HIGH-QUALITY AND PROCESSOR-EFFICIENT IMPLEMENTATION OF AN MPEG-2 AAC ENCODER

Yuichiro Takamizawa, Toshiyuki Nomura, and Masao Ikekawa

NEC Corporation

ABSTRACT

Presented here is MPEG-2 AAC LC Profile encoder software for an Intel Pentium III processor. MDCT and quantization processing are accelerated by the use of SIMD instructions. Psycho-acoustic analysis in the MDCT domain makes the use of FFTs unnecessary. Better sound quality is provided by greater efficiency in quantization processing and Huffman coding. All of this results in high-quality and processor-efficient implementation of an MPEG-2 AAC encoder. Sound quality achieved at 96 kbps/stereo is significantly better than that of MP3 at the same bitrate. The encoder works 13 times faster than realtime for stereo encoding on an 800MHz Pentium III processor.

1. INTRODUCTION

While MPEG-1/Audio Layer III (MP3) [1] has widely been used as a high quality audio coding algorithm for portable audio devices, PC jukebox software, and Internet music distribution systems, it is now being replaced by MPEG-2 Advanced Audio Coding (AAC) [2]. AAC can encode an audio signal of CD quality at 48~64 kbps/ch, a bitrate 30% lower than that of MP3. In the next few years, a wide range of AAC products, including portable audio devices and PC jukebox software, are expected to appear on the market.

In most cases, PC encoder software is currently attached to such products, and customers demand that it provides both high sound quality and fast encoding. In general, these two demands require a trade-off: better sound quality usually results in slower encoding. While this problem might be overcome with sufficiently detailed information regarding encoder implementation, the AAC standard document [2] describes only decoding procedures and bitstream format; it says almost nothing about high-quality, processor-efficient implementations, the performance of an encoder strongly depends on its manner of the implementation.

We have developed the necessary techniques for high-quality, processor-efficient implementation and have succeeded in developing high-quality, processor-efficient encoder software for use on an Intel Pentium III processor. This paper describes the performance of our AAC encoder and the techniques we have employed to achieve high performance.

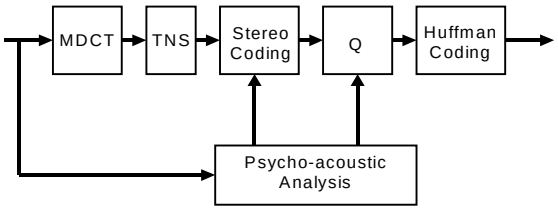

Figure 1: Block diagram of AAC LC profile encoder.

2. AAC ENCODING ALGORITHM

Three coding modes can be used with AAC: Main profile, Low Complexity (LC) profile, and Scaleable Sampling Rate (SSR) profile. Our newly developed encoder is designed to use the LC profile because it has been adopted for use in Japanese digital TV and currently seems to be the most widely used of the three in audio applications.

Figure 1 shows the components of an AAC LC profile encoder. An MDCT (Modified Discrete Cosine Transform) [3] block transforms an input audio signal into MDCT coefficients, which represent a frequency spectrum. The MDCT coefficients are quantized in the quantization block after redundancies have been removed in the TNS (Temporal Noise Shaping) block and the Stereo coding block. The psycho-acoustic analysis block controls the quantization step to minimize audible quantization error. The quantized MDCT coefficients are then Huffman coded and multiplexed to the bitstream.

Though one implementation example of an AAC encoder software is provided by ISO/IEC[2], its performance is insufficient in terms of both sound quality and encoding speed. It works over eight times slower than real-time for encoding (Pentium III 800 MHz, 44.1 kHz, 96 kbps/stereo), and its sound quality is generally worse than MP3 at the same bitrate. We have found methods, applicable to generic DSPs and microprocessors, for improving both encoding speed and sound quality (see Section 3). Further, we have introduced the use of SIMD (Single Instruction stream-Multiple Data streams) instructions to increase encoding speed even more (see Section 4).

3. QUALITY AND SPEED ENHANCEMENT

This section describes three methods we have employed in our encoder software to improve encoding speed and sound quality. These methods are applicable to implementation on generic DSPs and microprocessors.

3.1. Psycho-acoustic analysis on MDCT coefficients

In conventional encoder implementations [1],[2], the input PCM signal is transformed by FFT (Fast Fourier Transform), and psycho-acoustic analyses, such as masking calculations, are performed on the FFT coefficients [1],[2],[4]. We have found, however, that the 2048-point MDCT employed in AAC has sufficient frequency resolution for psycho-acoustic analysis and is able to replace the 2048-point FFT without any degradation in sound quality. For that reason, we have omitted the 2048-point FFT and utilized existing MDCT coefficients, which are generated in the MDCT block, for the psycho-acoustic analysis.

In the MDCT coefficients, the phase information of the input PCM signal which is utilized in conventional FFT based psycho-acoustic analysis [1],[2] is lost. We have modified the conventional psycho-acoustic analysis algorithm [4], which is performed on FFT coefficients, to fit the analysis on the MDCT coefficients [5]. By substituting the existing MDCT calculation for FFT calculation, encoding speed is accelerated while sound quality is maintained.

3.2. Filtering scalefactor values

For each MDCT coefficient, the quantization block searches for the quantization step that achieves best sound quality below a given bitrate. The derived quantization steps for each coefficient are expressed as scalefactor values. These values are integral values and are differential-coded along the frequency on the basis of the values shown in Table 1 [2].

Since this table is designed so that the smaller differential values can be coded with shorter codes, sudden major changes or successive minor changes in scalefactor values might make the code longer and degrade the coding efficiency. To prevent this, a low-pass filter is applied to the scalefactor values to suppress the changes.

Table 1: Code table for differential scalefactor values.

Differential value	Code
60	111111111111110011
:	:
3	11011
2	1100
1	1010
0	0
-1	100
-2	1011
-3	11010
:	:
-60	111111111111101000

This method decreases the code bits for the scalefactor and increases the code bits available for the quantized values. Though low-pass filtered scalefactor values may not be the best ones from the psycho-acoustic point of view, we have confirmed that bit-reduction by this method leads to sound quality improvement.

3.3. Selection of Huffman tables

In AAC, 1024-MDCT coefficients are quantized and grouped into 49 bands called “scalefactor bands”. The quantized coefficients are Huffman coded, in which one Huffman table is selected from 12 Huffman tables for each scalefactor band. In general, the Huffman table which outputs the shortest code is selected. The numbers (0..11), which indicate the selected Huffman table in each scalefactor band, are run-length coded and multiplexed into the bitstream as Huffman table information.

In the run-length coding, a longer run enables a shorter code and improves the coding efficiency. The bit count for Huffman table information should be taken into account in selecting a Huffman table, and selecting the Huffman table that minimizes the Huffman code is not necessarily the best approach. A Huffman table should be selected so that the total bit count for Huffman code and Huffman table information is minimized [2]. To this end, we have employed the following procedures in our encoder software for selecting Huffman tables.

(1) For each scalefactor band *sfb*, select a Huffman table which enables the shortest Huffman code, and set the selected Huffman table number to *cb[sfb]*
(2) Set *sfb* to 1
(3) If *cb[sfb] = cb[sfb-1]*, go to (10) because *cb[sfb]* is already suitable for run-length coding and is not to be changed
(4) Calculate the total bit count *normal_bits* for the Huffman code and Huffman table information for scalefactor band 0..*sfb*+1
(5) Calculate the total bit count *bits1* for Huffman code and Huffman table information for scalefactor band 0..*sfb*+1 with *cb[sfb] = cb[sfb-1]*
(6) Calculate the total bit count *bits2* for Huffman code and Huffman table information for scalefactor band 0..*sfb*+1 by replacing the run of *cb[sfb-1]*, which starts from (*sfb*-1) to lower frequency, with that of *cb[sfb]*
(7) If *normal_bits* is the minimum among *normal_bits*, *bits1*, and *bits2*, do nothing and go to (10)
(8) If *bit1* is the minimum among *normal_bits*, *bits1*, and *bits2*, let *cb[sfb] = cb[sfb-1]*
(9) If *bit2* is the minimum among *normal_bits*, *bits1*, and *bits2*, replace the run of *cb[sfb-1]* with that of *cb[sfb]*
(10) If *sfb* < 49, increment *sfb* and go to (3)

In these procedures, each neighboring run is examined for merging {(5), (6)}. Generally, by merging the runs, the bit count for Huffman table information is reduced, and the bit count for the Huffman code is increased. If the total bit count for the Huffman code and Huffman table information would be reduced, the runs are merged {(8), (9)}.

These procedures are executed in a quantization loop. In the quantization loop, the quantization step which would achieve the best sound quality below a given bitrate, is searched for over the course of several trials. That is to say, in the encoding of a frame, these procedures are executed several times.

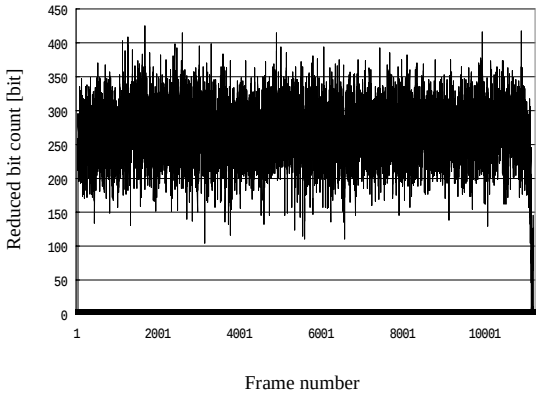

Figure 2: Reduced bit count in each frame.

Though these procedures should be repeated until no more changes in *cb[sfb]* are needed, in order to reduce complexity, they are executed only once in the quantization loop in our implementation. Rather, it is after the quantization loop processing in which the best quantization step is found that these procedures are executed repeatedly.

Figure 2 shows the reduced bit count in each frame when a typical pop music song (261 sec, 11,255 frames) is encoded at 96 kbps/stereo (2,229 bits/frame). The average reduced bit count is 267 bits, which is equivalent to 11.5 kbps reduction (gaining) in the bitrate. The bit count reduction is derived over all frames except for the silent period (at the beginning and the end of the song). By re-using the reduced bit count, the sound quality can be significantly improved.

4. ACCELERATION BY SIMD INSTRUCTIONS

Most recent PC microprocessors have a SIMD instruction set which is designed to accelerate the execution of multi-media applications. This section describes how to utilize the SIMD instruction set for AAC encoder software.

4.1. Parallel MDCT

MDCT can be efficiently implemented by utilizing FFT [6]. It is known that SIMD instructions are effective for accelerating such transform operations. Implementation examples of a sub-band synthesis filter in MPEG-1 Audio and FFT with SIMD instructions have been reported in [7],[8]. These methods find the parallelism in one transform operation and use SIMD instructions to execute multiple operations in one transform by one instruction. To make maximum use of SIMD instructions, values to be stored on a SIMD register are expected to be located on consecutive memory addresses. However, this is not possible with sub-band synthesis filters or FFTs because of their complex signal-flow. They need re-ordering or packing instructions to store multiple values located at non-consecutive memory addresses on a SIMD register. The overhead resulting from the re-ordering or packing instructions degrades execution speed.

To reduce the overhead, we took a different approach to the use of SIMD instructions for MDCTs. Though conventional methods utilize SIMD instructions to perform multiple operations in one MDCT, our method utilizes them to perform single operations in multiple MDCTs. We have implemented this method on an Intel Pentium III processor that has a SIMD instruction set (SSE: Streaming SIMD Extension) [9] which performs four floating-point operations in parallel. In a stereo AAC encoder, four MDCTs (left and right channels of current and next frames) are performed at the same time by SIMD instructions.

This method reduces the overhead caused by the re-ordering or packing operations described above. By interleaving and storing the four input signals to the MDCT, complex signal-flow operations can be easily and efficiently implemented by the SIMD instructions.

With the use of this method, MDCT processing is accelerated by 27% without any degradation in sound quality. There are two reasons why the processing is not accelerated by 75% by 4-parallel processing. One is that the SIMD instructions are not four times faster than conventional floating-point instructions [9]. The other is that we simply rewrote the C-code using the intrinsic functions [10]. By rewriting with assembly code, performance might be improved further. Our method is also applicable to other coding/decoding systems, such as MP3.

4.2. Parallel quantization

In the quantization block, the calculation of (x^0.75) is repeatedly executed. This calculation can be done by

pow(M, 0.75)

in C-language. However, the pow(M, N) function needs many CPU cycles for execution. Therefore, this calculation should be performed as follows:

sqrt(sqrt(M) x M)

By utilizing SIMD instructions, this calculation is done in parallel. The simplest way is to replace “sqrt” with a SIMD sqrt instruction. However, this is not the best approach for a Pentium III processor. In an SSE instruction set, “sqrt” is slower than “rsqrt” which calculates (M^-0.5). By utilizing “rsqrt” with “rcp”, which calculates (1/M), processor-efficient implementation of (M^0.75) can be achieved as follows:

rsqrt(rcp(M) x rsqrt(M))

These instructions execute four (M^0.75) calculations in parallel. By employing SIMD instructions, the quantization operation is accelerated by 20%.

5. PERFORMANCE EVALUATION

We used a PC with an 800 MHz Pentium III processor to evaluate the performance of our encoder software. Table 2 shows the consumed CPU cycles in each AAC block when a typical pop music song (44.1 kHz, stereo) is encoded at 96 kbps/stereo in realtime.

By employing the new methods described above, our encoder software achieves real-time encoding with a 48.6 MHz CPU (excluding file access), and works 13 times

Table 2: Consumed cycles (Mcycle/sec/stereo).

MDCT	5.9
TNS	5.8
Quantization	20.6
Psycho-acoustic analysis	8.8
Others (Stereo Coding, etc.)	7.5
Total	48.6

Table 3: Seven-grade comparison scale.

B is much better than A	+3
B is better than A	+2
B is slightly better than A	+1
B is the same as A	0
B is slightly worse than A	-1
B is worse than A	-2
B is much worse than A	-3

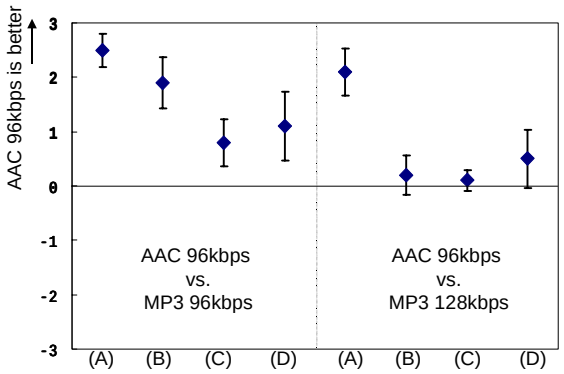

Figure 3: Subjective test result.

faster than realtime (including file access) and over 100 times faster than implementation by ISO/IEC[2].

We subjected 11 trained listeners to a rough subjective quality test using CMOS (Comparison Mean Opinion Score) test methodology [11]. The sequence played to the listeners for each trial was Ref/A/B, Ref/A/B, where Ref was the original (not coded) sound, and A and B were both coded signals. The assignment of encoders (MP3/AAC) to positions A and B was randomized and unknown to the listener. The listeners were asked to judge whether "A" or "B" had better sound quality by using a seven-grade comparison scale (Table 3). The playback was done using STAX Lambda Nova headphones in a controlled (acoustically isolated) room. The MP3 encoder used in the test was provided by FhG (Fraunhofer-Gesellschaft). This encoder is famous for its high sound quality and is widely employed in PC jukebox software. Figure 3 shows the test results (average scores / 95% confidence interval) obtained. The figure shows a comparison of subjective sound quality for "castanets", "pop music", "glockenspiel", and "Suzanne Vega", all of which are known as critical materials. As the figure indicates, the sound quality of our AAC encoder was significantly better than that of MP3 at the same bitrate (96 kbps/stereo) and was equivalent to or better than that of MP3 at 128 kbps.

6. CONCLUSION

We have developed MPEG-2 AAC LC profile encoder software. The introduction of several new methods to enhance sound quality and encoding speed have resulted in high quality, processor-efficient implementation of MPEG-2 AAC encoder software. The encoder achieves significantly better sound quality than MP3, and works 13 times faster than realtime for stereo encoding on an 800MHz Pentium III processor.

ACKNOWLEDGEMENTS

The authors wish to thank Mr. Ichiro Kuroda, Dr. Akihiko Sugiyama, and Dr. Masahiro Serizawa for their very valuable encouragement and support.

REFERENCES

[1] ISO/IEC 11172-3, "Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s, Part 3: Audio," Aug. 1993.

[2] ISO/IEC 13818-7, "Generic coding of moving pictures and associated audio, Part 7: Advanced Audio Coding (AAC)," Mar. 1995.

[3] J. Princen and A. Bradley, "Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation," IEEE Transactions on ASSP, Vol. 34, pp. 1153-1161, Oct. 1986.

[4] J. D. Johnston, "Transform Coding of Audio Signals Using Perceptual Noise Criteria," IEEE Journal on Selected Area in Communications, Vol. 6, No. 2, Feb. 1988.

[5] T. Nomura, Y. Takamizawa, "Processor-Efficient Implementation of a High Quality MPEG-2 AAC Encoder," to be presented at AES 110th Convention, May 2001.

[6] D. Sevic and M. Popovic, " A New Efficient Implementation of the Oddly Stacked Princen-Bradley Filter Bank," IEEE Signal Processing Letters, Vol. 1, No. 11, pp. 166-168, Nov. 1994.

[7] Intel, "Using MMX™ Instructions to Implement a Synthesis Sub-Band Filter for MPEG Audio Decoding," http://developer.intel.com/software/idap/resources/technical_collateral/mmx/ap533.htm.

[8] Intel, "Using MMX™ Instructions to Perform Complex 16-Bit FFT," http://developer.intel.com/software/idap/resources/technical_collateral/mmx/AP555.HTM.

[9] K. Diefendorf, "Pentium III = Pentium II + SSE," MICROPROCESSOR REPORT, Vol. 13, No. 3, Mar. 1999.

[10] Intel, "Software Development Strategies for Streaming SIMD Extensions", http://developer.intel.com/vtune/cbts/strmsimd/appnotes/ap814/swdevel.pdf

[11] B. Edler, J.Herre, and K.Brandenburg, "Core experiment methodology for MPEG-4 audio," ISO/IEC JTC1/SC29/WG11, N1748, Jul. 1997.