FLAC encoding using GPU acceleration

Yang Wang
Supervisor: Dr. Eric McCreath

COMP4560: Advanced Computing Project
Australian National University Semester 1, 2017
May 26, 2017
Acknowledgements

I wish to express my sincere thanks to Dr. Eric McCreath, my supervisor, for patiently providing me with all the necessary materials, answering my questions and guiding me through the process of this project over the past six months.

My thanks also go to Prof. Weifa Liang, the course convener, for valuable assistance in academic writing and in planning and creating presentations. I would like to take this opportunity to express gratitude to all the faculty members in the department for their help and support. I also thank my family members for their unceasing encouragement, and financial and spiritual support and attention. I am also grateful to my friends, Jiajin Huang and Qin Tian, who supported me throughout this venture.
Abstract

In digital audio compression processes, a lossless audio compression method is frequently used. This project is going to explore the FLAC lossless audio format. During the process of encoding digital audio signals, a 60-minute recording may take a few minutes to compress using a single thread CPU. However, the audio signal has been coded in parallel with a GPU, significantly improving its performance. The use of a GPU for encoding FLAC audio data has been explored and the efficiency of data compression has been improved by using CUDA programming in GPU. The parallelism in compression caused by implementing CUDA is data level parallelism. This means the input signal has been broken into several blocks or chunks of contiguous data, each of which is then encoded and compressed independently. The GPU encoding performance is directly determined by the number of threads and the size of input signals, following tests the speed of encoding was improved to the extent that it was 2.4 times faster than the original version.
Chapter 1

Introduction

The FLAC format is a lossless compression audio format under the compression algorithm of ac. The wave form music les can be compressed to 60-70 per cent of its original size without any loss of the raw signal, and is considered one of the most important methods of lossless compression. However, the encoding process of FLAC is implemented on the CPU; with the Dennard Scalling ceased, the clock frequencies of CPU will not be increased without raising the chip heat or the voyage leak, thus GPGPU computing is regarded as an alternative way to increase the compression efficiency of FLAC encoding. This project aims to explore the GPU implementation of the FLAC encoding process. GPU is a graphics processing unit which is normally used to accelerate the creation and operation of images. One of its great advantages is data parallelism. This will greatly improve the performance of computation when encoding the FLAC format. To improve the efficiency of computation and achieve lower energy consumption CUDA (Compute Unit Device Architecture) is implemented from the NVIDIA Corporation as a compression technique for the general purpose of processing.
Chapter 2

Background of FLAC format

This section provide the background introduction of the FLAC format.

2.1 History and introduction

The history of FLAC can be traced back to 2000 when it was first developed by Josh Coalson [1]. FLAC is an abbreviation for Free Lossless Audio Codec which is an audio format. This audio format can be compressed without any loss in quality [1]. FLAC is described as the fastest and most widely supported lossless audio codec, and the only one that is non-proprietary, unencumbered by patents, has an open-source reference implementation, a well-documented format and API, and several other independent implementations [2]. The process of compressing FLAC format normally refers to the conversion process which converts an analog signal to a digital signal using ADC (Analog-to-Digital Converter). The digital audio compressed by FLAC codec can be reduced to 50 to 60 percent of its original size [3] and generates an identical copy when it is decoded. When the audio signal is encoded in FLAC format, it can be used to record, store, process and manipulate sound.

2.1.1 The development of the lossless audio format

The lossless audio format is a mainstream audio format so-called because of to its high compression ratio which is similar to that of an MP3/OGG/AAC. However, given the decrease in audio quality during the process of lossy audio compression, it cannot meet the needs of lossless music files and high quality audio file playing. Thus, the need for lossless audio compressing becomes apparent. The following sections are going to discuss the advantages of lossless audio. These are:
• No signal is lost. The initial signal will be retained 100 per cent. The lossless audio compression format can preserve the entire data signal from WAV files. This can be tested using tools like UltraEdit. An audio copy of a music file is made and saved, first as a WAV file, then as an initial music file. The WAV file is then encoded into FLAC format before this FLAC file being encoded back into WAV format and UltraEdit tools being implemented. When compared to the 2 WAV files, the data signals will be the same.

• High audio quality will not be affected by the signal source. The audio quality of lossless compression format will be the same as the initial cd; however, the lossy compression format will have deficiencies when compared with the initial signal. This is due to the design of the lossy audio compression algorithm (normally it will not form part of the high frequency signal). Thus, owing to the limitation of the lossy compression format, when music is compressed with a high dynamic range, such as a symphony, the quality of the music will be less than satisfactory.

• Easy to transform. The lossless compression audio format can not only be decoded back to initial WAV files, it can also be encoded as a lossy compressed audio format such as an MP3 or AAC. This is another advantage compared to the lossy compression format, because converting the binary encoding process from a lossy compression format to another lossy format will result in loss of signals.

2.2 The encoding process

The Figure 2.1 indicates the total process of encoding part, the audio data compression technique of FLAC is included with following stages: Blocking, Interchannel Decorrelation, Prediction and Residual coding[1].

The blocking stage refers to the input into the encoder which is raw uncoded audio data divided up into several contiguous blocks [? ]. Normally, the size of blocks varies and the optimal size of block depends on many factors. To simplify the encoder design, the size of block in the FLAC ranges from 16 samples to 65535 samples [? ]. Once the blocks are generated, the blocked data will be transmitted by the sub-block. Each sub-block is independently coded into subframe, packed into a frame and then appended into the stream. During the inter-channel decorrelation stage, the input data are stereo type, the left channel and right channel correlate with each other which can then be used to compress data. By calculating the following formulas: mid=\((\text{left}+\text{right})/2\), side= left-right, the left, right and centre channels can be obtained [? ]. In the next stage, the FLAC models the
input signal by approximating it, using a mathematical function so that the result is much smaller than the original information. All these mathematical functions are available to the encoder and decoder. FLAC now uses 4 methods of modelling the input signal: (1) Verbatim, (2) Constant, (3) Fixed linear predictor and (4) FIR linear prediction. There are two methods of generating an approximate result: (1) Fitting a simple polynomial to the signal and (2) General linear predictive coding. Once the model is set up, the data residual signal also need to be encoded losslessly, then we enter the stage of residual coding. In the part of prediction part, FLAC has 2 methods of coding the error signal: one is Rice codes the entire residual by using the Rice parameter which is estimated by the encoder from the basis of the variance of the residual[1], the other one is encoder partition the residual into several sample length regions and code each part with its individual Rice parameter from the calculation of the region’s mean.

2.2.1 The basic principle of lossless audio encoding

The main encoding process involved in the lossless compression audio format comprises linear predictive coding, residual computations and entropy
encoding. As illustrated in the following graph, the input signal $s(n)$ will receive the estimated signal and estimated residual signal $s'(n)$ after it has travelled through the LPC model. It should then be implemented using optional entropy encoding algorithms to encode the estimated signal, and completes the data packaging process according to the different requirements of the data stream.

![Figure 2.2: Encoding](image1)

![Figure 2.3: Decoding](image2)

Figure 2.2 is a basic workflow of the lossless audio decoding, it can be considered as the reverse process of lossless audio encoding. The input data getting through the demultiplexer and then getting rid of the header information and frame information of the code stream, and send them to the decoder. Finally, the data, once decoded, will be subject to linear predictive decoding in to obtain the final PCM values.

### 2.2.2 The linear predictive calculation

The linear predictive calculation can be considered as an effective method to save more CD signals, it provides a set of concise predictive coefficients matrix[4] in order to create an autoregressive predictive model of the waveform for the compression process. The calculation will cost little, and the implementation of the predictive coefficient can decrease the size of the saved
audio signals and eventually restore the original audio file.

Normally, the higher the maximum LPC order, the slower and more accurate, the model will be [5]. Most of the compression algorithms will be implemented with some improved linear predictive model to remove the redundant data, and then generating a sequence of estimated residuals. The parameter of the predictor can be represented as the removed redundancy of the signal, the estimated residual and parameters of lossless encoder predictor can be combined together to represent the signal of each frame.

The basic principle underlying the linear predictive model is to make use of the correlation of the audio signal, which means the current value $x[n]$ is approximated from the value of former samples such as $x[n-1], x[n-2], x[n-3], \ldots$, (see formula 2-1) the more former samples we have, the higher approximation accuracy. The more former samples, the higher the approximation accuracy. The current sample value is then subtracted from the estimate sample value to gain the result $e[n]$ encoded. Formula 2-2 illustrates how is the residual generated and if the predictor performs well, the computed residual will be uncorrelated and acquire a flat frequency spectrum, the size of $e[n]$ will be smaller than $x[n]$ and it requires fewer bits of the digital representation[3].

$$x[n] = a_1 \times x[n-1] + a_2 \times x[n-2] + \ldots + a_M \times x[n-m] + e[n] \quad (2.1)$$

$$e[n] = x[n] - a_1 \times x[n-1] - a_2 \times x[n-2] - \ldots - a_m \times x[n-m] \quad (2.2)$$

### 2.2.3 Entropy Coding

Entropy coding is responsible for shifting the redundant part of the residuals and then begin encoding, during the process of encoding, no information will be lost, thus, entropy coding can be considered as a way of lossless compression method. Normally entropy coding has 3 types: rice coding, Huffman coding and range coding, but the flac compression codec will use the Rice coding[4].

Rice coding is a form of Huffman coding method which has a Laplacian (two-sided geometric) distribution, meanwhile Rice coding only has one parameter $k$ to match a signal’s distribution and then use this parameter to change as needed. The main step of Rice coding is comprised of 3 parts: (1) a sign bit (2) $k$ low-order bits (3) remaining high-order bits, the first part of each code is represented as the symbol of estimated residuals, the second part is is the $k$ least significant bits of the binary representation of the value of $e[n]$, and the third part is comprised of $N$ successive zeros, here $N$ means the binary representation of the $e[n]$’s $N$ remaining unused most significant
bits[3]. Here we take n = 576 (the binary representation of it is 1001000000) for example, assume k = 8

(1) sign, bit (0 means negative, 1 means positive ) = [1]
(2) n/2^k successive errors, n/2^k = 581/256 = 2 = [00]
(3) separate bit = [1]
(4) k least significant bits after n = [01000000]
(5) combine the above 4 steps: [1][00][1][01000000]

During the process of practical coding, the predictor of one frame data is selected from the minimal value of the subtraction of the practical value and predicted value, because this kind of algorithm is derived from the shift positions and addition without any multiplication, it is quite convenient to be implemented in different computing platform.

2.3 flac format introduction

2.3.1 FLAC format formal specification

Table 1 indicates the formal specification of FLAC format in Stream. All numbers used in FLAC bitstream are integers and no floating-point representations. As shown in table above, FLAC format specifies the ‘fLaC’ marker first then followed the mandatory metadata block and audio frames the numbers in bracket indicate how many bits are used for a given field. FLAC is provided with 128 kinds of different metadata blocks, the blocks can be assigned by their functions into 7 sections: STREAMINFO, APPLICATION, PADDING, SEEKTABLE, VORBIS COMMENT, CUESHEET and PICTURE.

STREAMINFO is responsible for the information of the whole stream, it includes the part of sample rate, number of channels, total number of samples, it begins first and will be skipped when the decoder can’t understand the content of it. [1] APPLICATION is responsible of the third-part applications. PADDING is responsible for user to enable the encoder to ask for a space of padding block to add those extra metadata[1]. SELECTABLE is responsible for the provide a table to contain seek points, and each stream should only have one SEEKTABLE[1]. VorbisComment is responsible for storing human-readable name/value pairs[1]. CUE SHEET has the function of backing up those CD-DA discs, and providing a mechanism of playback[1]. The block of PICTURE can store the information of pictures related to the music files.
2.3.2 The formal definition of FRAME

The FRAME is comprised of 4 parts: FRAME HEADER, SUBFRAME, padding part and FRAME FOOTER. As shown in Table2[1], the audio data has one or more audio frames, for the part of frame header, it includes the sync code, information of the block size in frame, sample rate and number of channels. To determine the start of the frame, the sync code begins first at each frame and won’t appear at anywhere else, to ensure a correct sync, the decoder will check the rest of the frame header have no invalid data, and also generating an 8-bit CRC in FRAME HEADER to compare with the other CRC which appears at the end of the frame header, the instance of sync code could be '11111110' [1]. On the other hand, considering the decoder have no direct access to the STREAM INFO, to have a knowledge of the information of the stream, the FRAME HEADER should also contain the information which appears in the STREAM INFO like sample rate, bits per sample, number of channels and so on.

2.3.3 The formal definition of SUBFRAME

The SUBFRAME is comprised of SUBFRAME HEADER, SUBFRAME CONSTANT, SUBFRAME FIXED, SUBFRAME LPC and SUBFRAME VERBATIM. In the stream, the subframe are appeared serially and coded separately, the decoder complexity are reduced dramatically due to the encode audio data are not interleaved[1]. After the the part of SUBFRAME HEADER, we have 4 methods of modelling the input signal: verbatim, constant, fixed linear predictor, and FIR linear prediction. The method of SUBFRAME CONSTANT include a n bit value of unencoded constant value of the subblock[1]. The method of SUBFRAME FIXED include n bit of value of unencoded warm up samples[1]. The method of SUBFRAME VERBATIM includes n times i bits of unencoded subblock.

2.3.4 The formal definition of RESIDUAL

The RESIDUAL is comprised of a 2 bit value of Residual coding method and RESIDUAL CODING METHOD PARTITIONED RICE [1]. The residual coding method has 2 models, one is partitioned rice coding with 4-bit rice parameter; the other one is partitioned rice coding with 5-bit rice parameter[1]. The RICE PARTITION is comprised of a 4 bit value partition order and RICE PARTITION.

2.4 GPU Architecture

The section will explore the architecture of GPU and introduction of cuda programming
Table 2.1: STREAM

<table>
<thead>
<tr>
<th>32</th>
<th>&quot;fLaC&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata Block</td>
<td>STREAMINFO metadata block</td>
</tr>
<tr>
<td>Metadata Block*</td>
<td>Zero or more metadata blocks</td>
</tr>
<tr>
<td>Frame</td>
<td>one or more audio frames</td>
</tr>
</tbody>
</table>

Table 2.2: FRAME[8]

<table>
<thead>
<tr>
<th>14</th>
<th>sync code '11111111111110'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reserved:[1]</td>
</tr>
<tr>
<td>1</td>
<td>Blocking strateg</td>
</tr>
<tr>
<td>4</td>
<td>Block size in inter-channel samples:</td>
</tr>
<tr>
<td>4</td>
<td>Sample rate</td>
</tr>
<tr>
<td>4</td>
<td>Channel assignment</td>
</tr>
<tr>
<td>3</td>
<td>Sample size in bits:</td>
</tr>
<tr>
<td>1</td>
<td>Reserved:</td>
</tr>
<tr>
<td>8</td>
<td>CRC-8</td>
</tr>
</tbody>
</table>

2.4.1 The introduction of GPU computation

The increasing demand for the realtime, high definition 3D graphic, and the programmable Graphic Processor Unit enables GPU become evolved in the highly parallel, multithreaded calculation as a revolutionary break through[6]. As a shared-memory parallel platform, by sharing the work of CPU, computation under the GPU implementation will become faster and less energy consuming[7], to be more specifically the GPU structure is well-suited to address and solve the problems which can be expressed and transmitted into data-parallel computations which have massive amounts of mostly independent calculations, data-parallel processing maps those data elements into the parallel processing threads, the application like audio encoding and decoding which need to process saved data set will speed up the computation after implementing the data-parallel programming model[6]. To make this process and computation become better, the NVIDIA Corporation introduce an implementation CUDA(Compute Unified Device Architecture) to implant this kind of parallel architect on GPUSs and offer support of APIs to develop parallel computations using C programming.
2.4.2 CUDA Architecture

Nvidia’s graphics card is considered as an extremely multithread computing architecture, the graphics card contains a set of parallel multiprocessors which are further divided into many cores and each core executes instructions form one thread at a time, all those threads have to execute the same instruction concurrently, CUDA will make the use of this massively parallel nature of GPU and contains one special C function called ‘kernel’ which is a set of C code can be executed on graphics card within fixed number of threads concurrently[8].

The Figure 2.3 illustrated the working principle of CUDA and GPU, firstly the processing data is copied from main memory to the memory for GPU, then CPU send instructions to instruct the processing, the data copied on the memory on GPU now is sent into desperate threads to execute parallel in each thread, the calculation result is copied back to the Main memory.

The hardware architecture of CUDA is followed with Grid and Block Architecture, the Grid is comprised of one-dimensional, two-dimensional or three-dimensional thread blocks, Figure 4.2.2 illustrates the a two-dimensional grid and block structure, each thread block then further divided into threads, all thread block is a set of threads running on processors, all threads of a single thread block can communicate with each other through thread memory, thus they are executed on the same multiprocessor, and the mechanism of synchronisation can be implemented in this way[8].
Figure 2.5: Grid and Block Structure
Chapter 3

Methodology and Frame work

This chapter provides an overview of the mathematical operation of modelling the input signal and residual coding, and the implantation of how to use CUDA to operate encoding operation on GPU.

3.1 Overview

![Overview Diagram]

Figure 3.1: Overview
The part will mainly go through the steps and process I take to combine data-parallel computation in CUDA with the FLAC encoding process. Firstly, I decode wav files from the source input, then decode the raw signal into samples. The decoded samples need to be assigned into blocks, each block will contain 4096 samples. I’ve implemented Linear Predictive Calculation algorithm to calculate the coefficients of a prediction autocorrelation formula, then the memory need to be allocated on GPU for the data of those samples and calculated coefficients. After that, we begin applying the LPC coefficients on each samples in parallel in thread of GPU, each thread is responsible for one group of samples to calculate, then we retrieve the calculated residual back from each thread the main memory of GPU, and finally send those data back to CPU.

3.2 The linear predictive calculation algorithm

3.2.1 Nature of linear prediction

The general introduction of linear predictive calculation algorithm has been introduced in chapter 2 before, here is the detailed description of the algorithm being implemented during the encoding process. Please make sure you understand the content of chapter 2 before going through this section. The aim of the linear predictive calculation algorithm is to establish a model for the speech final processing from the observation of signal inputs and outputs sequences. Through the calculation, we are able to estimate the set of coefficients to provide an estimate/prediction for a forthcoming output sample $y'[n]$ given the input $(x[n])$ and/or output$(y[n])$ samples:

$$s[n] = -\sum_{k=1}^{p} a_k s[n - k] + G \sum_{m=0}^{q} b_m u(n - 1)$$  \hspace{1cm} (3.1)

where $b_0 = 1$, for coefficients $a_k$, the variable $k \in [1, p]$; for $b_m$ the variable $m \in [1, q]$ and gain $G$ are the parameters of a hypothetical system, from the formula here, the output $s[n]$ is a weighted predictable signal from a linear combination of past inputs and outputs.

The most common form of the linear prediction being used is set the value of coefficients $b_m$ to be zero, so that the calculation of output is directly determined from the previous output.

Let the approximation of the $s[n]$ be $s'[n]$.

$$s[n] = -\sum_{k=1}^{p} a_k s[n - k]$$  \hspace{1cm} (3.2)

The output $s'[n]$ is the $p$th order linear auto-regressive predictor. The relation between predicted output and real output can be calculated by introducing the concept of predictor error $e[n]$, which is the actual residual values
prediction error for each sample given by\cite{10}:

\[ e[n] = s[n] - s'[n] = s[n] + \sum_{k=1}^{p} a_k s[n - k] \]  (3.3)

### 3.2.2 Solution of linear prediction equations

To calculate the predictor coefficients \( a_k \), we first observe the behaviours of the system over \( N \) samples, then set the order \( q \) of the predictor required, for my implementation I set LPC order to be 10, the coefficients is calculated by finding the values which minimise the energy in error signal over \( N \) samples, the following formula is so-called least-squares minimisation leads to a system of \( q \) equations in \( q \) unknowns to be solved to find the best fitting coefficients\cite{9}:

\[
\sum_{k=0}^{q} (a_k \sum_{n=0}^{N-1} y[n - k]y[n - j]) = 0, j = 1, 2, ..., q
\]  (3.4)

The above function is normally adapted for the unstable, considering the FLAC encoding process commonly take flat and continuous music signal as input, thus we have a simpler function which is known as the autocorrelation function and normally deals with the signal which is zero when it is out of the \( N \) sample region:

\[
\sum_{k=0}^{q} (a_k \sum_{n=0}^{N-1} y[n - k + j]) = 0, j = 1, 2, ..., q
\]  (3.5)

From the above equation we can derive out the autocorrelation coefficient \( r_i \):

\[
r_i = \sum_{n=-\infty}^{\infty} y[n]y[n - i]
\]  (3.6)

The above autocorrelation formulation of the least squares fit of the predictor coefficients produces a system of linear equations which can be represented in the form of matrix and using Gaussian elimination to solve it\cite{9}. The pseudo code of Algorithm 1 is my implementation of the linear predictive algorithm with Gauss-Jordan Elimination Methods.

### 3.3 CUDA Implementation

The different structure of CPU and GPU enables them have different task to dealt with, CPU mainly focus on the task with limited number of data but complicated calculation work while GPU focus on the large number of data but limited calculation of work. From the above linear predictive
Algorithm 1 Linear predictive algorithm with Gauss-Jordan Elimination

**Input:** $W$ is the waveform input
Order is the predictor order required
$M$ is an array
$S$ is an array

**Output:** $result$ is the result of the calculated coefficients

1: for $i : 0...Order$ do
2:   for $k : 1 ... W.count()-1$ do
3:     $sum += W[k] \times W[k + i]$  
4:   end for
5:   $M[i] = sum$
6: end for
7: for $i : 0...Order$ do
8:   for $j : 0...Order$ do
9:     $S[i \times (Order + 1) + j] = \text{dotproduct}(M[i])$
10: end for
11: end for
12: for $j : 0...Order$ do
13:    PivotRow = 0
14: for PivotRow : 0 ... Order do
15:   if PivotRow = rows then
16:     rowSwap($M, PivotRow, Order + 1$)
17:   end if
18:    $factor = M[PivotRow \times 10 + j]$
19: for $k : 0...Order$ do
20:    $M[PivotRow \times cols + k]/ = factor$
21: end for
22: $M[PivotRow \times 10 + j] = 1$
23: for $i : PivotRow+1....row$ do
24:    $factor = M[i \times 10 + j]$
25: for $k : 0...cols$ do
26:    $M[i \times cols + k] = M[pivotRow \times cols + k] \times factor$
27: end for
28: $M[j*col+j]=0$
29: end for
30: end for
31: for $i : 0...PivotRow-1$ do
32:   while $M[i \times cols + pivotCol] = 0, pivotCol < cols$ do
33:      $result[pivotCol] = M[i \times cols + cols - 1]$
34:   end while
35: end for
36: return $result$
37: end for
algorithm with Gauss-Jordan Elimination, we’ve calculated the coefficients result of the autocorrelation formulation, the residual computation means multiply this coefficients results with samples before to do shift operation to represent those original raw signals with less bits, this kind of computation is less complicated but need large numbers of data, it is well-adapted to the cuda programming.

The way of CPU and GPU support thread is also different, each core in CPU only has limited numbers of registers, each register will be used when they are operated with the any assigned tasks, to operate different tasks, CPU will have a quickly context switch between different tasks, while from the perspective of time, the cost of such kind of switch is quite expensive because during each time of switch, it need to save the data from the Register to the RAM, when the CPU begin operating this task again, it retrieve from the RAM[11].

Compared with this, GPU also implement with concept of context switch, but it has multiple groups of registers not the single group of register, thus one context switch will only need to set a register group scheduler, which can be regarded as change in and out from the current group of registers, thus the speed of it will be faster times of magnitudes compared to save the data to RAM[11].

To decompose the problem into parallel chunks, the data of each sample are independent from each other, the introduction of parallelism in this encoding process is available on different levels (from fine to coarse):

1. Parallel processing of one single functional units: sufficiently data parallel operations can be implemented by a number of threads, eg, the amplification of a signal by a certain factor with each thread processing a single sample[12].

2. Exploitation of data parallel structures within the encoding process: parallel processing of multiple functional units of the same type without dependencies, in the form of different of different CUDA blocks[12].

3. Depending on the outcome of levels 1 and 2, the amount of channels that can be processes per multiprocessor can be determined: it makes the multiprocessor has enough resources (like required registers per thread, shared memory) to encode multiple channels at the same time[17].

From above points mentioned, my implementation of cuda is at the level of 1 and 2, the process of multiple channels is still sequentially.

From the figure illustrated in 3.2 we can see that firstly i allocate the memory for the calculated coefficients and samples, then i also allocate the same
size of memory of these 2 array on GPU, the third array called 'save' is used to save the output of the samples after the residual computation, the size of 'save' array is double size of the 'channels' array, in each element of the 'save' array, the first half part is used to save the calculated samples which requires fewer bit to save, the second half part is used to record the size of each sample, which is further used to allocated memory for rice coding part. In my artefact, the LPC order of autocorrelation function is 10, thus the number of samples for each thread to process is 10.

From the above implementation description we can have the time complexity for the serial residual computation is $O(\text{Sample Number} \times 10)$, for the parallel residual computation part, we need to consider the time cost of copy data from CPU to GPU part, thus the time complexity for parallel part should be $O(\text{Sample Number}) + O(\text{Sample Number} / \text{Thread numbers} \times 10) + O(10)$

The speed up formula can be described as

$$\text{SpeedUp} = \frac{\text{Serialcode}}{\text{ParallelCode}}$$

(3.7)

Thus from the above illustration, we have the formula of speed up can be described as

$$S = \frac{O(\text{SampleNumber} \times 10)}{O(\text{Memory}) + O(\text{SampleNumber} / \text{ThreadNumber} \times 10)}$$

(3.8)
Chapter 4

Performance Evaluation And Analysis

This chapter summarizes the experiment result of the FLAC encoding process with cuda implementation using different number of samples from the waveform input file, and analyzes the comparison between the performance of CPU and GPU.

4.1 Experimented Environment

The implementation of the Experiment is written in cuda and C, the experiment is tested on the operation system of Ubuntu with version of 16.04.2 which is embedded with the system of LINUX (4.4.0-75-generic X86_64), the tested GPU is GeForce GTX 1080, with the compute capability of 6.1 and clock rate of 1835000, the tested CPU is a 2.5 GHZ Intel Core i7 with the memory of 16 GB 1600 MHZ DDR3.

4.2 Inputs

The input audio file of the evaluation is an one channel, 21.6 MB, 4 minutes 4 seconds long waveform audio format with the sample rate of 44,100, and the size of per sample is 16 bits.
4.3 Performance Analysis

The Figure 4.1 above illustrates the performance comparison of the FLAC encoding process between CPU and GPU, the unit of x-axis is the number of samples from the input audio file, the unit of y axis is the time spent on the encoding process, the unit is evaluated in usec which is microseconds ($10^{-6}$ seconds), form the graph it can be analysed that before the 2000 samples, both CPU and GPU didn’t take much time to finish the encoding process, the time spent on GPU is a little higher than that of CPU, when the number of samples larger than 2000, CPU cost much longer time to finish the encoding process and increase greatly with the increment of the sample numbers, while the GPU part keep a slight increment.

During the beginning of the encoding process, if it is implemented with the cuda programming, the system need to allocate extra memory for the data of samples, coefficients and calculated results so that they can be later transmitted on the GPU memory to the computation, at the same time, the GPU part should also allocate enough memory for keeping this data to be calculated, thus, the time spent on the memory allocation and transmission between CPU and GPU make GPU version take much longer time to finish the process, however with the increment of sample numbers, the code implemented on CPU will take much longer time to finish the encoding job since there is only one thread do deal with the large number of data, however in GPU, the threads assigned to deal with the encoding job is kept the same increasing trend with the audio samples, thus it won’t take much more time for the larger number of data. The Figure 4.2 describes the comparison of
percentage of operating time in CPU and GPU, to have a better illustration, the unit on the X-axis is 10 times the size of the former one, during the beginning of the encoding process, the percentage of time cost on GPU and CPU are nearly same, then within the increment of sample numbers, the percentage of cost on CPU reach nearly 70 percent of total running time of program while the percentage for GPU is still lower than the 20 percent, thus the cuda implementation is tested to be effective enough when it is deal with the large number of data.

The Figure 4.2 describes the comparison of percentage of operating time in CPU and GPU, to have a better illustration, the unit on the X-axis is 10 times the size of the former one, during the beginning of the encoding process, the percentage of time cost on GPU and CPU are nearly same, then within the increment of sample numbers, the percentage of cost on CPU reach nearly 70 percent of total running time of program while the percentage for GPU is still lower than the 20 percent, thus the cuda implementation is tested to be effective enough when it is deal with the large number of data.

4.4 Limitations

In the Figure 4.1, although the performance of GPU is much better than that of CPU, but we observe a circumstance that the increasing trend of CPU running time is not a linear curve, within the enlargement of the data samples, the slope of the curve is also increasing, the reason cause this circumstance may due to the my modification of the traditional FLAC encoding process. As it is mentioned in the chapter 2, the FLAC encoding
process separates the audio sample into blocks to make the encoding process more effective, each block contains 4096 samples, thus samples being sent to LPC will then write in FLAC format, however in my implementation of GPU, the samples are not assigned into blocks and distributed to the LPC to do the prediction process one time, which may cause the curve become non-linear, the solution to this limitation should find alternative to implement block methods on GPU and escape form the non-linear calculation.
Chapter 5

Future Work and Conclusion

This report and project gives a detailed and concise implementation of encoding FLAC files with the GPU acceleration. The decoding methods of the wave form input file is created to get the header information and data from the raw signal, one linear based algorithm is implemented to predict the current sample from the previous number of samples, and the methodology of applying calculated coefficients on residual in cuda programmmign style is illustrated and performed in the project to accelerate the rate of encoding process.

The result of the encoding process in CPU and GPU version is described and compared in the form of line graph, the results demonstrate that with GPU acceleration, the speed of encoding process has been improved significantly.

The future work of the project aims at combining the encoding process with GPU acceleration with the unified memory programming model in CUDA 6. In the model of UMA(Unified Memory Access), it creates a pool of managed memory that is shared between the CPU and GPU, managed memory is accessible to both the CPU and GPU using a single pointer, the system will automatically migrate data allocated in Unified Memory between host and device to make it look like CPU memory to code running on the GPU and vice in versa [13]. Within the implementation of UMA, it may possible to limit the time of transmitting data between CPU and GPU, and finish the writing frame jobs without copying data back to CPU.
Appendices
Appendix A

Independent Study Contract
### Assessed project components:

<table>
<thead>
<tr>
<th>Component</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report: name style:</td>
<td>50%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g. research report, software description...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artefact: name kind:</td>
<td>40%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g. software, user interface, robot...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation:</td>
<td>10%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MEETING DATES (IF KNOWN):

During the summer session every few days, and then during semester 1 2017 weekly.

### STUDENT DECLARATION: I agree to fulfil the above defined contract:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
</table>

### SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project.

<table>
<thead>
<tr>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
</table>

### REQUIRED DEPARTMENT RESOURCES:

+ Most of the development can be done on the student's laptop.

### SECTION C (Course coordinator approval)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
</table>

### SECTION D (Projects coordinator approval)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
</table>

---

Research School of Computer Science  
Form updated Jun-12
PROJECT DESCRIPTION:
The project will explore the FLAC lossless audio format. This format has grown in popularity as a
lossless audio format because the format is “free”, relatively simple, and effective for compressing audio
data. This project would explore encoding of a FLAC audio file. There are many different ways raw
audio data can be encoded using the FLAC format, all of them would decode to an identical audio signals.
However the encoded file would be of different lengths, so the task of the encoder is to search an encoding
which minimizes the overall length. Thus there is a balance between the overall compression and the time
taken to compress this data. For good compression a 60 minute recording may take a few minutes to
compress using a single threaded CPU, however, such performance could be greatly improved by using a
GPU. This project would explore the use of a GPU for encoding FLAC audio data. The performance
and the performance bottlenecks will be evaluated for the proposed approach, in particular the analyses
of memory transfer time along with synchronization costs will be evaluated.

Such performance improvement would be most useful for audio editing or transcoding software.

The project report will contain:
+ An introduction to the topic.
+ A background section which describes the FLAC format,
+ A section which provides a background to GPU computing.
+ A description of the algorithm for encoding
+ A description of the implementation.
+ Experimental chapter which: describes the hardware used for evaluation, the experiments done, and
  the results tabulated/graphed.
+ Conclusion/discussion/limitations/future work chapter.

ASSESSMENT (as per course’s project rules web page, with the differences noted below):

Research School of Computer Science

Form updated Jun-12
INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the project's co-ordinator

SECTION A (Students and Supervisors)

UnitID: u5833990

SURNAME: Wang  FIRST NAMES: Yang

PROJECT SUPERVISOR (may be external): Dr Eric McCraith

COURSE SUPERVISOR (to RUCT Academic):

COURSE CODE, TITLE AND UNITE: COMP4560 _ Advanced Computing Project

SEMESTER  S1  S2  YEAR: Summer session 2016/2017 (6u) and Semester 1 2017 (6u)

PROJECT TITLE:
FLAC encoding using GPU acceleration

LEARNING OBJECTIVES:
The student would gain a good understanding of the binary formats, particularly the FLAC audio format, and GPGPU software development. With a focus on looking at performance relating to the encoding of a FLAC audio file. More generally the project would strengthen the programming and problem solving abilities along with research skill associated with exploring approaches and ideas and then implementing, testing and evaluating these approaches and ideas.

Also it is expected that the student would gain general skills relating to: writing a report, and giving a seminar.

Research School of Computer Science

Form updated Jun-12
Appendix B

Project Description

PROJECT DESCRIPTION: The project will explore the FLAC lossless audio format. This format has grown in popularity as a lossless audio format because the format is: 'free', relatively simple, and effective for compressing audio data. This project would explore encoding of a FLAC audio file. There are many different ways raw audio data can be encoded using the FLAC format, all of them would decode to an identical audio signal. However, the encoded file would be of different lengths, so the task of the encoder is to search an encoding which minimizes the overall length. Thus there is a balance between the overall compression and the time taken to compress this data. For good compression a 60 minute recording may take a few minutes to compress using a single threaded CPU, however, such performance could be greatly improved by using a GPU. This project would explore the use of a GPU for encoding FLAC audio data. The performance and the performance bottlenecks will be evaluated for the proposed approach, in particular the analyses of memory transfer time along with synchronization costs will be evaluated. Such performance improvement would be most useful for audio editing or transcoding software. The project report will contain:

+ An introduction to the topic.
+ A background section which describes the FLAC format.
+ A section which provides a background to GPU computing.
+ A description of the algorithm for encoding.
+ A description of the implementation.
+ Experimental chapter which: describes the hardware used for evaluation, the experiments done, and the results tabulated/graphed.
+ Conclusion/discussion/limitations/future work chapter.
Appendix C

Readme

Plac Encoder with cuda based Acceleration

# source file:
- book.h // includes the nodes and API function for cuda kernel by NVIDIA Corporation
- Header.h // Header file for wave file decoding
- main.cu // The main code for decoding wav file and encoding to FLAC format

Set up and compile

# compile
# To compile this program, it needs the environment of nvcc compiler and Nvidia CUDA environment

The hardware need: one cuda-enabled NVIDIA GPU

Usage: nvcc -o main main.cu

To test the encoding process, you need to change the path of the wave file you want to encode in main function wav_dec("your_path_file")

Run

# function
# wav_dec Finishing reading a WAV file and decode sample data from it.

**par: keep the record of all information about wave files, includes: number of channels, sample rate, sample depth (bits for each sample) , the sample numbers for each channel

**samples : Keep record of all audio samples, each sample is represented as a Integer, the size of samples is (number of channels * samples)

**global encode : The kernel function implemented to calculate the residual with coefficients
Bibliography


