ARCHITECTURAL OPTIMIZATIONS FOR SOFTWARE-BASED MPEG4 VIDEO ENCODER

F. Nasim, S. Masud, N. Khan, K. Virk, A. Farrukh

Multimedia Research Labs, Department of Computer Science,
Lahore University of Management Sciences,
Sector-U, D.H.A, Lahore 54792, Pakistan
Email: {smasud, nkhan} @ lums.edu.pk

ABSTRACT

This paper presents a set of architectural optimizations for improving the performance of an MPEG4 video encoder. The techniques presented here focus on optimizing the encoder architecture rather than module level algorithmic modifications. The optimizations contribute to the development of a fast and memory efficient encoder without affecting video quality. An interface driven methodology has been developed to identify and solve performance bottlenecks for the encoder. Appropriate data flow between components has been developed so that memory intensive operations, such as memory access and copying, are minimized. These optimizations have been applied on MPEG4 simple profile encoder. Results demonstrate orders of magnitude computational improvements without any algorithmic modifications.

1.0 INTRODUCTION

Because of intricate algorithms, the real-time software only implementation of MPEG4 is considered to be computationally complex. Software optimizations are necessary at all levels of the encoder to achieve high performance. Many papers have previously been presented on the algorithmic improvements to the constituent modules of the MPEG4 encoder, especially the motion estimation module [3, 4, 5]. This paper addresses the problem of optimization of the software architecture of an MPEG4 encoder. Purely architectural approaches presented in this paper complement other MPEG4 algorithmic optimizations.

This paper specifically focuses on the interfaces between constituent modules rather than the individual modules shown in figure 1. Optimization has been achieved through improved logical partitioning of encoder modules and efficient data transfer between these modules. An MPEG4 reference implementation by MoMuSys [7] has been used to demonstrate the performance improvements.

The rest of the paper is organized as follows: Section 2 describes typical issues that are encountered in the software implementation of a MPEG4 video encoder. Section 3 describes core optimization techniques that have been developed in this work. Section 4 documents the results obtained from application of the proposed methods on MPEG4 implementation followed by conclusions in Section 5.

2.0 ENCODER IMPLEMENTATION ISSUES

2.1 Memory Management

CPU speed in modern computers is orders of magnitude greater than the bus transfer speed and memory access speed. Memory access is therefore an important factor in improving encoder efficiency. Many core components of an encoder are data intensive in their working. For example, DCT computation, bitstream writing and motion estimation are all core components of an encoder; that are also closely related to memory operations e.g. memory reading, writing, and copying. In general, operations that involve memory access have a high cost on all development platforms [2, 6]. The proposed approach aims to improve encoder performance by minimizing memory operations.

2.2 Inter Module Data Transfer

Video encoders process large amounts of data in many modules. This data must be stored in memory and passed between various blocks of the encoder efficiently. Data transfer and storage costs contribute to performance deterioration since they involve copying of data between memory locations. The 'number-of-memory-operations' factor is an issue in data transfer as well as in general memory management.

2.3 Memory Size Reduction

The amount of memory used by an encoder becomes important when dealing with DSP processors. By reducing the amount of memory used by an encoder, its portability to different DSP systems increases. An encoder should ideally be designed to use minimum possible amount of memory without creating memory

allocation/de-allocation loops that cause memory fragmentation.

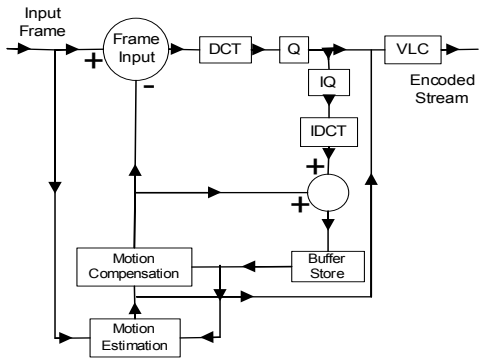

Figure 1: Common video encoder components

Interface optimization, as described in this paper, improves the architectural framework and data communication in the MPEG4 encoder. However, to achieve further performance improvement, algorithmic changes must be assimilated in respective encoder modules.

3.0 OPTIMIZED IMPLEMENTATION

The proposed architectural optimizations affect almost all major MPEG4 encoder components. Figure 2 illustrates which optimizations will affect various parts of the encoder.

Before the techniques are described, it is important to consider the load distribution for the MPEG4 encoder: Motion estimation contributes 40 – 60 % of the load while the remaining percentage is contributed by the other encoder components. This paper does not propose any algorithmic change in the motion estimation component; performance gains reported are the result of only the architectural optimizations at the interface level of various encoder components alone. The motion estimation component has only been optimized at the architectural level, i.e. through data transfer optimizations and memory management.

Platform-independent optimization methods for MPEG4 encoder components as well as optimized architectural design are described below. These include schemes for efficient memory management, bitstream writing and structural improvements in motion estimation and other encoder components.

3.1 Motion Estimation

Because of its large computational load, much research work is being carried out to discover new algorithms that improve the computational complexity of motion estimation. Algorithms that reduce motion estimation load [5, 9] automatically lead to a substantial decrease in encoder load. A good motion estimation algorithm results in accurate identification of motion vectors which leads to a substantial reduction in inter frame size. This paper discusses an optimized method of handling the *output* generated by the MPEG4 motion estimation module including motion vectors and difference data. Effective handling of this output facilitates data reuse and memory size reduction.

In the MPEG4 reference encoder, motion estimation is applied to every frame and the difference data for the entire frame is stored in a frame-sized buffer. In the suggested architecture, motion estimation is still applied in the frame loop but difference data is not stored at this point, only the motion vector and the mode information are stored. These are used while processing macroblocks to calculate difference data for DCT and reconstruction. This saves unnecessary copying of frame-sized buffers and also improves memory utilization by using much smaller macroblock-sized buffers for intermediate calculations.

3.2 Image Interpolation

MPEG4 supports half-pel precision motion estimation. Image interpolation is therefore necessary for computing sub-pixel motion estimation. Increasing the depth of interpolation gives better block matching performance at the expense of increased computational complexity [1]. An efficient method of storing the interpolated pixel information has been developed that reduces this memory overhead.

Sub-pixel motion estimation requires interpolation of the samples of the search area in the reference frame to form a higher resolution interpolated region [1]. This interpolated information is usually stored (for the entire frame) in a buffer that is several times larger than the frame size depending on the sub-pixel accuracy. In the MPEG4 reference implementation, a buffer four times larger than the frame size has been used to store the interpolated data for half-pel motion estimation. This is an inefficient use of memory as motion estimation occurs once per macroblock and only a small fraction of the interpolated information is used at any time. It is thus sufficient to interpolate only the macroblock that is currently being processed instead of interpolating the entire frame resulting in a much smaller buffer size.

3.3 Bitstream Writing

The bitstream writing module writes a variable number of bits to the output bitstream for several components of the encoder. Optimization of MPEG4 bitstream writing functions is described below:

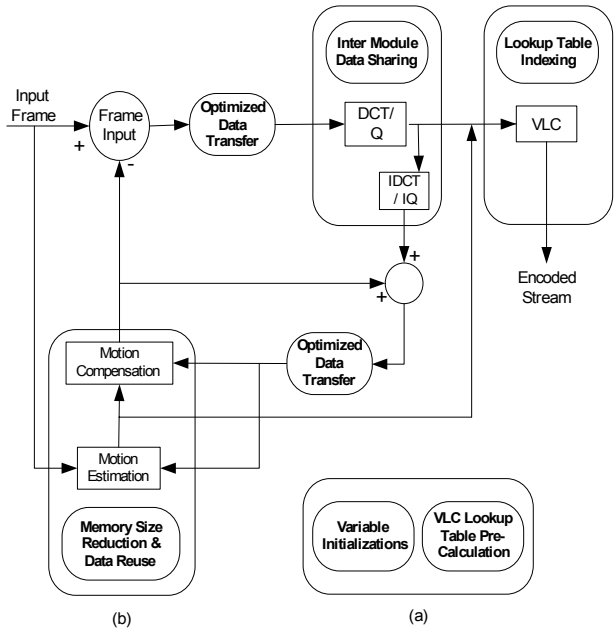

Figure 2: Location of architectural optimizations indicated in the encoder structure. (a) Initializations phase occurs before frame processing begins (b) Frame processing architecture

During initializations, a memory buffer is declared that is equal to largest possible stream size for one frame for storing the encoded stream. During bitstream writing, the 'put_bits' function uses a 32-bit register for storing bits. The contents of the register are moved to stream buffer when they have filled these 32 bits. Thus memory is accessed only at multiples of 32 bits resulting in reduction in memory load operations. To improve performance further, the mechanism has been enhanced to two 32-bit registers instead of one and the writing pointer switches to next register when the first is filled. The contents of both registers are moved to memory simultaneously when the second register is filled, thus reducing the memory load operations by a factor of 50%.

In RISC CPU architectures that have a large number of registers such as TM1300 [6], the above mechanism can be expanded to more than two registers. However, the bottleneck in using more registers is the overhead caused by increased condition checking statements.

3.4 Other Improvements

It has been observed in the MPEG4 implementation that frame-sized buffers are allocated and then de-allocated for every frame that is processed. Such allocation/de-allocation affects the underlying memory structure. All memory allocations for frame-size buffers and any other variables that are used for every frame should be carried out together in a single pass. All such variables were identified in the MPEG4 reference code and their allocation/de-allocation and usage were modified so that these variables were initialized only once and then reused during frame processing. These initializations are performed before any frame processing begins.

The 'best-fit' data types have been used to store image data. For example, unsigned char arrays have been used in the optimized architecture while in the MPEG4 reference implementation the 'short int' type was being used. This was a waste of space since the maximum data size would always fit inside one byte.

Since it uses the smallest possible data type to store values, the proposed architecture can support data level parallelism. For example, when copying is required, instead of copying each 'unsigned char' individually, four unsigned char types can be copied simultaneously by using a single 'int' data type to perform the copying.

Additional performance gain has been realized through pre-calculation of VLC tables, for all possible combinations, and storing them in a lookup table. The pre-calculation occurs in the initialization phase at the same time as the variable initializations and results in speed-up as complex computations are replaced by memory accesses.

4.0 RESULTS AND DISCUSSION

The results of applying these techniques to the MPEG4 reference implementation [7] are shown in Table 1. Sequences 'Akiyo' and 'Foreman' (300 frames each) were used for testing the encoder performance. For the full D1 sequence (720 * 480 pixels), a sequence of 300 frames was captured and used from the movie 'Patriot'. The performance results have been obtained using Intel VTune(TM) Performance Analyzer ver 7.0 running on Intel Pentium-4 at 2.6 GHz with 256MB DDR RAM.

As can be seen in table 1, performance improvement is proportional to the size of the input video frames. This is a result of improved memory management and data transfers between interfaces of various modules of the encoder.

Table 1: Improvement in encoder performance

Seqs	*Size*	*MPEG4 Reference (ms)**	*MPEG4 Optimized (ms)**	*Speed Improvement factor*
Akiyo	QCIF	1131	76	15.93
Foreman	QCIF	1361	89	15.29
Akiyo	CIF	14157	292	48.48
Foreman	CIF	14806	334	44.33
Patriot	Full D1	50139	1100	45.58

* Note: Times have been calculated for encoding 25 frames excluding motion estimation time in order to highlight the gains achieved through architectural optimizations.

Table 2: PSNR values

Seqs	*Size*	*MPEG4 Ref PSNR (Y) dB*	*MPEG4 Opt PSNR (Y) dB*
Akiyo	QCIF	34.51	34.50
Foreman	QCIF	31.41	32.41
Akiyo	CIF	36.60	36.61
Foreman	CIF	32.41	33.41
Patriot	Full D1	35.29	36.21

* Note: Average PSNR of Y component for 300 frames

The performance gain is more pronounced on frames of larger size as the memory and data size considerations become more important on frame sequences of larger dimensions. Application of the proposed methods contributes to a better underlying physical memory structure. Superior data exchange and reuse methodologies also reduce the memory size required and the total number of memory accesses. These techniques are generic and can be applied to any software video encoder to improve real-time performance.

The optimized encoder takes 16 times less time than the original implementation when a QCIF sequence is used. The performance improves even further when the larger CIF and Full D1 frame sequence is used, giving approximately 45 times improvement. The improvement is independent on the amount of motion in the test sequence. This is because the techniques presented in this paper do not affect the Motion Estimation algorithm used (or the algorithm of any other module). Further gains in performance can be achieved by introducing algorithmic improvements in various modules of the encoder.

The performance gains achieved using the techniques presented in this paper are close to the improvements achieved through algorithmic optimizations presented in other research. It is important to note here that in previously published work, algorithmic optimizations were applied to the motion estimation module. As discussed in section 3, the motion estimation module contributes greatest load in a video encoder. Therefore, its optimization results in large performance gains. For example, Zheng et al [8] demonstrated a 35 – 80 times improvement in codec performance using algorithmic modifications. In our work, performance gains in the range of 15 to 50 times for different frame sizes have been achieved by using only software architectural modifications without changing the motion estimation algorithm. This has been due to improved memory access and data transfer structure.

5. CONCLUSION

This paper presents an optimization of the software implementation of an MPEG4 encoder using only architectural optimizations. The techniques presented here provide 15 to 50 times performance gains as a result of improved coupling of data within modules and efficient transfer of data between communicating modules. Optimum memory management practices have also been suggested. The proposed methods do not degrade the video quality of the encoded bit stream as they only affect data and memory structure of the MPEG4 encoder and leave intact the algorithms used in various modules of the encoder. The results illustrate the fact that architectural optimizations can have a strong impact on software based MPEG4 encoder efficiency.

REFERENCES

[1] I. E. G. Richardson, “Video Codec Design”, John Wiley & Sons, 2002

[2] Intel Corporation, “IA-32 Intel Architecture Software Developer’s Manual”, vol. 1, vol.2

[3] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, “A novel unrestricted center-biased diamond search algorithm for block motion estimation”, IEEE transactions on Circuits and Systems for Video Technology, Vol. 8, No. 4, pp. 369-377, August 1998

[4] J. McVeigh, G. K. Chen, J. Goldstein, A. Gupta, M. Keith, and S. Wood, “A Software-Based Real-Time MPEG-2 Video Encoder”, IEEE transactions on Circuits and Systems for Video Technology, Vol. 10, No. 7, pp. 1178-1184, Oct. 2000

[5] K. Ramkishor, P.S.S.B.K. Gupta, T.S. Raghu, K. Suman, “Algorithmic optimizations for software-only MPEG-2 encoding”, IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, pp 366 – 375, 2004

[6] Philips Electronics North America Corporation, “TM1300 Data Book”, 1999

[7] Publicly available MPEG-4 reference software from ISO (ISO/IEC 14496-5), http://www.iso.ch/iso/en/ittf/PubliclyAvailableStandards/14496-5_Compressed_directories/Visual/, MoMuSys Implementation

[8] W. Zheng, I. Ahmad, M. L. Liou, “Real-Time Software Based MPEG-4 Video Encoder”, Proceedings of Workshop and Exhibition on MPEG-4, 18-20 June 2001 pp 71 – 74

[9] X. Jing, L. Chau, “An Efficient Three-Step Search Algorithm for Block Motion Estimation”, IEEE transactions on Multimedia, June 2004, pp435-438