



Do Not Zero-Pute: An Efficient Homespun MPEG-Audio Layer II Decoding and Optimization Strategy

Patrick De Smet
Dep. of Telecommunications
and Information Processing
Ghent University, TELIN/TW07
Sint-Pietersnieuwstraat 41
B-9000 Ghent, Belgium
pds@telin.UGent.be

Hiệp Quang Luong
Dep. of Telecommunications
and Information Processing
Ghent University, TELIN/TW07
Sint-Pietersnieuwstraat 41
B-9000 Ghent, Belgium
hiep.luong@telin.UGent.be

Filip Rooms
Dep. of Telecommunications
and Information Processing
Ghent University, TELIN/TW07
Sint-Pietersnieuwstraat 41
B-9000 Ghent, Belgium
frooms@telin.UGent.be

Wilfried Philips
Dep. of Telecommunications
and Information Processing
Ghent University, TELIN/TW07
Sint-Pietersnieuwstraat 41
B-9000 Ghent, Belgium
philips@telin.UGent.be

ABSTRACT

In this paper we point out that the general principle “do not compute what you do not need to compute” can be applied easily and successfully within a MPEG audio decoding strategy. More specifically, we will discuss the problem of eliminating costly computation cycles being wasted at processing useless zero-valued data. Hence, the title: “do not zero-pute”. At first, this may all sound somewhat obvious or trivial. Indeed, this can be true in many cases, but experience gathered in various teaching related projects during several academic years has also lead us to believe the opposite. Moreover, a survey of the existing literature quickly reveals that the approach discussed below has not been investigated and documented properly. Although we will only illustrate our optimization approach by discussing the MPEG-audio layer II decoding process in detail, we hope the reader will be able to apply, extend, and implement the basic principles presented here within many other applications.

Categories and Subject Descriptors

F.2.m [Analysis of algorithms and problem complexity]: Miscellaneous; D.1.m [Programming Techniques]: Miscellaneous

General Terms

Performance, Design, Algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MM'04, October 10-16, 2004, New York, New York, USA.

Copyright 2004 ACM 1-58113-893-8/04/0010 ...\$5.00.

Keywords

MPEG audio, optimized decoder

1. INTRODUCTION

Generally speaking, multimedia standards are commonly based on some type of frame based “syntax”, i.e., a repetitive and conditionally templated data structure that can be used to describe an encoded bit stream. Typically, the body of a multimedia bit stream frame contains a set of “control bits” that indicate how the remaining set of condensed information bits are to be processed further. By decoding and interpreting the control bits one typically arrives at an array of data that needs to undergo some further signal processing procedure, e.g., a subband audio reconstruction filtering procedure, or a JPEG 2D- or audio M-DCT, etc. This signal reconstruction process is most often responsible for the larger part of the computational complexity of the entire decoding process. Obviously, many excellent optimization strategies have been devised in order to arrive at various memory and/or computationally efficient implementations of several specific algorithms. Unfortunately, optimizing some of the basic algorithms typically involves designing and implementing more complex control logic. Very often this may not yield any real problem, but in some cases a straight-forward search for conceptually more simple optimizations, i.e., in our case the zero-pute paradigm, may also yield some very interesting results that can also be combined with other more complex optimizations.

Up to this point, we, and many other authors with us, have implicitly assumed that signal processing algorithms are always operating on a fixed-width array of meaningful data. In principle, this assumption would make the data structures and the many functions that may depend upon them, simple and efficient to be implemented. However, in many situations a relatively large part of the data structures may not contain any real or useful decoded data, e.g., large chunks of the array may be zero-filled (by default, or even by an inefficient encoder!). Obviously, it makes no sense to

drag along, e.g., multiply and/or add, this zero-valued data; one should try to remove this data from the computational process as early as possible: do not zero-pute.

It is important to note that the zero-pute reduction strategy has already been experimentally implemented and used for various applications, e.g. in [1] and [2]; an important goal of this paper is to return some renewed, more complete, and more formal attention to this optimization principle. Despite the emphasis of this paper on formal results, a package containing the source code implementing all techniques and formulas discussed below is freely available from the main author. Preliminary simulation results on a 200 Mhz Pentium MMX processor have already confirmed the more formal results reported below.

Let us now illustrate how we can apply the zero-pute reduction strategy within the implementation of an optimized MPEG-audio decoder. Note that while several excellent papers have documented important optimizations for building efficient decoders, the issues discussed below seem never to have been reported on fully. Additionally, please note that although we will focus on a layer II decoder, layer I and layer III, a.k.a. "MP3", also require the subband synthesis filtering step discussed below. The layer II standard is a crucial component of the DAB (Digital Audio Broadcasting) standard.

2. MPEG-AUDIO LAYER II DECODING

The MPEG layer II audio coding scheme operates by representing the input audio signal as bit stream encoded sets of samples of 32 equally spaced subband signals. Figure 1 illustrates the general layer II syntax layout. Each frame encodes 1152 samples. The header encodes synchronization and general parameter information (mono/stereo/dual channel, bit rate, sampling frequency, etc.). The CRC provides optional error detection. The bit allocation information (BIT ALLOC) points out which subband signals have been encoded and indicates their respective quantization accuracy. The scale factor side information (SCFSI) indicates how many scale factors (SCALEFAC) are encoded. The scale factors are used to rescale the normalized and quantized samples to their original value range. The ANC field can be used to fill up the frame size to its appropriate length, or to encode other "user defined" information (e.g. song information etc.). At the encoder side, compression is achieved by using a so-called psycho-acoustical model that enables the bit allocation routine to roughly quantize, or even remove, the signal components that are less relevant perceptually. An excellent tutorial on MPEG-audio coding can be found in [3].

Let us now consider the decoder side. For each multirate filtering step at time t , the basic ISO synthesis subband filtering algorithm starts with $M = 32$ subband samples extracted from the encoded bit stream syntax, i.e., $S_{sb}[i;t]$, $i = 0, 1, \dots, M-1$, and produces M PCM audio output samples; $S_{pcm}[j;t]$, $j = 0, 1, \dots, M-1$. First, the intermediate IPQMF (inverse polyphase quadrature mirror filter bank) vector $V[i;t]$ is shifted over $2M$ memory positions according to: $V[i;t] = V[i-2M;t-1]$, for $i = 2PM-1, 2PM-2, \dots, 2M$ with $P = N/M = 16$ and $N = 512$ the length of the prototype filter window. Then, the basic MPEG audio matrixing operation [6] is calculated:

$$V[i;t] = \sum_{k=0}^{M-1} S_{sb}[k;t] \cos(\pi(2k+1)(M/2+i)/2M), \quad (1)$$

for $i=0, 1, \dots, 2M-1$.

A N -value intermediate vector $U[i;t]$ can then be built using both $U[2Mk+l;t] = V[4Mk+l;t]$, $U[2Mk+M+l;t] = V[4Mk+3M+l;t]$, for $k = 0, 1, \dots, P/2-1$, and $l = 0, 1, \dots, M-1$. Finally, the so-

called windowing component computes M new PCM audio output samples, i.e., $S_{pcm}[j;t] = \sum_{i=0}^{P-1} U[j+Mi;t] D[j+Mi]$, $j=0, 1, \dots, M-1$, in which the values D are the N synthesis window coefficients (see Table B.3 in [6]). This algorithm requires $2M^2 + MP = 2560$ multiplications and $2M(M-1) + M(P-1) = 2464$ additions. Obviously, several optimizations can be used to realize important speed-ups of Eq. (1), but this always involves more complex control logic. This will be discussed further below; see Eq. (4).

3. STATIC CUT-OFF OPTIMIZATIONS

Now, examining several layer II encoded bit streams easily reveals that any real bit stream normally does not contain meaningful samples for all 32 subbands. Let us investigate this in somewhat more detail.

While the standard was being drafted, it became clear that (at low bit rates) high frequency subbands would rarely (need to) be encoded. Hence, the standard contains a set of tables that specify a hard subband cut-off index for each possible combination of the audio sampling rate and the encoded bit rate; e.g. at 32 kbit/s encoder output for a CD-quality 44.1 kHz audio input, the subband cut-off index is 8; see Table B2.c in [6]. The cut-off index is called the subbandlimit ("sblimit" in [6]); no subbands above it are allowed to be encoded. We will call this behavior "static" or "sblimit cut-off".

A simple standard decoder will zero-fill any subband sample values above the cut-off index, and use Eq. (1) with its full subband range to reconstruct the audio signal. Hence, if we drop this zero-valued subband range and reformulate Eq. (1) as

$$V[i;t] = \sum_{k=0}^{sblimit-1} S_{sb}[k;t] \cos(\pi(2k+1)(M/2+i)/2M) \quad (2)$$

we can realize an important optimization.

Table 1 indicates the static subbandlimit cut-offs defined within the MPEG-1 standard and the worst-case computational complexity of the subband synthesis matrixing procedure based on Eq. (2). You can compare these results with the $2M^2 = 2048$ multiplications and the $2M(M-1) = 1984$ additions of Eq. (1).

4. DYNAMIC CUT-OFF OPTIMIZATIONS

Next, even if we consider these predefined cut-offs, a normal encoder will still choose to allocate most of the bits to the subbands (well) below the hard cut-off index. This is due to the perceptual properties of the human hearing system that are modeled in the psycho-acoustic component of the encoder. Hence, some subbands will not be assigned any quantization bits, and many zero-valued BIT ALLOC values, see Fig. 1, will be encoded. If a simple syntax decoding algorithm would again be used, all subband sample values for each of these zero-allocated subbands, would be set to zero, and the output of the standard synthesis filter would be computed using Eq. (1). Again, this would be highly inefficient. Using Eq. (2) we can improve upon this situation, but even then we can easily do better: for each decoded frame, we can determine the largest subband index that has been assigned any quantization bits and use this index as a new subband cut-off index; i.e. we can use the simple algorithm

```
/* determine alloc_cut, i.e. locate Y!=0 in:          */
/* xxxxxxxxY0<-00...00 BIT ALLOCAtion array        */
for (chan_i=0; chan_i < stereo; chan_i++) {
  for (j_pds = sblimit; j_pds > 0; j_pds--)
    if (bit_alloc[chan_i][j_pds] != 0) break;
  alloc_cut[chan_i] = ++j_pds; /* increment:*/
} /* so we can use "< alloc_cut" testing elsewhere */
```

HEADER	CRC	BIT ALLOC	SCFSI	SCALEFAC	QUANT. SUBBAND SAMPLES	ANC
--------	-----	-----------	-------	----------	------------------------	-----

Figure 1: MPEG-1 syntax structure (layer II).

Table 1: Upper limits for the MPEG layer II subband synthesis computational complexity using the proposed static-sblimit cut-off optimization approach

ISO standard	Table B.2a	Table B.2b	Table B.2b	Table B.2d
sblimit	27	30	8	12
max. complex. $\times$	1728	1920	512	768
max. complex. $+$	1664	1856	448	704

and an adapted formula

$$V[i;t] = \sum_{k=0}^{\text{alloc_cut}-1} S_{sb}[k;t] \cos(\pi(2k+1)(M/2+i)/2M) \quad (3)$$

to realize another speed-up. Note that due to their simplicity neither the cut-off determination algorithm, nor the adapted formula, introduce any real additional complexity.

Obviously, this cut-off for a specific frame at a given instance in the bit stream will vary, and depend on the perceptual content of that frame. Hence, we will call this behavior “dynamic cut-off”.

5. INITIAL OPTIMIZATION RESULTS

Table 2 illustrates some average decoded cut-off subband indexes, i.e., $\mu[\text{alloc_cut}]$, determined by decoding an example audio bit stream using an adapted version of the pseudo code given above and the ISO codec (coder-decoder) from [7]. The computation of the values $V[i;t]$ in Eq. (3) requires $2M\text{alloc_cut}$ multiplications and $2M(\text{alloc_cut} - 1)$ additions; Table 2 also indicates the complexity computed using the average (over all frames) for the dynamic cut-off values and the corresponding static cut-off results. Again, please note that Eq. (1) required $2M^2 = 2048$ multiplications and the $2M(M-1) = 1984$ additions. Despite the conceptual and mathematical simplicity of these proposed optimizations they are most often glanced over. Instead, significantly more complex optimization strategies are often pursued. These more complicated optimization strategies will yield important improvements in terms of computational complexity, but they can do so only by also significantly increasing the time needed to study, program, verify and test their required adaptations!

6. COMBINATION WITH ADVANCED OPTIMIZATION STRATEGIES

Let us now investigate the cut-off optimizations discussed above and show that it can still be combined with more complex existing optimizations. As already indicated earlier, in real-life situations it can be inefficient to implement the matrixing procedure as formulated in all three equations given above; it is a well-known fact that Eq. (1) can be simplified based on the properties of the cosine function; see also [4], and [5].

If, for $l = 0, 1, \dots, M/2 - 1$, we let the values SDS (Sum/Difference Symmetry) reflect the symmetry due to the factor $(2k+1)$ in the cosine argument of Eq. (1), one can easily deduce that with the values $SDS_0[l;t] = S_{sb}[l;t] + S_{sb}[M-1-l;t]$ and $SDS_1[l;t] =$

$S_{sb}[l;t] - S_{sb}[M-1-l;t]$, we can also use

$$V[i;t] = \sum_{k=0}^{M/2-1} SDS_{(i \bmod 2)}[k;t] \cos\left(\frac{\pi}{2M}(2k+1)\left(\frac{M}{2}+i\right)\right) \\ \text{with } \{V[i;t] = -V[M-i;t], i = 0, 1, \dots, M/2-1\}, \\ \{V[M+i;t] = V[2M-i;t], i = 1, 2, \dots, M/2-1\}, \\ V[M/2;t] = 0, \text{ and} \\ V[3M/2;t] = - \sum_{k=0}^{M/2-1} SDS_0[k;t], \quad (4)$$

to implement the subband matrixing procedure. These partially optimized matrixing formulas require only $(M-1)M/2 = 496$ multiplications and $M + (M/2-1)(M-1) + (M/2-1) = M^2/2 = 512$ additions to compute the $V[i;t]$ matrixing results. Please note that, as already indicated above, using these existing optimizations requires a considerable increase in the amount of computational control and sequencing logic (further optimization would increase this even more so).

Now, implementing the proposed zero-pulse reduction and cut-off principles within Eq. (4) means that first we have to discriminate between two cases (with $M/2 - 1 = 15$):

1. $M \geq \text{alloc_cut} > M/2$: in this case we can only realize an adaptation of the SDS computation in which we can omit some additions and subtractions; i.e. $SDS_0[m;t] = SDS_1[m;t] = S_{sb}[m;t]$ for $m = 0, \dots, M - \text{alloc_cut} - 1$, and for $m = (M - \text{alloc_cut}), \dots, M/2 - 1$ nothing changes, i.e., we can use the original SDS formulas given above. Thus, the matrixing process then requires only $(M-1)M/2 = 496$ multiplications and $M - 2(M - \text{alloc_cut}) + (M/2-1)(M-1) + (M/2-1) = 448 + 2 \text{ alloc_cut}$ additions.
2. $\text{alloc_cut} \leq M/2$: first, note that the SDS values no longer need to be computed, i.e., $SDS_0[m;t] = SDS_1[m;t] = S_{sb}[m;t]$. Next, we can adapt the formulas for $V[i;t]$ and $V[3M/2;t]$ to only iterate their index k from 0 to $\text{alloc_cut} - 1$. This reduces the computational complexity to $(M-1)\text{alloc_cut} = 31 \text{ alloc_cut}$ multiplications, and $(M-1)(\text{alloc_cut} - 1) + (\text{alloc_cut} - 1) = 32(\text{alloc_cut} - 1)$ additions.

Although these optimizations only yield a relatively small improvement when compared to the adaptation from Eq. (1) to Eq. (4), they are certainly still worth the minor increase in complexity they require, especially for the lower bit rate range.

Table 2: Static-sblimit and average dynamic- $\mu[\text{alloc_cut}]$ subband cut-off index and computational complexity as a function of the bit rate (song: U2: “40”, $f_s = 44.1$ kHz, 6123 frames, data shown here: channel 0 only, encoder used: lsf, psycho-acoustic model 2).

	sblimit	$\mu[\text{alloc_cut}]$	average complex. $\times$		average complex. $+$	
			sblimit	alloc_cut	sblimit	alloc_cut
64 kbit/s	8	7,9	512	505,6	448	441,6
128 kbit/s	27	18,3	1728	1171,2	1664	1107,2
256 kbit/s	30	20,7	1920	1324,8	1856	1260,8

7. ADDITIONAL ADVANTAGES AND FUTURE WORK

Additionally, it is important to note that the cut-off optimizations discussed above always yield a scalable algorithm. If, e.g., the computational load would be too high to meet any real-time decoding deadlines, or if a system over-heating or low-battery condition would be detected, the subband cut-off parameter could be gradually adapted by forceably decrementing `alloc_cut`. The system would certainly still degrade down to a level of unacceptable behavior, but it would do so using a near continuous and soft degradation curve that makes these algorithms an ideal solution for various situations, e.g. in portable devices.

As a final extension, one could also look for zero-allocated subbands within the subband index range $[0, \dots, \text{alloc_cut}]$ as determined above. Note that this approach would require another increase in complexity (due to an index-dependent computation process that can only be implemented efficiently within a more complex hardware-based solution). An interesting and probably even more popular extension can also be realized within layer III of the MPEG standard: we are currently studying how to apply the zero-put reduction principle to the layer III aliasing, windowing and MDCT computation process.

8. CONCLUSIONS

In this paper we have discussed how using the seemingly simple “do not zero-put” design principle, important optimizations and design alternatives can be studied or used to realize efficient MPEG-audio layer II decoders. We proposed both a static strategy that could be hard-coded into an actual decoder implementation and a dynamic or “signal adaptive approach” that can realize even better speed-up and/or power-saving results. Both strategies can be combined with a well-known and very important existing optimization strategy, and the scalability of the algorithm was pointed out. The results reported here are also useful for devising efficient layer III (MP3) subband signal synthesis schemes, but these and several other adaptations are currently being studied further. Finally, incremental to [7], a package containing the source code for implementing all techniques and formulas discussed above is freely available from the main author.

9. REFERENCES

- [1] JPEG decoder source code from the Independent JPEG Group; <http://www.ijg.org/>
- [2] W.-K. Paik and S.-Y. Hwang. Design of a novel synthesis filter for real-time MPEG-2 audio decoder implementation on a DSP chip. *IEEE Trans. on Consumer Electronics*, 45(4):26–28, February 1999.
- [3] P. Noll. MPEG digital audio coding. *IEEE Signal Processing Magazine*, pages 59–81, September 1997.
- [4] K. Konstantinides. Fast subband filtering in MPEG audio coding. *IEEE Signal Processing Letters*, 1(2):26–28, February 1994.
- [5] P. D. Smet and I. Bruylant. Optimized recursive subband synthesis windowing for implementing efficient MPEG audio decoders. *IEEE Signal Processing Letters*, 10(10):303–306, October 2003.
- [6] ISO/IEC 11172-3, Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5Mbit/s, 1993.
- [7] ISO dist10 source code package available in `./mpeg2/software/` at: <ftp://ftp.tnt.uni-hannover.de/pub/MPEG/audio/>