Lossless Audio Compression: A Case Study

Cristobal Rivero
cristobalrivero@gmail.com

Prabhat Mishra
prabhat@cise.ufl.edu

CISE Technical Report 08-415
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA.

August 7, 2008

Abstract

Audio compression is used everywhere – largely due to the rise of the internet, computers, and embedded systems. Since original, uncompressed digital recordings are very large in size, efficient ways are needed to compress large audio signals into small, high-quality, convenient formats. While lossy audio formats, like MP3, are widely popular in many everyday applications, lossless formats are also important to retain the original audio signals. It is feasible to perform lossless compression due to decreasing memory cost and increasing internet bandwidth. This paper investigates improvements to the Free Lossless Audio Codec (FLAC), one of the best lossless audio formats, by conducting tests on nineteen quality benchmarks. We have also studied lossless compression techniques from other domains and applied dictionary-based encoding for audio compression. The results show that dictionary-based compression is not useful, while modifying the blocking size in FLAC shows minor improvements in compression efficiency.

1 Introduction

In recent years, digital audio has become extremely popular. Through the widespread use of high-speed internet, consumers can download and transfer audio files. This is largely due to the effect of audio compression – specifically, lossy audio compression. Lossy audio compression techniques encode an original audio file (usually a .wav file or .aiff file) by removing information that is not audible to human ears. The goal of lossy compression is to reduce the file size (estimated 9:1) with minor or no quality loss. However, the original file can never be recovered from a lossy compression format.

In order to keep an exact copy of the original file, lossless audio compression must be used. The lossless audio formats are not as well-known as the lossy formats simply because they consume more memory. However, they are highly effective at reducing the size of original audio files. For example, an uncompressed Wave (.wav) file, three minutes long, can take 30.3 Mb of memory, too big to be sent by e-mail. Using lossless audio compression, it can be reduced to 50 – 60% of its original size, without any quality loss and it is always possible to obtain the exact original file if needed. Since memory cost continues to decrease, it is becoming increasingly popular for music producers, engineers, and audiophiles to keep their original, high-quality audio collection through lossless compression.

This paper studies existing lossless compression techniques to see if it can be improved directly or by employing compression techniques from other domains. One of the best lossless audio compressors is Free Lossless Audio Codec (FLAC). It has very fast encoding and decoding, competitive compression, extensive hardware support, and is available in open-source (original source code) – which makes it the current best lossless audio codec. In fact, the European Broadcasting Company adopted FLAC as its format for distributing audio over the Euroradio Network [4]. A method from the embedded systems domain is dictionary-based compression. It provides high compression efficiency with low decompression cost. The goal was to investigate dictionary-based compression with FLAC to see if FLAC could be improved. If not, is there a way to enhance dictionary-based compression for audio by borrowing ideas from FLAC?

The rest of the paper is organized as follows. Section 2 presents background information on FLAC and dictionary-based compression. Section 3 describes our approach followed by a case study in Section 4. Finally, Section 5 concludes the paper.

2 Background

2.1 Free Lossless Audio Codec (FLAC)

FLAC originated in 2000, originally developed by Josh Coalson. In order to compress audio it uses a four stage method: blocking, inter-channel decorrelation, prediction, and residual coding [1]. The input to FLAC is an uncompressed audio file, either Wave or Aiff. The blocking stage divides the audio signal into blocks or portions of a specified size. This directly affects the compression ratio. If the block is too small, the total number of blocks will increase, wasting bits on encoding headers. If the block size is too large, the signal could vary too much for the predictor to make an accurate prediction [1]. Thus, choosing the right block size is essential for good compression.

The inter-channel decorrelation stage (or mid-side conversion) is performed by removing redundancy in the stereo signals' left and right channels. Often, the left and right channels of a stereo signal are very similar. Thus, mid-side conversion was devised to reduce the amount of bits it takes to store the left and right channels. By encoding the left and right channels into a middle channel (left and right average) and a side channel (left minus right), the amount of bits needed to store the signal can be reduced. In cases where the left and right channels are very different, it can be passed without any decorrelation.

The prediction stage is essential for providing good compression. It is dependent on the efficiency of the block size chosen in the first stage. By using linear prediction and run-length encoding, FLAC predicts how each block could be most closely modeled. For example, if the current block resembles a sine wave, then

FLAC would ideally choose a sine wave to model the signal in that block. Many times, the prediction takes less memory to store than the original signal. This leads to the final encoding step, residual coding.

The residual coding stage (or error coding) is where the difference between the prediction and the original signal is computed and compressed. For example, if the predictor concludes that the current block resembles a sine wave, the original signal would likely not resemble an exact sine wave. The difference between the predicted signal and the original signal must be recorded in order for FLAC to be lossless. In this way FLAC preserves the exact original signal and reduces the file size. Once the difference is computed, FLAC uses Rice coding to compress the error signal from the prediction stage [1]. For future improvements, FLAC has reserved space for other coding methods, which leaves room for FLAC to be explored and enhanced.

2.2 Dictionary-based Compression

Dictionary-based compression can use either a static or a frequency-based dictionary. For our experiments, a 32-bit, frequency-based dictionary was used in order to encode signals. Figure 1 shows an illustrative example to describe the working principle of dictionary-based compression.

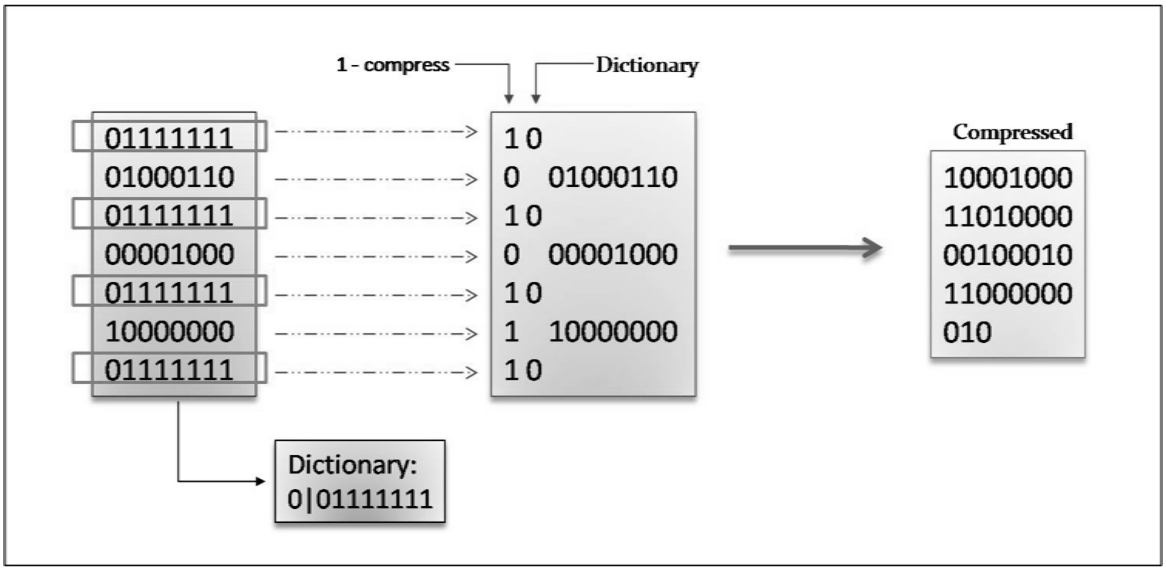

Figure 1 – Dictionary based compression

As Figure 1 shows, the dictionary finds the most repeated bit-words in the file and stores those words in a dictionary. For example, in this case, "01111111" is repeated four times and stored in the dictionary. Then, it encodes the signal by placing a '1' if it is to be compressed and a '0' if the word is to be left uncompressed [2]. In case of compression, it stores the dictionary index. For example, the first data is identical to the first dictionary entry, therefore, it is compressed using the first dictionary entry (index '0'). This can greatly reduce the size of the input files. This method provides good compression and has minor impact on decompression overhead.

3 Our Lossless Audio Compression Approach

Figure 2 shows the overall methodology of our approach. This paper investigates three different compression approaches. All of them begin with the use of original, uncompressed Wave files.

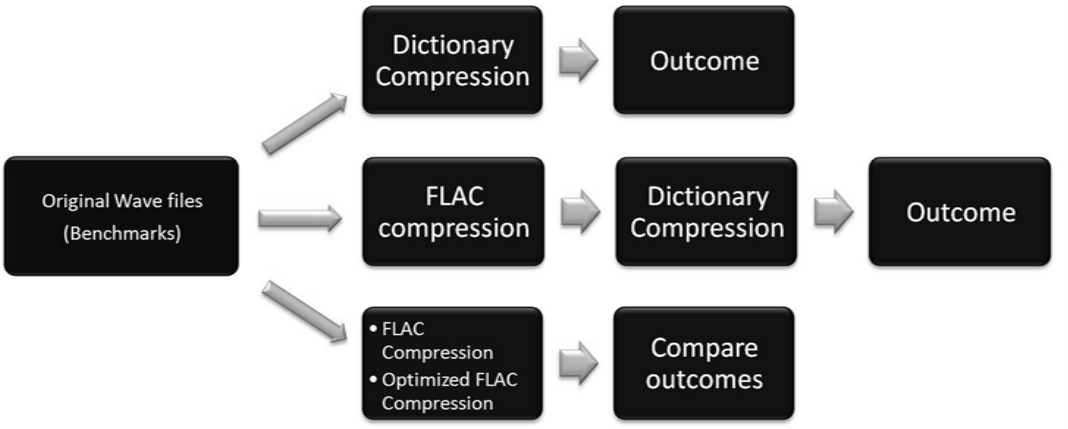

Figure 2 – Our audio compression methodology

First, dictionary-based compression was used on original Wave files to observe the compression ratio, as is shown in Figure 2. Since the dictionary compressor uses text files (*.txt) as inputs, the original Wave file had to be converted to 32-bit binary text words. The process is shown in Figure 3.

Figure 3 – Procedure for using dictionary-based compression

As shown in Figure 3, the hex values of the original Wave file were obtained by using Hex Editor Neo 4.51. Then, the hex values were transformed into 32-bit binary words in .txt format using a program that we developed. At this point, the file was ready to be used by the dictionary compressor. Once the Wave file was compressed, the ratio of the compressed file's size to the original file's size was recorded. Second, dictionary-based compression was performed on the compressed FLAC file, to test if any improvement was possible. As shown in Figure 2, the Wave file was first compressed using FLAC. Then, FLAC was compressed using dictionary-based compression (with the process shown in Figure 3). Lastly, we measure the compression efficiency of dictionary-based encoding as well as FLAC based compression. If the dictionary-based compression reduces the file size produced by FLAC, then the method will be beneficial.

Finally, we have tried various optimizations on FLAC. As shown in Figure 2, audio files were first compressed using the default highest compression values for FLAC. Then, the audio files were compressed again using an optimized version of FLAC, which we modified by varying block size and rice parameter optimization. Then, the two resulting files were compared. Even a small increase in compression would be beneficial, since the audio files being used are very large.

4 Experiments

The experiments were performed using nineteen Wave file benchmarks shown Table 1 [5, 6]. First, dictionary-based compression was used directly on uncompressed Wave files, to test their effectiveness in

compressing audio files. Second, dictionary-based compression was tested on FLAC files to test if further compression was available. Finally, we experimented with different parameters in FLAC for Wave files.

Table 1 – Various audio benchmarks [5, 6]

Dave Matthews	A Train	Beauty Slept	Chan Chan	Compilation 2
Compilation	Intense	Experiencia	Female_speech	Floor Essence
It Could be Sweet	Layla	Life Shatters	Macabre	Male Speech
Since Always	Hard	Toms Diner	Velvet	

4.1 Experimental Setup

A 32-bit, frequency-based dictionary compressor was used to test dictionary-based compression on FLAC and Wave files. Section 3 described this method in detail. FLAC was downloaded from [1] and compiled using Microsoft Visual Studio 2008. Once compiled, Wave files were compressed by FLAC with and without optimizations. Finally, the tests were run on nineteen benchmarks, chosen from various audio compression quality tests. They were purposely chosen to be in diverse genres and styles. Male and female speech was also tested. The estimated length of each file was 20 seconds.

4.2 Effect of Dictionary-based Compression on FLAC and Wave files

We wanted to test the effect of dictionary-based compression on Wave and FLAC files. Using the method described in Section 3, the dictionary compressor was first tested on Wave files. The results are shown in Figure 4 and Figure 5.

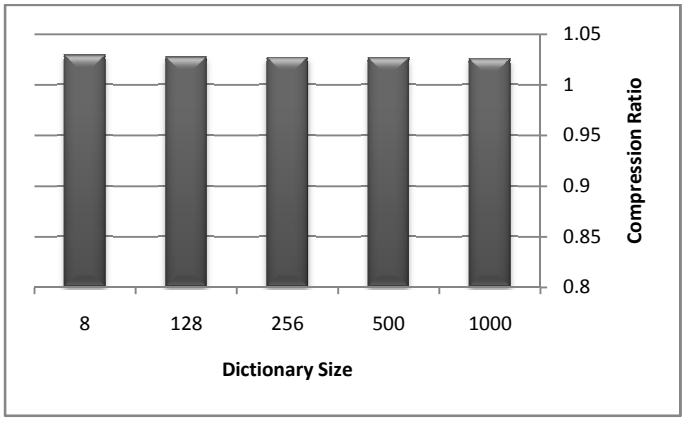

Figure 4 – Dictionary-based compression using 'Chan Chan' Wave file for different dictionary sizes

Note that "compression ratio" in this paper is defined by the ratio of the compressed file's size to the original file's size. Therefore, *smaller compression ratio implies better compression technique*. As is shown in Figure 4 and Figure 5, dictionary-based compression actually had an adverse effect on the original Wave file. The dictionary compressor was manually varied by dictionary size, in order to ensure good tests – nevertheless, the method was not successful at compressing Wave files.

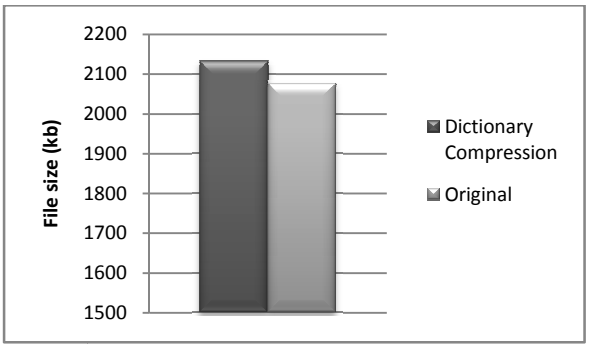

Figure 5 – Dictionary-based compression using 'Chan Chan' Wave file

Next, a FLAC file was compressed using dictionary-based compression. FLAC has initial metadata which was left out of dictionary compressor's text file because, in reality, the compressor would have to leave the initial metadata uncompressed in order for the FLAC header to remain intact. The results are shown in Figure 6 and Figure 7.

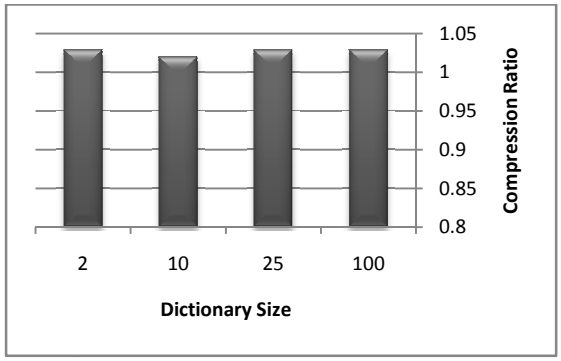

Figure 6 – Dictionary-based compression using 'Chan Chan' produced by FLAC

As shown in Figure 6 and Figure 7, dictionary-based compression adversely affected FLAC. Instead of compressing a portion of the file, it increased the file's size. The dictionary size had no effect in varying the compression ratio.

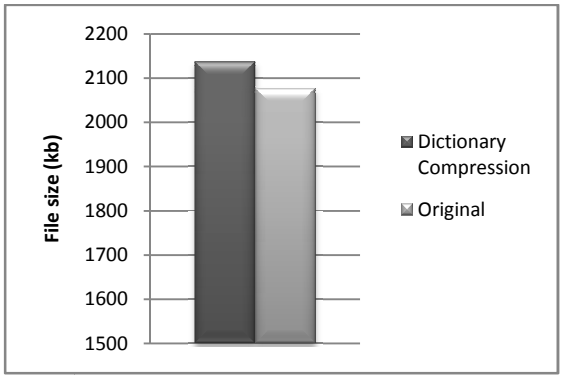

Figure 7 – Dictionary-based compression using 'Chan Chan' produced by FLAC

These were undesired results, since it showed that dictionary-based compression is not proficient at compressing either Wave or FLAC files. Wave and FLAC files are not repetitive enough to allow beneficial dictionary-based compression. The dictionary-based compression method must be modified to allow for compression of audio files. By extending the method to cover word mismatches, bitmasks, or an increment or decrement bit, the compression may perform better for audio files. Dictionary-based compression might be better suited to be used within the residual coding section in FLAC.

4.3 Optimization in FLAC

We wanted to test if FLAC could be improved by optimizing blocking and rice coding parameters. There is variation in compression that can occur by choosing different blocking sizes and rice coding parameters. For this experiment, the benchmarks were compressed using the default highest setting in FLAC (the default block size is 4096 samples). Then, different blocking sizes were tested to see if improvement was possible. Tests showed that blocks of 2048 samples provided better compression. For rice coding, FLAC has a built-in function that seeks the optimal rice coding parameters. The average results from the nineteen benchmarks are shown in Figure 8 and Figure 9.

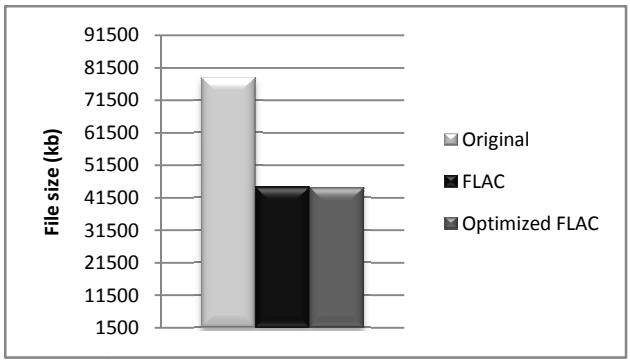

Figure 8 – FLAC optimization for nineteen benchmarks

As is shown in Figure 8 and Figure 9, the FLAC optimization only improved the compression ratio by 0.2%, which is not a significant increase. While it shows that at 2048-sample blocks, FLAC seems better at predicting the signal, it is not a notable increase to compel a change from the 4096-sample block standard for highest compression.

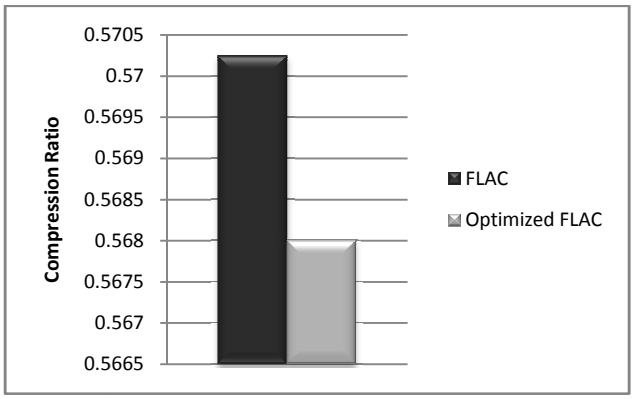

Figure 9 – Results of various FLAC optimizations

5 Conclusions

Lossless audio compression is gaining popularity and will continue as long as memory costs continue to decline and internet bandwidth increases. FLAC is a great lossless audio compressor, compressing original audio files an average of 57%, according to our tests (other high-quality lossless audio codecs are within 3% of FLAC [1], [3]). Improvement to the FLAC method will likely have to come from another domain, since optimizing FLAC from within provided only slight improvement. As shown in Section 4.2, dictionary-based compression was not able to improve the quality of FLAC. The likely place for improvement in FLAC is in the residual coding section by testing other error encoding means. Moreover, it may be possible to reduce the file size using bitmask-based compression since it tries to match data patterns that are not identical using bitmasks [2].

An interesting direction is to study FLAC in order to incorporate near-lossless audio compression. Near-lossless audio compression is the balance between lossless and lossy audio codecs. Its goal would be to provide the highest quality audio possible by taking away completely unnecessary characteristics of the audio signal. For example, many Wave files contain frequencies above 20 kHz, which humans cannot hear – and most cannot hear above 17 kHz. Performing this minimal psychoacoustic analysis would still be acceptable to musicians and audiophiles since the small quality loss is not noticeable to anyone. A near-lossless FLAC option would provide great quality and compression – highly attractive to musicians, audiophiles, and mainstream consumers.

6 References

[1] J. Coalson. *FLAC - Free Lossless Audio Codec*. 2008. http://flac.sourceforge.net/.

[2] S, Seong and P. Mishra. Bitmask-Based Code Compression for Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* , volume 27, no 4, pages 673-685, April 2008.

[3] M. Ashland. Monkey's Audio. http://www.monkeysaudio.com, 2008.

[4] *European Broadcasting Union*. http://www.ebu.ch/en/radio/ops_rdo/faq/index.php, 2007.

[5] Vorbis.com. Dare to Compare. http://www.xiph.org/vorbis/listen.html, 2005.

[6] Media Fire. Loss.7z, http://www.mediafire.com/?3xgdgokzhyt, 2008.