Cascading of the PDLZW Compression Algorithm with Arithmetic Coding

NiraliThakkar
Department of Computer Engineering,
Madhuben&Bhanubhai Patel Women's Institute of Engineering
New V. V. Nagar, India

Malay Bhatt
Department of Computer Engineering, Dharamsinh Desai University
Nadiad, Gujarat, India

ABSTRACT
This paper proposes the cascading of two algorithms that combines the features of both PDLZW and Arithmetic coding and also compares this with deflate which is a cascading of LZ77 and Huffman Coding.In PDLZW algorithm, the dictionary is divided into several dictionaries based on the size of the words. With the hierarchical parallel dictionary set, the search time can be reduced significantly. All this dictionaries are operated independently with each other. The results generated by Arithmetic Coding are close to the optimal value (as predicted by entropy in information theory).

General Terms
Data Compression Algorithms.

Keywords
Arithmetic Coding,Huffman Coding, Lossless Data Compression, Lossy Data Compression, Parallel Dictionary LZW (PDLZW), Word-based Dynamic LZW (WDLZW).

1. INTRODUCTION
Data compression is the process of encoding the data, so that fewer bits will be needed to represent the original data whereby the size of the data is reduced.The primary intension of data compression is to reduce the physical capacity of the data. Data compression technique can be broadly divided into two major families: Lossless data compression and Lossy data compression.

As the name implies, the encoded data while restored will be identical to original data in case of lossless data compression. The various data compression techniques include Run Length Encoding, Huffman Encoding, LZW encoding etc. In case of lossy compression there are many trade-offs whereby some degradation in the quality of the data will occur as part of compression.

Deflate is two-stage lossless data compression algorithm that uses the combination of LZ77 and Huffman coding. This will take advantage of both the algorithms. It is a popular compression method that was originally used in the well-known Zip and Gzip software and has since been adopted by many applications. The following figure shows the block diagram of deflate. At encoder side, the row data are compressed by LZ77 encoder. The output of LZ77 is Literals and length/distance. It is processed by Huffman Encoder which results in compressed bit stream. At decoder side, compressed data is decoded in the Huffman Decoder to construct a stream of symbols required by the LZ77 decoder. The LZ77 decoder operates reconstruct the original data.

The Huffman algorithm is simple and efficient. Huffman codes have to be an integral number of bits long, and this can sometimes be a problem. If the probability of a character is 1/3, for example, the optimum number of bits to code that character is around 1.6 (= $-\log_2(1/3)$) bits. Huffman coding has to assign either one or two bits to the code and either choice lead to a longer compressed message than is theoretically possible. This non optimal coding becomes a noticeable problem when the probability of a character is very high [2]. Thus the Huffman coding always produces rounding errors while Arithmetic coding replaces a stream of input symbols with a single floating-point output number.

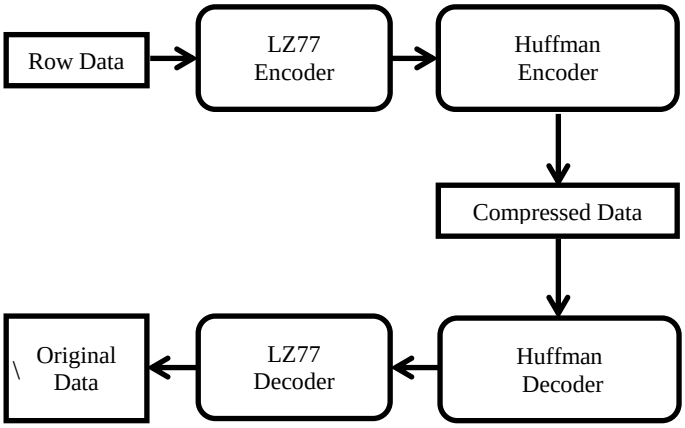

Fig 1 Block diagram of deflate

One of the most widely used compression methods for lossless compression is LZ77.LZ77 encoder maintains a sliding window to the input stream and shifts the input in that window from right to left as strings of symbols are being encoded. The window below is divided into two parts. The part on the left is called the search buffer. This is the current dictionary, and it always includes symbols that have recently been input and encoded. The part on the right is the look-ahead buffer, containing text yet to be encoded. An LZ77 token has three parts: offset, length, and next symbol in the look-ahead buffer. The main disadvantage of LZ77 is the size of both the buffers is very small. Increasing the sizes of the two buffers also means creating longer tokens. These will produce the higher compression ratio but it will reduce the compression efficiency [5].

LZW is a dictionary based compression, which encodes input data through establishing a string table and searching the table to identify the longest possible input data string that exists in the table. The encoded output is a sequence of the matching string's address and length. It can typically compress large English texts to about half of their original sizes. However,

conventional LZW algorithm requires large amount of processing time for adjusting and searching through the dictionary [3].

The dynamic LZW (DLZW) and word-based DLZW (WDLZW) algorithms were proposed to improve searching efficiency. In DLZW, the dictionary has been initialized with different combinations of characters. It is organized in hierarchical string tables. The baseline idea is to store the most frequently used strings in the shorter table, which requires fewer bits to identify the corresponding string. The tables are updated using the move-to-front and weighting system with associated frequency counter. During the compression time, after the longest matching string is recognized in the table, it is moved to the first position of its block. The table updating process is based on the least recently used (LRU) policy to ensure that frequently used strings are kept in the smaller tables. This is to minimize the average number of bits required to code a string when compare with a single table implementation [3].

The WDLZW algorithm is a modified version of DLZW that focuses on text compression by identifying each word in the text and make it a basic unit (symbol). The algorithm encodes the input word into literal codes and copy codes. If the search for a word has failed, it is sent out as a literal code, which is its original ASCII code preceded by other codes for identification. The copy code is the address of the matching string in the string table. However, both algorithms are too complicated. To improve this, parallel dictionary LZW (PDLZW) was proposed. Since not all entries of the DLZW dictionary contains the same word size, this leads to the need of the entire dictionary search for every character. Consequently, the PDLZW has designed to overcome this problem by partitioning the dictionary into several dictionaries of different address spaces and sizes. With the hierarchical parallel dictionary set, the search time can be reduced significantly since these dictionaries can operate independently and thus can carry out their search operation in parallel [3].

2. TWO-STAGE ALGORITHM

In this section, a new two-stage algorithm is proposed. In this approach, the row data is given to the PDLZW encoding algorithm. The output of the PDLZW is given to the Arithmetic coding for further compression. The decompression process is totally reverse. Figure 2 shows the block diagram of this new two-stage algorithm.

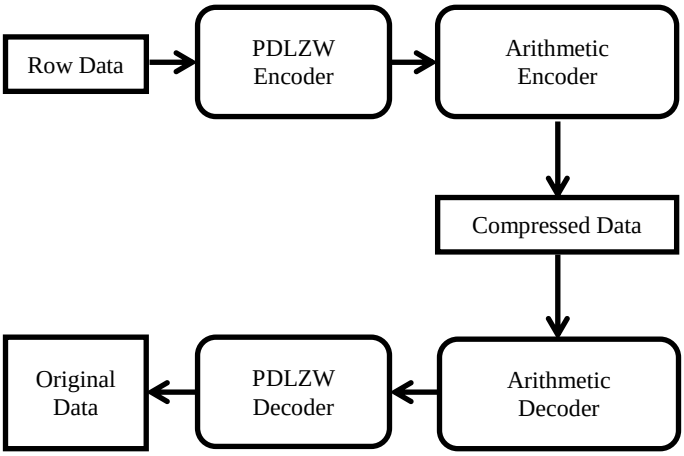

Fig 2 Block diagram of a new two-stage algorithm

As shown in figure 2, the row data is given to PDLZW encoding algorithm.PDLZW algorithm is a LZW based implementation using a parallel dictionary set. It partitions one large dictionary into several small variable-word-width dictionaries. Searching in parallel through small dictionaries would require less amount of processing time than searching sequentially through one large-address-space dictionary.

The PDLZW encoding algorithm is based on a parallel dictionary set that consists m of small variable-word-width dictionaries, numbered from 0 to m-1, each of which increases its word width by one byte. More precisely, dictionary 0 has one byte word width, dictionary 1 two bytes, and so on. The following show the detailed operation of the PDLZW encoding algorithm. PDLZW dictionary initialized with the input symbols. $\sum$ represents the set of input symbols and $|\sum|$ indicate the number of input symbols. The PDLZW Encoding Algorithms are shown in [1] and [4]. The detail example of this algorithm is shown in [6].

As shown in figure 2, the output of the PDLZW is given to the Arithmetic encoder as an input. The output from an arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. To construct the output number, the symbols are assigned set probabilities. The detail arithmetic encoding algorithm is shown in [2].

At the decoder side, first compressed file will be decompressed by Arithmetic Decoder and the dictionary index required by PDLZW is generated. Output of Arithmetic decoder is given to the PDLZW decoder as an input. PDLZW decoder uses this index and generates the actual string. One reason for the efficiency of the PDLZW algorithm is that it does not need to pass the dictionary to the decoder. The PDLZW decoder can be built exactly as it was during compression, using the input stream as data.

3. IMPLEMENTATION PROCESS

The following table shows the content of input file. This row data is given to the PDLZW encoder as an input. PDLZW encoder first separate out all character used in this file and store those character into first dictionary.

TABLE I
CONTENT OF INPUT FILE

Input File:
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <times.h>
#include <stdlib.h>
#include <unistd.h>
#defined max1 500
int END_OF_STREAM, *a1;
static unsigned short int code;
static unsigned short int low;
static unsigned short int high;

The following table shows the dictionary structure which is generated by the PDLZW Encoder. The total entry available in the dictionary is 249. So the table shows the some selected

entries in the dictionary. In table II, '∟' indicate new line character. Here some selected entries are given. Dictionary numbers are based on the size of string.

TABLE II
DICTIONARY STRUCTURE

Dictionary Number	Dictionary Index	String
Dictionary – 0	0	#
	1	i
	2	n
	• • •	
Dictionary – 1	45	#i
	46	in
	• • •	
• • •	• • •	
Dictionary – 9	248	.h>∟#incl
Dictionary – 10	249	.h>∟#inclu

After generating the dictionary, the PDLZW Encoder generates the final output. The final output is shown by table III.

TABLE III
OUTPUT OF PDLZW ENCODER

Index	Output
0	0
1	1
2	2
3	3
4	4
5	5
• • •	• • •
21	49
22	51
23	53
24	55
• • •	• • •
204	10
205	124

Now, this final output of PDLZW Encoder is given to the Arithmetic Encoder. Arithmetic Encoder findsthe frequency of the each digit. Following table shows the frequency count of some selected digits.

After calculating frequency count, Arithmetic Encoder finds the lower and upper range of each value and then calculates the low and high value according to the Arithmetic Encoding Algorithm [2].

TABLE IV
FREQUENCY COUNT OF EACH COUNT

Digits	Frequency Count
0	1
1	8
2	6
3	4
4	4
5	2
6	4
7	4
8	0
• • •	• • •
247	1
248	1

Arithmetic Encoder generates the compressed bit-file. Figure 3 shows the size of original file and compressed bit file. File size of original file is 363 bytes and the file size of compressed file is 173 bytes. Figure 4 shows the content of compressed bit file. The first character of bit-file is NUL. The second character is STX and so on. These are the ASCII character for the specific value. These are the non-printable characters.

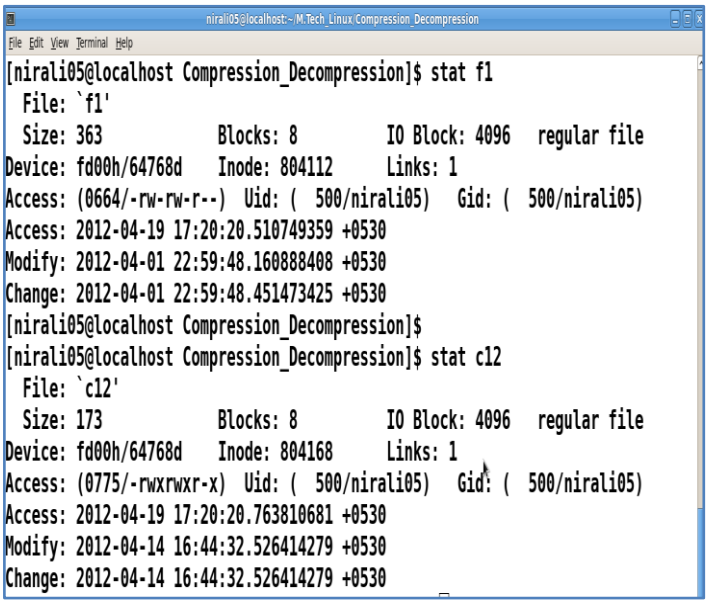

Fig 3File sizes of original file and compressed file

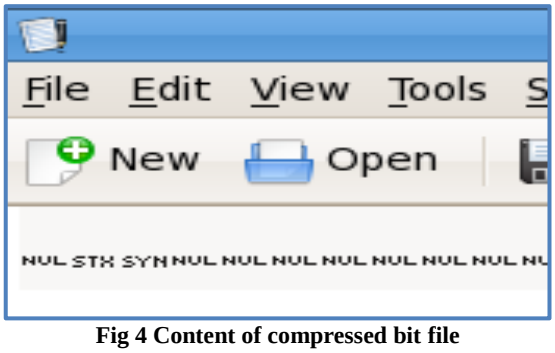

Fig 4 Content of compressed bit file

TABLE I
IMPLIEMANTATION RESULTS

File_name	Original_size (Byte)	Deflate		Proposed Algorithm	
		File Size	Compression Ratio	File Size	Compression Ratio
new.txt	261	142	54.40	97	62.83
file1.txt	12288	960	92.18	948	92.88
PDLZW10.c	68608	2662.4	96.11	2560	96.26
PDLZW_Arithmetic_ Decoding.c	88064	2867	96.74	2867	96.74
main.txt	872448	4505	94.83	4505	94.83

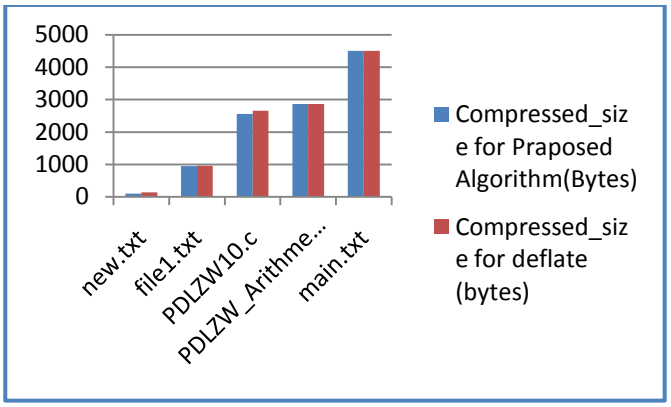

Fig 5 Comparison between Deflate and Preposed algorithm (PDLZW + Arithmetic Codig)

At decoder side, first Arithmetic Decoder one by one reads the content of input file and decodes it. The output of the arithmetic decoder is the same as the input of the arithmetic encoder which is shown in table III. This is given to the PDLZW decoder as an input. PDLZW decoder generates the output string which is identical to the input file which is shown in table I. PDLZW decoder also generates the dictionary which is same as the dictionary generated by PDLZW Encoder (see Table II).

4. COMPRISON WITH DEFLATE

Table I shows the comparison between a new two-stage algorithm with deflate. First column shows the different size of text files. Figure 5 shows the graphical representation of both the algorithms. For the small size of files the compression ratio of both the algorithm is around 50-60 precent. As the file size increases, the compression ratio also increases.

5. CONCLUSION

The cascading of PDLZW algorithm with Arithmetic coding combines the features of both the algorithm and takes advantage of PDLZW algorithm and Arithmetic coding. This combination achieves the higher compression ratio compares to deflate. The PDLZW is better than the other dictionary based algorithms. In this algorithm, the dictionary is divided into several hierarchical partitions. This improves the searching ability of the algorithm. This combination improves the efficiency of algorithm and also gains the higher compression ratio.

6. REFERENCES

[1] M. B. Lin, Jang-Feng Lee and Gene Eu Jan, "A Lossless Data Compression and Decompression Algorithm and Its Hardware Architecture*", IEEE Transections on VLSI Systems*, vol. 14, No. 9, pp. 925-936, Sep. 2006.

[2] M. Nelson and Jean-Loup Gailly , "*The Data Compression Book",* 2nd ed., BPB publications, 1996.

[3] P. Vichitkraivin and O. Chitsobhuk, "An Improvement of PDLZW Implementation with a Modified WSC Updating Technique on FPGA", *World Academy of Science, Engineering and Technology*, 2009.

[4] M.B. Lin, "A Hardware Architecture for the LZW Compression and Decompression Algorithms Based on Parallel Dictionary," *Journal of VLSI Signal Processing 26*, pp. 369-381, 2000.

[5] D. Salomon, "*Data Compression the Complete Reference*", 4th ed., Springer, 2007.

[6] N. S. Thakkar, M. S. Bhatt, "Two-Stage Algorithm for Data Compression", International Conference on Advances on Computer, Electronics and Electrical Engineering, 2012.