FAST IMPLEMENTATION OF THE MPEG-4 AAC MAIN AND LOW COMPLEXITY DECODER

Antonio Servetti[1], *Alessandro Rinotti*[1], *Juan Carlos De Martin*[2]

[1]Dipartimento di Automatica e Informatica/[2]IEIIT-CNR
Politecnico di Torino
Corso Duca degli Abruzzi, 24 — I-10129 Torino, Italy
E-mail: [servetti|alessandro.rinotti|demartin]@polito.it

ABSTRACT

We present a fast software implementation of the MPEG-4 AAC Main and Low Complexity (LC) decoder. The reference implementation is analyzed and selected algorithms are presented to improve the performance of most of its building blocks, i.e., bitstream de-formatter, noiseless decoding, prediction, and filterbank.

The code is further optimized by means of assembler procedures for the filterbank and prediction tools to exploit the Intel Pentium Streaming SIMD Extension (SSE) instruction set. SSE performs operations over four single precision floating-point numbers in a single instruction using 128-bit long registers. Results shows that the presented decoder proves to be almost five times faster than the reference implementation, while preserving its full compatibility with the MPEG-4 standard.

1. INTRODUCTION

The increasing demand for digital audio media is mostly addressed by high-quality audio coding devices implementing the MPEG-1 Layer-3 standard. Only recently its scheduled successor, the MPEG-4 Advanced Audio Coding (AAC) standard [1], is gaining recognition. Among the factors slowing down the adoption of this advanced standard, computational complexity is one of the most important.

The MPEG-4 AAC coder is approximately 30% more bitrate efficient than the MP3 algorithm and outperforms its predecessor achieving "indistinguishable" CD audio quality at 128 kb/s [2]. The quality of AAC streams, however, must be supported by fast decoders with a speed comparable to current MP3 players. The reference AAC ISO software does not meet such speed requirement, making a thorough revision of the code mandatory.

Previous works addressed the design of AAC software decoders (e.g., [3]), while other papers analyzed possible optimizations for implementations on DSP [4][5][6] or RISC processors for portable players [7].

In this paper we presents an MPEG-4 AAC software implementation optimized for speed that exploits the Intel Pentium Streaming SIMD Extension (SSE) instruction set.

The paper is organized as follows. Section 2 and 3 describe the MPEG-4 AAC decoder tools we focused on and the capabilities of SSE instructions. Section 4 presents a performance analysis of the ISO reference software followed in Section 5 by detailed description of the presented optimization process. The results of the optimized implementation are discussed in Section 6.

This work was supported in part by Telecom Italia Lab, Italy, http://www.telecomitalialab.com.

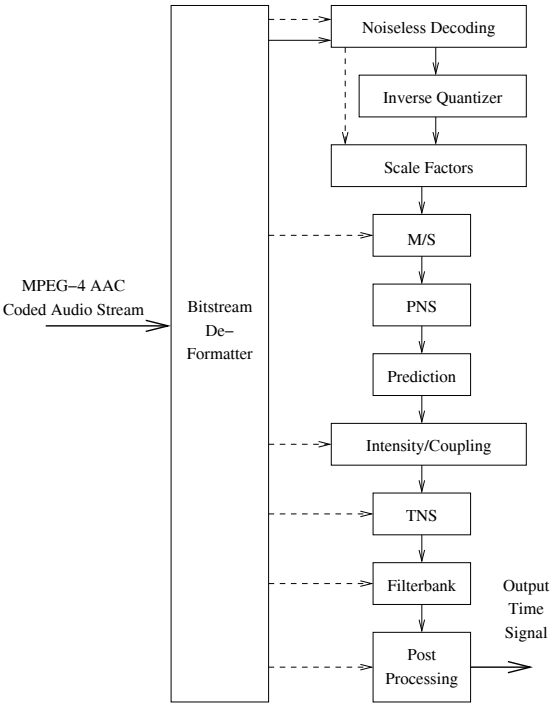

Fig. 1. MPEG-4 AAC decoder block diagram.

2. AAC ALGORITHM

The MPEG-4 standard [1] defines several tools with different audio coding algorithms to establish optimal coding efficiency for a broad range of applications that span from low bitrate coding of speech signals up to high-quality multi-channel audio coding. The key component of MPEG-4 General Audio, that covers the bitrate range from 16 kb/s per channel up to 64 kb/s and higher, is the AAC tool, backward compatible with the MPEG-2 AAC [8].

The AAC encoding algorithm, as other perceptual encoders, e.g., the widely known MP3, uses psychoacoustics to reach the target of efficient compression. It is a lossy compression tech-

nique where the quantization noise is carefully distributed over frequency bands so that it is masked by the signal energy.

AAC supports up to 48 main audio channels with sample rates that range from 8 kHz to 96 kHz. Different trade-offs between quality and complexity are provided by three profiles: *Main profile*, *Low Complexity profile*, and *Scalable Sample Rate profile*.

Fig. 1 shows the arrangement of the building blocks of an MPEG-4 Main decoder [9]. They are here briefly illustrated to understand the optimization process described in Section 5.

Bitstream De-formatter: It parses the bitstream to separate AAC data into the parts used as input for each tool. The length of the AAC frames varies from frame to frame because of the bit reservoir technique. Each frame represents 1,024 PCM samples per channel. Two different headers are specified in the AAC standard. The Audio Data Interchange Format (ADIF) header is for file-based applications and it is present at the beginning of the file. The Audio Data Transport Stream (ADTS) header is suited for transmission protocols and it consists of one header per AAC frame.

Noiseless Decoding: It is used to reduce redundancy of scalefactors and quantized spectral coefficients. The latter are segmented into sections and a different Huffman codebook can be used to code each section. There are eleven Huffman codebooks for the spectral data and one differential scale factor codebook.

Prediction: In the Main profile an optional second-order backward adaptive predictor, with a lattice structure, improves coding efficiency especially for stationary signals. For each spectral component up to 16 kHz there is one corresponding predictor where each predictor exploits the autocorrelation between spectral component values of the two preceding frames. The predictor parameters are adapted to the current signal statistics on a frame-by-frame base, using an LMS-based adaptation algorithm.

Filterbank: Its function is to convert spectral values into a time-domain output by an Inverse Modified Discrete Cosine Transform (IMDCT) filter that can dynamically switch between block lengths of 2,048 and 256 points. Long blocks offer improved coding efficiency for stationary signals and short block present optimized coding capabilities for transient signals.

3. SSE INSTRUCTIONS

With the launch of the Intel Pentium III processor many new features are available for application developers [10]. Among them seventy new instructions, called Streaming SIMD Extensions (SSE), improve the Pentium II as MMX did for the Pentium. Processors having Single Instruction Multiple Data (SIMD) support process more than one data element in one instruction. MMX and SSE share the concept of SIMD, but MMX instructions represent SIMD for integers while SSE instructions are SIMD for single-precision floating point numbers. SSE instructions operate on four 32-bit floats simultaneously: eight new 128-bit registers have been defined and applications can execute non-SSE and SSE instruction at the same time. Not all the new instructions are for SIMD floating-point. Of the 70, 50 are SIMD floating-point, 12 are for SIMD integer, and the remaining eight are cacheability instructions. Even if SSE can perform operations on four single precision floating-point numbers in a single instruction, the number of instructions when using SIMD and SSE is not exact one-fourth of the non-SIMD case. Some instructions will be required to rearrange the data (this process is called swizzling) so that it is in a format acceptable to the SIMD instructions. In practice, SIMD instructions should be able to roughly double the efficiency of the operations [11].

Hence audio applications, which utilize floating-point calculations, stand to benefit from the usage of SSE.

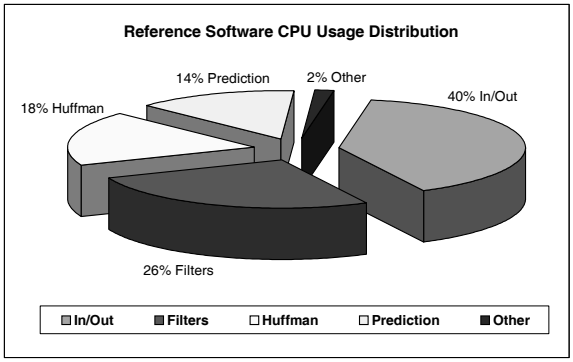

Fig. 2. CPU usage distribution between MPEG-4 AAC decoding blocks.

4. REFERENCE SOFTWARE DECODER ANALYSIS

The MPEG-4 AAC reference software implementation has been developed by a large group of contributors with the aim of implementing all the required functionalities and acting as a reference for stream compatibility issues. Since it was not developed to deliver good performance, it suffers from heavy computational load even on modern processors.

Version 1.0 of the ISO MPEG-4 Natural Audio Decoder reference software, tested on a 866 MHz Pentium III processor with 128 MB of RAM memory, and compiled optimized for speed with the Intel C compiler v.5.0 for the Windows operating system, achieves only 8.5 realtime speed for a stereo signal.

With regard to this implementation, a profile session can evaluate the CPU cycle distribution between the various AAC modules described in Section 2. The results are shown in Fig. 2.

It can be noticed that the majority of time is spent in the input/output functions (bitstream reading and parsing) that are build upon a generic library for bit manipulation. Secondly, filters (above all MDCT and windowing) take 26% of the overall load, while Huffman decoding represents 18% of the whole decoding process. Since the profiling was applied on the decoding of a Main profile bitstream, part of the time (14%) is spent in the prediction process, a tool not used in the Low Complexity profile.

5. DECODER OPTIMIZATION

Each different building block of the AAC decoder was re-implemented to exploit the functionalities of the Intel SSE instruction set. Speed improvements have been provided for the Main and Low Complexity profiles maintaining full conformance with the standard.

A detailed description of each improved algorithm follows.

5.1. Bitstream Processing

The AAC bitstream syntax is very flexible and permits its instantaneous data rate to vary as required by the audio signal: the length

of each frame is, therefore, not constant and variable-rate headers are used. As a consequence, the bitstream parsing function of the reference software is very complex and slow. It does not exploit the knowledge of the bitstream format (frame and field size) to reduce unnecessary checks while reading the data buffer.

An improved bit manipulation library was developed: many levels of indirection were removed and needless checks on buffer overruns were skipped.

A boost in performance was obtained using a 32-bit cache to reduce bitstream reads. Only on cache misses a new word is fed into the buffer cache of the decoder, so successive bit level accesses can be carried out using only fast bit operators.

5.2. Huffman decoding

The Huffman decoding function operates on 1,024 compressed spectral coefficients read from the bitstream. They are split into several sections each one using a different Huffman codebook.

Original ISO code uses a fairly efficient algorithm that saves memory and CPU cycles: Huffman codewords are read from the bitstream and searched in the corresponding codebook from the shortest to the longest one. That is because shorter codes are more common than longer codes, and this way short codes are decoded quite fast while long ones take longer.

When speed is the main concern and more memory can be used, a faster decoding technique can be implemented building a lookup table with an efficient structure. Firstly, the decoder unsigned Huffman codebooks have been extended to speed up the decoding, but maintaining full compatibility with the standard. Secondly, multi-level tables have been designed for faster codeword search. If N is the length of the longest codeword of a codebook, a first level lookup table that covers only M_1 bits, with $M_1 < N$, is used. The decoder reads M_1 bits from the stream and looks them up in the table. The table will tell if the codeword is that many bits or less, in which case it is completely decoded, or if it is longer, and more bits (M_2) are needed to continue the search process on a second level table.

In the worst case two table lookups must be performed but decoding test show that usually above 90% of the compressed spectral values are present in the first lookup table with no need of the second step. The memory used to store the new Huffman tables is typically affordable for a software decoder, compared to the case of a single lookup table with 2^N entries. Using the proposed technique, Huffman decoding module is now five times faster than its original implementation.

5.3. Prediction

In AAC decoding a second-order backward-adaptive lattice structure predictor is used for each spectral component up to 16 kHz. Since there is no dependence between prediction coefficients of the same frame then computation can be parallelized with SIMD instructions: four spectral coefficients can be processed in a single instruction.

The LMS-based adaptation algorithm applied to predictor parameters was modified to exploit SIMD capabilities and to reduce memory access. A dependence graph was built for variables to optimize access to SSE registers. We grouped together operations on independent data so that *out-of-order* Pentium III processor instructions could be used to improve performance.

Table 1. Comparison of time (ms) spent during profiler session by the ISO reference software and the optimized decoder using the aacs48s96.aac bitstream.

Tool	*Reference Time (ms)*	*Optimized Time (ms)*	*Speed Improvement*
In/Out	11340	3188	x 3.55
Filters	7554	1652	x 4.57
Huffman	5198	1062	x 4.89
Prediction	3980	1936	x 2.05
Other	473	187	x 2.52
Total	**28545**	**8025**	**x 3.55**

5.4. Filters

The filterbank, a fundamental component of the AAC decoder, uses an Inverse Modified Discrete Cosine Transform (IMDCT) to convert the internal time-frequency representation into a time-domain output signal. The IMDCT employs a technique called Time-Domain Aliasing Cancellation [12] to transform, for each channel, the N/2 time-frequency values X_{ik} into the N time-domain values x_{in}. The analytical expression is:

$$x_{in} = \frac{2}{N}\sum_{k=0}^{\frac{N}{2}-1} X_{ik}\cos\left[\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2})\right], \quad n = 0,\ldots,N-1, \tag{1}$$

where n is the sample index, N the transform window length, i the block index, and $n_0 = (N/2+1)/2$. The transform block length N can be set to either 2048 or 256 samples.

After a manipulation of the input data the IMDCT filter can be implemented via an N/4 FFT algorithm as shown in [13]. The ISO reference software FFT implementation was rewritten in assembler to take advantage of the Pentium SIMD instructions and SSE registers. The Decimation In Time (DIT) approach for the Radix-2 FFT algorithm was used [14]. The first and second stage operations were redesigned for SIMD processing while the standard algorithm was applied to the next stages on four data inputs simultaneously. The first two FFT stages perform simple transforms on respectively two and four spectral values. An optimized algorithm and a data swizzling procedure were developed to exploit short transform characteristics and SIMD parallelism also in these stages.

6. RESULTS

The performance of the optimized decoder implementation has been evaluated using an encoded test signal from the MPEG repository. It is a 96 kb/s AAC main profile stereo audio signal sampled at 48 kHz.

Firstly, we compared the results of function profiling for the ISO reference software and for the presented optimized version. As it is shown in Table 1 the execution time of the decoder was reduced from 28.54 seconds to 8.02 seconds. Timing data from profiling sessions is especially useful to evaluate speed improvements on each decoding block. We notice that filtering operations took advantage of SIMD instructions with a speed increase of nearly five times. This is also the result of a better code implementation,

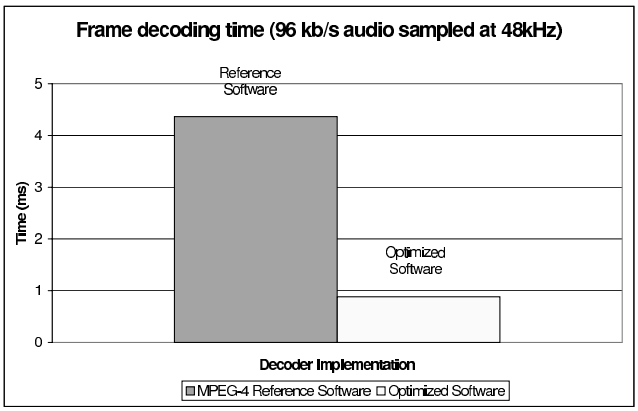

Fig. 3. MPEG-4 AAC decoder performance.

e.g., many values for windowing were computed in advance and needless functions were removed during the code reorganization. The new design of Huffman decoding and of bitstream manipulation were the key for a speed improvement of 4.89 and 3.55 times respectively. The prediction tool is now two times faster than the reference code.

A PC a with Pentium III 700 MHz processor was used in the decoding speed evaluation. Fig. 3 shows the average CPU usage in frame decoding for a 96 kb/s audio sampled at 48 kHz. The ISO reference decoder provides only 4.5 realtime decoding (4.36 ms), while the Pentium III optimized version is nearly five times faster (0.88 ms). The presented implementation also proves to be 21% faster than an existing AAC software decoder designed for speed, the FAAD decoder ver. 26102001.

The optimized decoder has been successfully tested against the standard test streams for computational accuracy and bit-compliance in accordance with the conformance standard by the ISO/IEC 14496-4, 2000 [15]. To be called an ISO/IEC 14496-3 audio decoder, the decoder shall provide an output such that the RMS level of the difference signal between the output of the decoder under test and the supplied reference output (both normalized to be in the range between -1 and +1) is less than $2^{-15}/\sqrt{12}$. In addition, the difference signal shall have a maximum absolute value of at most 2^{-14} relative to full-scale.

7. CONCLUSIONS

We presented several techniques for implementing a fast MPEG-4 AAC audio decoder that exploits modern Intel Pentium SSE instructions operating on floating point data. A detailed analysis of the reference software implementation is provided. Results shows how decoding performances can be improved especially for filter-bank processing and Huffman decoding. The optimized decoder proves to be almost five times faster then the reference implementation, while preserving its compatibility with the MPEG-4 standard.

8. REFERENCES

[1] ISO/IEC JTC1 SC29/WG11, "ISO/IEC FDIS 14496-3 Subparts 1, 2, 3, Coding of Audio-Visual Objects - Part 3: Audio," October 1998.

[2] M. Bosi, K. Brandenburg et al., "ISO IEC MPEG-2 Advanced Audio Coding," *Journal of Audio Enginnering Society*, vol. 45, no. 10, pp. 789–814, October 1997.

[3] M.A. Watson and P. Buettner, "Design and implementation of AAC decoders," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 3, pp. 819–824, August 2000.

[4] K.H. Bang, J.S. Kim, Y.C. Park, and D.H. Youn, "Design optimization of a dual MP3/AAC decoder," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, May 2002, vol. 4, pp. 3776–3779.

[5] K.H. Bang, J.S. Kim, Y.C. Park, and D.H. Youn, "Design optimization of MPEG-2 AAC decoder," *IEEE Transactions on Consumer Electronics*, vol. 47, no. 4, pp. 895–903, November 2001.

[6] J. Chen and H.M. Tai, "MPEG-2 AAC coder on a fixed-point DSP," *IEEE Transactions on Consumer Electronics*, vol. 45, no. 4, pp. 1200–1205, November 1999.

[7] K.S. Lee, Y.C. Park, and D.H. Youn, "Software optimization of the MPEG-audio decoder using a 32-bit MCU RISC processor," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 3, pp. 671–676, August 2002.

[8] ISO/IEC 13818-7, "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 7: Advanced Audio Coding," 1997.

[9] K. Brandenburg and O. Kunz and Akihiko Sugiyama, "MPEG-4 Natural Audio Coding," *Signal Processing: Image Communication*, vol. 15, no. 4, pp. 423–444, January 2000.

[10] Intel, "IA-32 Intel Architecture Software Developer's Manual," vol. 2, no. 245471, 2001.

[11] S. Thakkar and T. Huff, "Internet Streaming SIMD Extensions," *IEEE Computer Magazine*, vol. 32, no. 12, pp. 26–34, December 1999.

[12] J.P. Princen, A.W. Johnson, and A.B. Bradley, "Subband/transform coding using filter bank designs based on Time Domain Aliasing Cancellation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, 1987, pp. 2161–2164.

[13] P. Duhamel, Y. Mahieux, and J.P. Petit, "A fast algorithm for the implementation of filter banks based on 'Time Domain Aliasing Cancellation'," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, 1991, vol. 3, pp. 2209–2212.

[14] V.P. Rodriguez, "A radix-2 FFT algorithm for modern single instruction multiple data (SIMD) architectures," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, 2002, vol. 3, pp. 3220–3223.

[15] ISO/IEC, "Information technology – coding of audio-visual objects – part 4: Conformance testing," *ISO/IEC 14496-4*, 2000.