so the same code-wide optimizations and some local optimizations are being performed. The profiling results indicate that motion estimation and mode decision take most of the encoder's time (55% of the total encoder time for one platform). The maximum possible frame rate is currently two orders of magnitude slower than for the decoder. However, profiling results show that a large speedup could be achieved by using sub-optimal motion estimation and rate-control algorithms and by other algorithmic improvements. Future work in this direction is currently performed in our lab as well as by others, and is expected to improve the encoder's time performance substantially.

[2] No profiling results for the Blackfin optimized decoder were available at the time of writing this paper.

ACKNOWLEDGEMENT

The authors would like to thank the students who invested time and effort in implementing the H.264/AVC codec on various DSPs - Boaz Ackerman and Nir Weingarten, David Katz and Gilad Raichshtain, Roman Baer and Tomer Cohen.

The authors would also like to thank the head of the lab, Prof. David Malah, for his support and valuable comments.

REFERENCES

[1] "Advanced Video Coding for Generic Audio-visual Services," ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, 2003.

[2] G. J. Sullivan, T. Wiegand, "Video Compression – From Concepts to the H.264/AVC Standard," *Proc. of IEEE*, Dec. 2004.

[3] Joint Model ver. 7.2 (H.264/AVC reference software). Available via http://bs.hhi.de/~suehring/tml/

[4] V. Lappalainen, A. Hallapuro, T. D. Hämäläinen, "Complexity of Optimized H.26L Video Decoder Implementation," *IEEE Trans. Circ. and Syst. for Video Technol.*, vol. 13, pp. 717-725, July 2003.

[5] M. Horowitz, A. Joch, F. Kossentini, "H.264/AVC Basline Profile Decoder Complexity Analysis," *IEEE Trans. Circ. and Syst. Video Technol.*, vol. 13, pp. 704-716, July 2003.

[6] X. Zhou, E. Q. Li, Y. K. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," *SPIE Conf. on Image and Video Comm. and Process.*, vol. 5022, pp. 224-235, May 2003.

[7] T.T. Shih, C.L. Yang, Y.S. Tung, "Workload Characterization of the H.264/AVC Decoder," *Proc. 5th IEEE Pacific-Rim Conf. Multimedia*, Japan, November 2004, pp. 957-966.

[8] Berkeley Design Technology, Inc. (BDTI): http://www.bdti.com/