Lossless and Near-Lossless Audio Compression Using Integer-Reversible Modulated Lapped Transforms

Henrique S. Malvar
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA

Abstract

We present a simple lossless audio codec, composed of an integer-reversible modulated lapped transform (MLT) followed by a backward-adaptive run-length/Golomb-Rice (RLGR) encoder. Its compression performance matches those of state-of-the-art predictive codecs, and it has the advantage that its compressed bitstream contains frequency-domain data that can be used for applications such as search, identification, and visualization. Its compression gain can be improved through a novel data model based on cross-block smoothed spectral magnitude estimates. Its bitstream can be transcoded into a lossy format, for transfers to portable players, at about twice the speed of other codecs. The codec also supports a near-lossless mode, which allows for an extra factor of two in compression without noticeable distortions.

1. Introduction

In the past decade the use of compressed digital audio has grown rapidly, and the popularity of the compact disc (the CD, which carries uncompressed digital audio) has dropped significantly. At the time of this writing, the top-selling CD player at Amazon.com doesn't make it into the top 500 best-selling items in consumer electronics, whereas four of the top ten items, including the top item, were portable digital audio players that mainly work with compressed audio.

Usually audio is compressed in lossy formats [1], to minimize file size or network bandwidth, with the bit rate determined by the required fidelity for the application. For example, Internet radio stations typically use MP3 at 128 kbps, or WMA or AAC at 64 kbps, which are appropriate for background music. For headphone listening, though, higher quality is usually desired, and many users prefer rates such as 192 or 256 kbps. For a home digital music library stored in a personal computer (PC) or file server, where significant more storage is available, many users prefer to compress audio to a lossless format [2], so that later playback on high fidelity equipment will have the same fidelity as that from the original CDs. Most lossless audio compressors today achieve a compression factor of about 2:1 for audio sampled at 44.1 kHz or 48 kHz.

In a typical scenario, a user wants to refresh the music collection stored in a portable digital audio player, especially for inexpensive devices that do not have enough memory to store the user's entire audio collection at a high enough fidelity. Even users that demand high audio quality prefer that music in the portable device be stored in a lossy compression format (typically at bit rates between 128 and 256 kbps), so that more tracks will fit within the device's

memory. The loss in audio quality is usually acceptable for the typical playback scenarios involving portable devices.

Unfortunately, for the file transfer scenario described above, the user is not well served with today's compressed audio formats. As each music file is transferred from the user's PC to the portable device, it has to be decoded from a high-rate lossy or lossless format and re-encoded into a medium rate using a lossy format. Since most devices support high-speed USB 2.0 connections, even with a modern PC the transfer time for each file is typically 80% for transcoding and 20% for the actual file transfer. Therefore, the user experience can be significantly improved if the lossless audio compression format can be quickly transcoded into a lossy format. Existing popular formats do not support fast transcoding, because lossless audio encoding typically uses adaptive time-domain prediction and adaptive entropy coding, whereas lossy encoding uses fixed or switchable-length time-frequency transforms, perceptual data weighting, and adaptive entropy coding.

In this paper we propose a simple architecture for lossless audio compression, using an integer-reversible modulated lapped transform (MLT) [3]–[6] and a backward-adaptive Golomb-Rice (RLGR) entropy coder [7]. Even without any context modeling, the proposed encoder achieves a compression ratio that is quite close to those of state-of-the-art encoders. The architecture supports fast transcoding into lossy formats and also a simple near-lossless mode with negligible transcoding time. With the addition of a data model based on inter-block spectral estimation, the encoder can achieve compressed file sizes within less than 1% of the best results reported to date, while maintaining fast encoding times.

In Section 2 we describe the proposed transform-domain lossless audio codec, and in Section 3 we describe how its performance can be improved via inter-block spectral estimation. In Section 4 we show that the proposed codec compares favorably with existing ones, and in Section 5 we discuss a simple near-lossless mode that may be appropriate for high-capacity digital audio players. We conclude by noting that the proposed lossless audio codec can provide a better user experience in typical scenarios.

2. Transform-domain lossless audio coding

Digital music libraries can become unwieldy large if stored in uncompressed form. A typical 4-minute stereo music track, when stored in raw CD format, occupies 42 megabytes, so that a 5,000 track library would need about 200 gigabytes. Even by modern standards, that's a considerable amount of storage. Thus, any amount of compression, even the typical 2x of lossless codecs, is usually welcome.

2.1. Predictive Coding

One of the early well-known designs for lossless audio compression is SHORTEN [8]. Although motivated by the need to compress digital speech files, it also works reasonably well with music data. SHORTEN decomposes the audio in short blocks (typically with 256 samples), and achieves a significant dynamic range reduction by the use of linear prediction (LP) or a low-order polynomial predictor. The prediction residuals are encoded with a Golomb-Rice (GR) encoder, because GR encoder approximate very closely the optimal Huffman coders

for sources with Laplacian probability distributions, which are good models for prediction errors [8]. In fact, GR encoders can perform very well for a family of generalized Gaussian distributions (of which the Laplacian is a special case) [7]. Each block in the compressed bitstream has a header area that stores an index to the kind of prediction used, the values of the prediction coefficients, and the value of the GR parameter, followed by the encoded residuals. Because the prediction and encoding parameters are pre-computed and then applied to the entire block, we say that SHORTEN uses forward-adaptive prediction and forward-adaptive entropy coding.

Shorten also supports a "near-lossless" mode where the samples in each block can be right-shifted by *n* bits, where *n* is adaptively changed from block-to block, to maintain specified signal-to-noise ratio per block. In this paper we borrow the terminology "near-lossless" to indicate lossy encoding at a high fidelity level (e.g. for imperceptible quality degradation).

The basic concepts behind SHORTEN – forward-adaptive prediction followed by forward-adaptive encoding – are still the basis of modern lossless audio codecs. One example is the popular FLAC [9] format, which typically leads to about 5% improvement over SHORTEN by supporting a larger set of predictors, and by dividing blocks into sub-blocks with different GR parameters for each. Another example is Monkey's Audio, which currently has the best performance on most comparison tests. According to [10], Monkey's Audio uses adaptive linear prediction and GR encoding, and [11] mentions that it also uses neural networks and range coding. Another modern codec that seems to be based in the same principles, but with more sophisticated cross-channel prediction for stereo data, is OptimFROG [12], whose compression performance usually comes quite close to that of Monkey's Audio.

The latest MPEG standard for audio coding supports two lossless formats: "Audio Lossless Coding" (ALS) [13] and "Scalable Lossless Coding" (SLS) [14]. ALS supports only lossless compression, whereas SLS supports a scalable-to-lossy mode, in which the bitstream includes an AAC lossy representation, for fast transcoding. ALS also uses forward-adaptive prediction (linear only) and forward-adaptive GR coding. ALS achieves some improvement in compression by encoding larger frames containing variable-size blocks, and by encoding small values of prediction residuals via Gilbert-Moore block codes [13]; unfortunately, those lead to a significant increase in computational complexity (see Section 4), while still not surpassing Monkey's Audio performance. We discuss SLS in the next subsection.

2.2. Transform coding

There are two disadvantages of using predictive coding for audio, even with the well-developed technologies described above. First, in many audio segments there are periodic tones, which cannot be efficiently predicted by low-order predictors. The use of very high order predictors would not be a feasible solution, because in short audio frames there is not enough data for reliable convergence of algorithms for finding optimal prediction coefficients, and the use of pitch predictors (as in speech coders) would not work well, either, because in music there are usually several simultaneous tones. Second, most lossy compression algorithms use a transform front-end, and thus there's no simpler way to transcode from lossless into lossy then full decoding of the samples and re-encoding.

Frequency-domain coding using fast transforms can solve both problems. If the audio frame has dominant tones, than most of the energy in the frequency domain is concentrated in a few transform coefficients, allowing for efficient compression. If the same transform that is used for lossy coding is also used for lossless coding, then fast transcoding can be achieving by just decoding the transform coefficients and re-encoding in lossy mode; thus, no transform computation is needed for transcoding. Transform coding has other advantages: the frequency-domain coefficients can be used directly for audio recognition, classification, and search, and summarization [15].

Transform coding can have the many desirable properties described above, but until a few years ago the use of transforms in lossless coding led to a significant loss in compression performance, as well as limited transcoding capability. That's because for lossless compression the transforms have to be exactly reversible in integer arithmetic. Early direct approaches for integer transforms replaced each 2x2 orthogonal factor by a lifting-based integer-invertible (or integer-reversible) module [16]. That works well for short-length transforms such as those used in image compression, but for larger transform lengths (e.g. 256 to 4096), the accumulation of rounding errors leads to a significant drop in lossless compression, or excessive noise in lossy compression.

Recently, new "matrix lifting" techniques have been developed for the computation of an integer-reversible modulated lapped transform (MLT, also known as modified discrete cosine transform – MDCT) [4], [5]. Even for large block sizes, those new methods can compute an integer MLT whose coefficient values are mostly within an error of less than +/–1from the real-valued MLT coefficients.

The MPEG-4 SLS uses such a low-noise integer MLT, of length 1024, 2048, or 4096 samples [14]. The MLT coefficients are sent to a "core" layer using an AAC encoder, and the transform-domain errors between the AAC encoded values and the integer MLT coefficients are encoded in bit planes with a hybrid Golomb coder (BPGC) and context-based arithmetic coder (CBAC). MPEG-4 SLS also has a "non-core" mode, in which the integer MLT coefficients are directly encoded via BPGC or CBAC. Several parameters that control the entropy encoders are transmitted in the header for each block. The compression performance of MPEG-SLS is quite close to that of Monkey's Audio.

2.3. Proposed coder

In this paper we propose a simple transform audio coder (STAC), using an integer MLT followed by an adaptive run-length Golomb-Rice (RLGR) entropy coder [7]. Its block diagram is shown in Fig. 1. The main difference from the coders discussed above is that the signal mapping block is fixed – an integer MLT of length 1024 – and the entropy encoder is backward-adaptive, with no parameter estimation during encoding. Each block is encoded independently, and for stereo signals the block header needs only one parameter value: a single bit indicating if the channels are encoded independently or after a mean/difference-like matrix computation.

For a stereo audio input, the encoder processes each channel into 50% overlapping frames with 2*M* samples. For each frame we compute an integer MLT with *M* subbands, via a matrix lifting algorithm [4], [5], to minimize rounding noise. In this paper we use $M = 1024$.The stereo

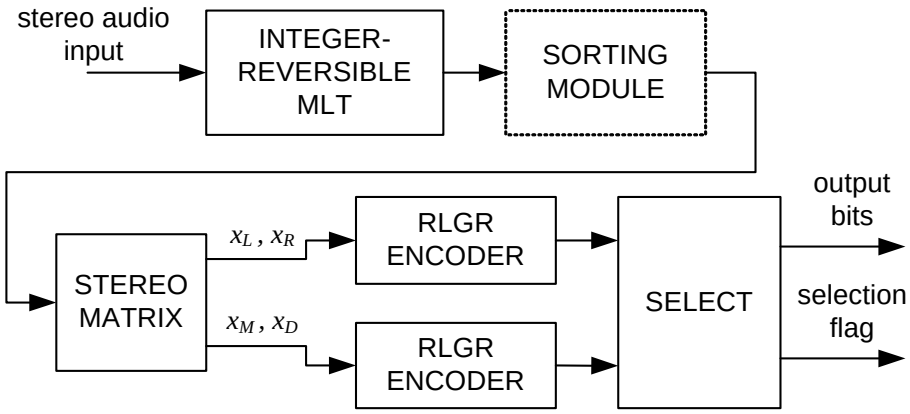

Figure 1. Simplified diagram of the proposed lossless simple transform audio coder (STAC); the sorting module is optional.

matrix module maps each pair $\{x_L, x_R\}$ of transform coefficients into a new pair $\{x_M, x_D\}$ of coefficients that carry mean and difference information, respectively. Instead of the usual mean-difference computation, we use the following lifting-based orthogonal approximation, which leads to a reduced dynamic range and thus slightly better compression:

$$\begin{aligned} x_D &= x_L - \left[\left(a\, x_R + Q \right) >> N \right] \\ x_M &= x_R + \left[\left(c\, x_D + Q \right) >> N \right] \\ x_D &= x_D - \left[\left(a\, x_M + Q \right) >> N \right] \end{aligned} \tag{1}$$

where the operations are computed in the order shown, N is a fixed shift parameter that should be set as large as possible but without leading to overflow, $Q = 2^{N-1}$, $a = \text{round}\{(\sqrt{2} - 1)Q\}$, and $c = \text{round}\{\sqrt{2}\, Q\}$.

We then encode each of the length-M vectors x_L, x_R, x_M, and x_D with our run-length Golomb-Rice (RLGR) encoder [7]. Unlike the GR encoders used in most lossless audio coders, as discussed above, our RLGR encoder is fully backward-adaptive, so no parameters from the input data need to be computed and added to the bitstream as side information. We expect the RLGR encoder to perform well, because it has a performance comparable to context-adaptive arithmetic encoding for sources with a generalized Gaussian (GG) probability distribution [7], which is a good model for the distribution of MLT transform coefficients [17]. The encoder then chooses the shorter of the encoded bitstreams for the pairs $\{x_L, x_R\}$ or $\{x_M, x_D\}$, and adds a flag bit indicating the choice.

In its simple mode, that's all there is to the encoder: fixed-length integer MLTs followed by RLGR encoders. That simple architecture allows for efficient code optimization in software implementations: the integer MLT can leverage existing work on fast FFT implementations [3]–[5], and the program code for the RLGR module is also easy to optimize, due to its simplicity. Using data parallelism and thread affinity, it's easy to divide the tasks equally

among two processors, if running on a dual-core machine (for a nearly 2x increase in encoding speed when compared to a single-core machine).

3. Improved compression via inter-block coefficient magnitude estimation

As discussed in Section 4, our simple codec described above is very fast for both encoding and decoding. In fact, it is significantly faster than lossy perceptual audio codecs. Since most encoding and player software modules and playback devices usually support both lossy and lossless compression formats, there will usually be room for increasing the complexity of the lossless codec, even if only for a small gain in compression.

For our coder we propose a novel data modeling strategy, which improves compression at a relatively small penalty in complexity. In this optional mode, the encoder and decoder compute a smoothed magnitude spectral estimator $x_S(k)$, where $k = 0, 1, \ldots, M-1$ is the frequency index. Calling $\{x_L(k), x_R(k)\}$ the MLT spectra of the current frame to be encoded, we map them into their sorted versions $\{\underline{x}_L(k), \underline{x}_R(k)\}$, as well as their corresponding versions $\{\underline{x}_M(k), \underline{x}_D(k)\}$, and encode those also via RLGR encoders. Thus, the encoder now has to choose among four encoded bitstreams {direct L-R, mapped M-D, sorted L-R, or sorted M-D}, so the selection flag now has two bits. The sorting indices are determined by sorting $x_S(k)$ in order of decreasing values; the goal is to map the original MLT vectors into vectors with a more rapid decay in magnitudes, which compress better. Note that no side information on the sorting indices is needed; the decoder can compute them because $x_S(k)$ is available at the decoder. For that, the encoder and decoder update $x_S(k)$ by the simple filtering equations:

- bi-directional smoothing:

$$\begin{aligned} u(k) &= \alpha\, u(k-1) + (1-\alpha)\sqrt{|x_L(k)|^2 + |x_R(k)|^2},\ k = 0,1,\ldots,M-1 \\ v(k) &= \alpha\, v(k+1) + (1-\alpha)\, u_L(k),\ k = M-2, M-1,\ldots,1,0 \end{aligned} \tag{2}$$

- spectral estimate update:

$$x_S(k) = \beta\, x_S(k) + (1-\beta)\, v(k),\ k = 0,1,\ldots,M-1 \tag{3}$$

The bi-directional smoothing equations run a left-to-right followed by a right-to-left first-order infinite impulse response (IIR) filter, with an effective zero phase response (and hence zero delay), controlled by the smoothing parameter α. The spectral estimate is updated via a first-order IIR filter controlled by β. In our implementation we found that for most audio tracks best compression is achieved with $\alpha = 0.25$ and $\beta = 0.55$. The computations in (2) are scaled so that they're performed in integer arithmetic. Note that for the decoder to run the update equations in (2), it needs the current spectral magnitude estimate $x_S(k)$, which assumes that all previous frames were decoded. Therefore, to allow for efficient seeking in the encoded bitstream, we reset $x_S(k)$ to predetermined values (e.g. $x_S(k) = M - k$) at regular intervals of L blocks. Thus, frames of L blocks can be independently decoded. In our implementation we choose $L = 94$, so that frames have a length of about 2 seconds at sampling rates of 44.1kHz or 48 kHz.

codec/ option track	Shorten	FLAC -5	Optim Frog	MAC -c2000	MAC -c5000	MPEG4 ALS -default	MPEG4 ALS -max	MPEG4 SLS noncore	STAC	STAC with sorting	STAC near lossless
avemaria	41.9	40.8	37.4	37.9	36.7	38.7	36.6	37.2	37.4	36.86	16.9
blackandtan	60.2	57.2	54.4	54.7	53.4	55.8	54.1	54.6	54.8	54.29	32.3
broadway	54.3	51.2	47.6	47.9	46.9	49.3	47.7	48.8	49.2	48.76	26.2
cherokee	56.9	54.5	52.1	52.4	51.8	53.3	52.1	52.6	52.7	52.42	31.2
clarinet	50.3	49.4	46.2	46.6	45.0	47.3	45.0	46.1	46.1	45.29	22.4
cymbal	33.1	29.5	25.9	28.7	27.4	27.5	25.7	29.0	27.2	26.28	20.8
dcymbals	66.3	61.8	59.5	60.0	57.7	60.9	59.2	60.2	60.1	59.40	49.0
etude	46.0	44.1	40.6	41.0	39.7	41.9	39.6	40.2	40.5	39.88	18.8
flute	43.9	42.4	39.0	39.8	36.9	40.2	36.7	38.0	39.0	37.47	20.0
fouronsix	50.3	48.3	45.2	45.6	44.3	46.3	44.9	45.7	45.7	45.25	28.4
haffner	58.3	57.3	53.9	54.4	52.7	55.0	52.5	53.8	53.7	52.91	28.1
mfv	34.9	33.4	28.7	29.2	27.8	30.1	27.8	28.2	29.0	27.98	9.4
unfo	56.4	53.6	50.3	50.9	48.6	51.7	49.3	50.0	50.3	49.30	29.3
violin	51.7	50.2	47.3	47.8	45.5	48.5	45.3	46.7	46.8	45.56	23.8
waltz	57.5	55.0	52.0	52.4	50.7	53.2	51.3	52.1	52.2	51.51	30.6
Average	50.8	48.6	45.3	46.0	44.3	46.7	44.5	45.5	45.6	44.9	25.8
Enc. time, seconds	5.6	29.2	23	7.1	56	11	433	–	12 *	20 *	13 *
Enc. speed, Mbytes/sec	15	3.0	3.7	12	1.6	7.9	0.2	–	7.2	4.3	6.6

(*) estimated

Table 1. Lossless compression ratios in %, encoding times, and encoding speed for the MPEG-4 48 kHz test set. The last column also shows the results for STAC in near-lossless mode.

4. Performance

We implemented our STAC codec as a Matlab script, with C modules for the MLT transform, RLGR encoding, and spectral magnitude estimation. In Table 1 we compare the compression performance and encoding times for STAC, with respect to other popular codecs: Shorten [8], FLAC [9], OptimFROG [12], Monkey's Audio (MAC [10]), MPEG-4 ALS [13], and MPEG-4 SLS [14]. For each codec, encoding options were chosen for either fast execution or highest compression, as indicated in Table 1. We ran our tests on a PC with a Core Duo 2.4 GHz processor, and the encoding times are for all 15 tracks in the MPEG-4 48 kHz stereo test set for lossless audio codecs, for a total of 450 seconds of audio. The encoding times for STAC are estimated; they include actual processing times for all modules except for the MLT, for which

the processing time is estimated by increasing by 40% the actual running time of our optimized MLT implementations (the 40% is based on estimates for integer versions of the MLT [4][5]). No code optimization to take advantage of dual-core was performed. For our STAC codec the decoding times are similar to the encoding times, because the decoder has to compute inverse MLTs (which take most of the processing time in simple mode). For the prediction-based decoders usually decoding times are much shorter, because the predictor coefficients can be read directly from the encoded bitstream and the predictor orders are usually very low (much lower than the MLT lengths).

For the entries in Table 1, we define the compression ratio, as usual, as the ratio of the size of the encoded bitstream to the original file size, in percent. We see that all codecs produce a similar performance, with compression ratios varying from 44.3% to 50.8%, that is, a reduction in files size from 1.97x to 2.25x. As it is typical with lossless codecs, small compression ratio improvements come with a significant penalty in speed. For Monkey's Audio, for example, changing from default to maximum compression modes increases complexity by ~ 8x, for a 3.8% reduction in compressed file size. For MPEG-4 ALS, a ~ 5% compression improvement comes with a jump in encoding time by almost 40x. For STAC, the sorting mode leads to a modest compression improvement of 1.6%, but with less than 2x increase in encoding time.

We see that STAC in simple mode achieves the same compression performance of the MPEG-4 SLS in "noncore" mode, slightly surpassing that of MPEG-4 ALS, at about the same encoding speed as MPEG-4 ALS. In sorting mode, STAC achieves a compression performance that's within 1.3% of Monkey's audio and 0.9% of MPEG-4 ALS in maximum compression, but STAC is about 2.8x faster than Monkey's and 22x faster than MPEG-4 ALS.

We should note that the relative results in Table 1 show very small differences among all codecs (except for Shorten and FLAC). In fact, those differences are not statistically significant; we have tested the codecs with several additional audio file sets, and the relative performances of the codecs do change slightly among different sets. Thus, we see that the main advantage of STAC over MPEG-4 ALS or Monkey's Audio (in default mode) is not a small gain in compression, but rather a frequency-domain representation that enables additional processing without full decompression, especially fast transcoding. For example, if music is ripped from CDs to a personal library in a predictive format and then transferred to a music player that uses a transform-based lossy format, the full encoder for the player format has to be run. If that encoder uses an MLT front-end, as it is the case for many formats, then transcoding from STAC would save on MLT computation time, which usually accounts for half the lossy encoding time. Thus, transcoding would be sped up roughly by a factor of two.

5. Near-lossless encoding

In some scenarios, true lossless encoding may not be needed. A 5,000-song music library takes about 100GB using lossless coding. That would not fit on a device with 50GB memory, for example, so a user may be willing to use lossy encoding for an additional factor of two in compression, as long as the encoding is "near lossless," that is, the errors are truly imperceptible. That would be very easy to achieve with lossy codecs such as MP3 or AAC, because at a compression factor of only 4:1 they produce a very high fidelity output, making them truly transparent. However, the high transcoding time would still be an issue.

With our STAC it is easy to achieve near-lossless encoding for an additional improvement of about 2xin compression. For that, during encoding we right-shift all transform coefficients of each block by *b* bits, where *b* is small enough so that quantization errors are not noticeable. For blocks with lower energy, it is important to reduce *b* to maintain a high signal-to-noise ratio. One way to achieve that is to vary *b* for each frame according to

$$\begin{aligned} \bar{b} &= B + \frac{1}{2}\log_2\left(\text{mean}\left\{x^2(k)\right\}\right) - \delta \\ b &= \min\left\{\lfloor B \rfloor, \max\left[\lfloor \bar{b} \rfloor, 0\right]\right\} \end{aligned} \tag{4}$$

where $\lfloor \cdot \rfloor$ denotes the floor operator, B is the quantization parameter that controls the maximum amount of shift for high-amplitude coefficients, and δ is parameter that controls how quickly b is reduced as a function of the block root-mean-square value. This strategy can be seen as an extension of the data shifting strategy used in SHORTEN [8], with the advantage that adaptive quantization (shifting) in the frequency domain produces much less noticeable noise than quantization in the time domain.

In Table 1 we also show typical results for STAC in near-lossless mode, obtained with B = 4.8 and δ = 11. The average compression ratio achieved for the MPEG-4 test set was 25.8%, that is, a reduction in file size of about 4x when compared with the original data, or about 2x when compared to the output of STAC in lossless mode. In all those cases, the decoded output was indistinguishable from the original, through informal listening tests with high-end headphones. Although formal evaluation tests were not performed, we measured the average segmental signal-to-noise ratio (SegSNR) to be about 50 dB, and the average noise-to-masking ratio (NMR) [18] to be about –5.1 dB; both are good indicators of near transparency.

In the scenario discussed above, assuming that the music library is stored in true lossless format with STAC in simple mode, transcoding to near-lossless can be done very quickly: for each block, the transform domain data is recovered by RLGR decoding, then all coefficients in the block are shifted right by *b* bits according to (4), and then re-encoded with RLGR (of course, no re-encoding is needed for blocks for which b = 0). For the data set in Table in, the time required for such transcoding would be on the order of 5 seconds on a Core Duo 2.4 GHz PC, which corresponds to a processing speed of about 17 megabytes/sec, which is about 5x faster than the typical data transfer speeds for digital audio players. Therefore, in that case the transcoding time is essentially negligible.

6. Conclusion

We have presented a simple transform-domain audio codec (STAC), which can operate in lossless or near-lossless mode. In lossless mode it achieves the same compression performance of state-of-the-art lossless audio codecs, and slight compression gains can be achieved via an inter-block spectral estimation and data sorting strategy. In near-lossless mode STAC can achieve an extra factor of two in bit rate reduction, while maintaining perceptual transparency. The codec's simplicity comes from the use of a fixed integer modulated lapped transform (MLT) and simple to implement backward-adaptive run-length/Golomb-Rice (RLGR) entropy coders [7]. Compression in the transform domain allows the bitstream to be quickly decoded to

obtain frequency-domain data, which can speed up applications such as search, identification, visualization, and transcoding.

References

[1] Wikipedia: "Audio data compression," available at: http://en.wikipedia.org/wiki/Audio_data_compression.

[2] Hydrogen Audio: "Lossless comparison," available at http://wiki.hydrogenaudio.org/index.php?title=Lossless_comparison.

[3] H. S. Malvar, *Signal Processing with Lapped Transforms.* Norwood, MA: Artech House, 1992, Chapter 5.

[4] H. Huang, S. Rahardja, "Integer MDCT with enhanced approximation of the DCT-IV," *IEEE Trans. on Signal Processing*, vol. 54, pp. 1156–1159, Mar. 2006.

[5] J. Li, "Low noise reversible MDCT (RMDCT) and its application in progressive-to-lossless embedded audio coding," *IEEE Trans. on Signal Processing*, vol. 53, pp. 1870–1880, May 2005.

[6] J. Li, "A progressive to lossless embedded audio coder (PLEAC) with reversible modulated lapped transform," *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, Hong Kong, vol. III, pp. 221–224, Apr. 2003.

[7] H. S. Malvar, "Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian sources with unknown statistics," *Proc. Data Compression Conf.*, Snowbird, UT, pp. 23–32, Mar. 2006.

[8] A. J. Robinson, "SHORTEN: simple lossless and near-lossless waveform compression," Tech. Rep. CUED/F-INFENG/TR.156, Cambridge University Eng. Dept., Dec. 1994.

[9] J. Coalson, "FLAC - Free Lossless Audio Codec," available at http://flac.sourceforge.net.

[10] M. T. Ashland, "Monkey's audio: a fast and powerful lossless audio compressor," available at http://www.monkeysaudio.com.

[11] Hydrogen Audio: "Monkey's Audio," available at http://wiki.hydrogenaudio.org/index.php?title=Monkey's_Audio

[12] F. Ghido, "Ghido's data compression page," available at http://www.losslessaudio.org.

[13] T. Liebchen and Y. Reznik, "MPEG-4 ALS: an emerging standard for lossless audio coding," *Proc. Data Compression Conf.*, Snowbird, UT, pp. 439–448, Mar. 2006.

[14] R. Yu, S. Rahardja, L. Xiao, and C. C. Ko, "A fine granular scalable to lossless audio coder," *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 14, pp. 1352–1363, July 2006.

[15] C. J. C. Burges, D. Plastina, J. Platt, E. Renshaw, and H. S. Malvar, "Using audio fingerprinting for duplicate detection and thumbnail generation," *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, Philadelphia, PA, vol. III, pp. 9–12, Mar. 2005.

[16] T. Krishnan and S. Oraintara, "Fast and lossless implementation of the forward and inverse MDCT computation in MPEG audio coding,' *Proc. Int. Symp. Circuits and Systems*, Scottsdale, AZ, vol. II, pp. 181–184, May 2002.

[17] R. Yu, X. Lin, S. Rahardja, and C. C. Ko, "A statistics study of the MDCT coefficient distribution for audio," *Proc. IEEE Int. Conf. on Multimedia and Expo*, Taipei, Taiwan, vol. 2, pp. 1483–1486, June 2004.

[18] K. Brandenburg, and T. Sporer, T "NMR and Masking Flag: Evaluation of quality using perceptual criteria," *Proc. 11th Int. AES Conf.*, Portland, OR, pp. 169–179, May 1992.