High Throughput Lossless Data Compression Algorithm

[1]Deepali V.Patil , [2]S.V. Viraktamath

[1,2]Dept of Electronics and communication Engineering, SDM College of Engineering and Technology
Dharwad-580 002, Karnataka, India
[1]dplpatil27@gmail.com, [2]svvmath@gmail.com

Abstract: Compression algorithms reduce the redundancy in data representation to decrease the storage required for that data. Data compression offers an attractive approach to reducing communication costs by using available bandwidth effectively. Though this scheme has clear benefits, the execution time of compression and decompression is critical to its application in real-time systems. Software compression utilities are often slow, leading to degraded system performance. Hardware-based solutions, on the other hand, often drive large resource requirements and are not amenable to supporting future algorithmic changes. The proposed algorithm suggests a lossless data compression with hash table-dictionary-based scheme

Keywords: *compression; decompression; hash table; dictionary; phrase comparator; data encoder.*

1. INTRODUCTION

The continual increase in information explosion has challenged tremendous improvement in data transmission and storage technologies. Data compression is a method of encoding data with fewer bits by removing the redundancies in the data before either storage or transmission over a communication channel and then reinserted on the associated decompression process. The available bandwidth of a wired or a wireless communication channel can be effectively increased by reducing the amount of data injected in the channel the data redundancy introduced by channel coding for reliability is also offset by data compression techniques [1]. Therefore data compression is a cost effective tool for preserving the expensive resources like communication bandwidth and data storage space and particularly more applicable in wireless communications [2] [3]. Data compression Algorithms/techniques can be divided into two categories - Lossless data compression and Lossy data compression. In Lossless data compression, decompressed. Data is identical to the original uncompressed data. This type of compression scheme is adopted when storing software, text compression, spreadsheets, and word processing files. Lossy data compression is used, when a little loss of data/information is acceptable to user. Normally this technique is used for graphics images and digitized voice.

Data compression algorithms are often implemented in software. Although this approach saves important real estate on processor chips and allows for later modifications to the algorithm, it can sometimes be a performance bottleneck. Most of the existing work on data compression has been concentrated on achieving the best

Compression efficiency possible –aiming for the entropy of the data. However, there are a number of applications where execution speed of the compression / decompression operation is more important than the compression efficiency. Hardware-based fast compression algorithms may be used in such applications. Many such algorithms exist, including custom hardware based ALDC [4], MXT [5], 842[6], and FPGA based XMatchPRO [7]. Most of these solutions, however, utilize expensive CAM (content-addressable memory) structure for implementing the history windows (dictionaries) and achieve throughputs in the range of 100 MB/sec to 400MB/sec.

The remainder of the paper is organized as follows: in section II, we describe the compression algorithm. Section III, we present the results.

2. COMPRESSION ALGORITHM

This section gives an overview of the proposed compression algorithm. We first briefly describe the algorithm and several important features.

Algorithm achieves compression by two means: (1) it uses statically decided, compact encodings for frequently appearing data words and (2) it encodes using a dynamically updated dictionary allowing adaptation to other frequently appearing words. The dictionary supports partial word matching as well as full word matching.

A. Compression

The algorithm identifies repeating patterns of size 8, 4 and 2 bytes in the input data stream and replaces them with 4 bit pointers to previously seen data. Each 8-byte chunk of the input data is divided into 7 phrases (Figure 1) which are compared against previously seen phrases. Dictionaries are used to store the input phrases for lookup in processing subsequent inputs. For constant-time phrase look-up, the address of the phrase in the dictionary (pointer) is stored in a hash table the dictionary supports partial word matching as well as full word matching.

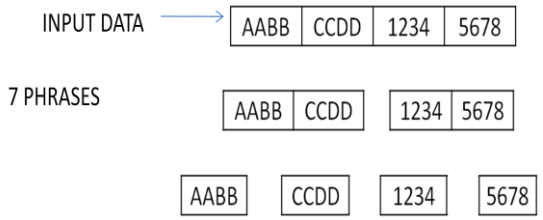


Figure1 8-byte input split into 7 phrases

The patterns and coding schemes used are summarized in Table I, the 'Pattern' column describes frequently appearing patterns, where 'z' represents a zero byte, 'm' represents a byte matched against a dictionary entry, and 'x' represents an unmatched byte. In the 'Output' column, 'b' represents a bit.

We use 64 bit input per cycle. During one iteration each phrase is first compared with patterns. If there is a match, the compressor output is produced by combining the corresponding code and unmatched bytes as indicated in Table I Otherwise; the compressor compares the word with all dictionary entries and finds the one with the most matched bytes. The compression result is then obtained by combining code, dictionary entry index, and unmatched bytes, if any. Words that fail pattern matching are pushed into the dictionary.

TABLE I .ENCODING

Code	Patterns	Output	Bits
0000	ZZZZ_ZZZZ	0000	4
0001	mmmm_mmmm	0001(bbbb)	8
0010	mmmm_xxxx	xxxx 0010(bbbb)	40
0011	xxxx_mmmm	xxxx0011(bbbb)	40
0100	xxxx_xxmm	xxxxxx0100(bbbb)	56
0101	xxxx_mmxx	xxxxxx0101(bbbb)	56
0110	xxmm_xxxx	xxxxxx0110(bbbb)	56
0111	mmxx_xxxx	xxxxxx0111(bbbb)	56
1000	xxxx_xxxx	xxxxxxxx1000	68

Figure 2 shows the different stages of the compression pipeline. The compression pipeline takes 8 bytes of input per cycle and outputs one compressed data. Each 8-byte input is broken into 7 phrases; the hash-table look-up, dictionary look-up and phrase comparison for all these phrases are performed in parallel and a 7-bit match/mismatch status is generated. The data encoder encodes the pointers and the raw input phrases in the smallest possible output, based on the match/mismatch status. Both the compressor and the decompresor are designed as multi-stage pipelines. Input data is streamed through the pipelines, which process one block of input per cycle. The compression operation is purely feed-forward and lends itself well to pipelining.

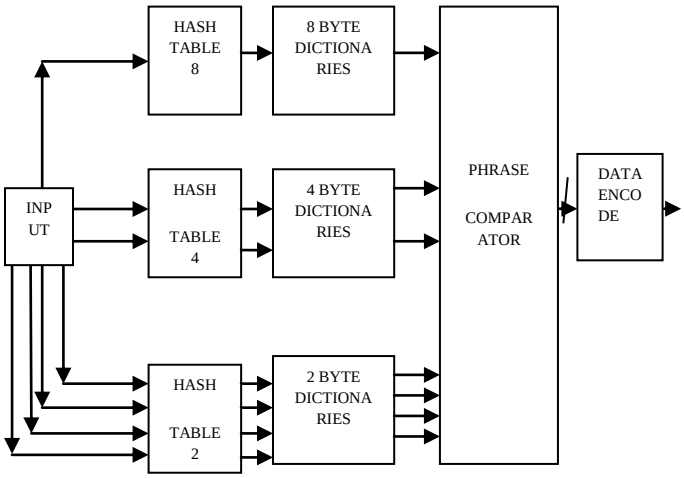


Figure 2 Compression pipeline

B. Decompression

Decompression involves decoding the template and extracting different pointers and raw phrases from the compressed data, reading the remaining phrases from the dictionaries and reconstructing the uncompressed data. Figure 3 shows different blocks of the decompression pipeline. One set of compressed data and its template is fed to the pipeline per cycle. The data decoder decodes the template and extracts the various pointers and raw phrases from the compressed data. The extracted pointers are used to read the phrases from the three dictionaries. That is during decompression the decompresor first reads compressed words and extracts the codes for analyzing the patterns of each word, which are then compared against the codes defined in Table I. If the code indicates a pattern match, the original word is recovered by combining zeroes and unmatched bytes, if any. Otherwise, the decompression output is given by combining bytes from the input word with bytes from dictionary entries, if the code indicates a dictionary match.

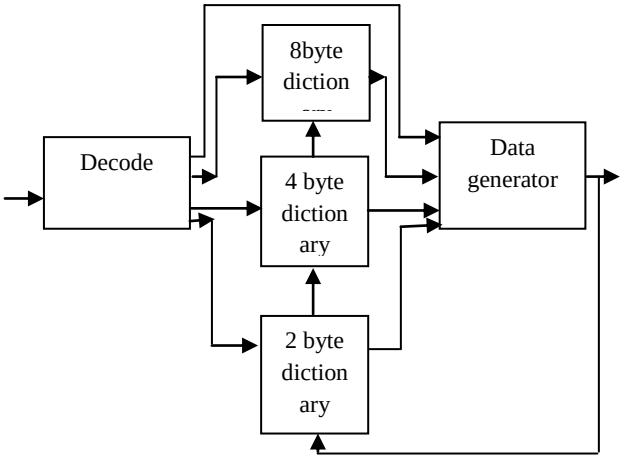


Figure 3 Decompression pipeline

3. RESULTS

A. Timing summary

Timing summary for compressor is as follows

Speed Grade: -1

Minimum period: 8.249ns (Maximum Frequency: 121.224MHz)

Minimum input arrival time before clock: 0.336ns

Maximum output required time after clock: 3.400ns

Maximum combinational path delay: No path found

B Device utilization

Slice Logic Utilization:

Number of Slice Registers: 5704 out of 69120 8%

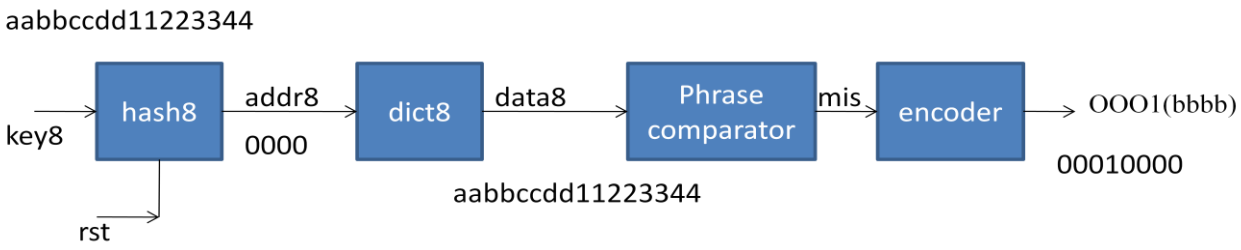
Number of Slice LUTs: 2551 out of 69120 3%

Number used as Logic: 2550 out of 69120 3%

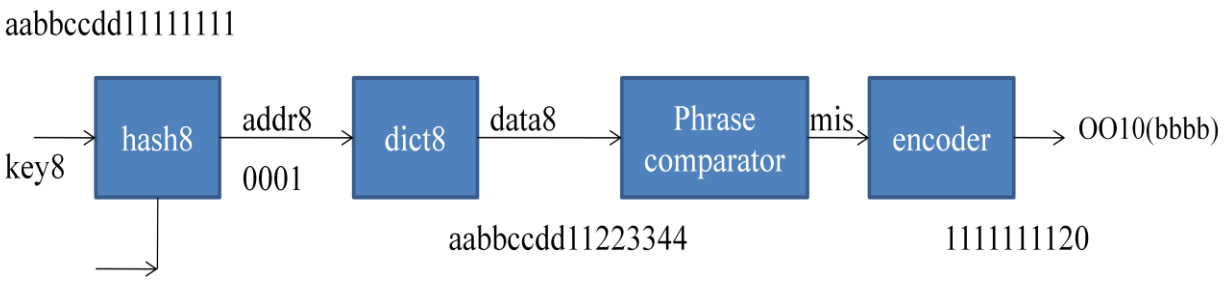
Number used as Memory: 1 out of 17920 0%

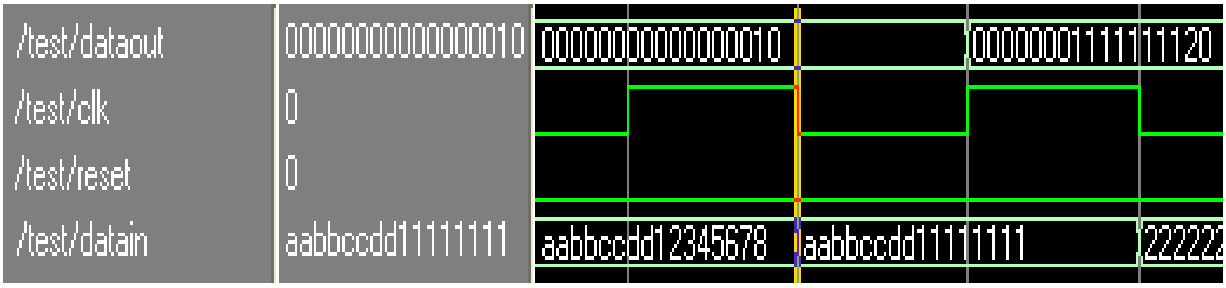
Number used as SRL: 1

C. Output of 8 byte path



D. test input





E. Decompression

Decompressed output for test inputs

4. CONCLUSION

In this work we have presented hardware friendly algorithm that increases the throughput and improves the compression ratio as well as rate. Algorithm is designed specifically for hardware implementation. It takes advantage of simultaneous comparison of an input word with multiple potential patterns and dictionary entries. Algorithms inherently parallel design allows an efficient hardware implementation, in which pattern matching, dictionary matching, and processing multiple words are all done simultaneously.

5. ACKNOWLEDGMENT

The authors wish to thank the authorities of SDMCET Dharwad, for facility extended to carry out the present work.

REFERENCES

[1]. Shih-Arn Hwang and Cheng-Wen Wu, "Unified VLSI Systolic Array for LZ Data Compression", IEEE Transactions on Very Large Scale Integration,Vol. 9, No. 4, August 2001

[2]. Bjong Jung, Wayne P. Burleson, "Efficient VLSI for Lempel_Ziv Compression in Wireless Data Compression Networks", Proceedings of International Symposium on Circuits and Systems, London June 1994.

[3]. Kenneth C. Barr and Krste Asanovi´c 1995, "Energy Aware lossless Data Compression", ACM Transactions on Computer Systems, Vol. 24, No. 3, August 2006, Pages 250–291.

[4]. Craft, D. J. "A fast hardware data compression algorithm and some algorithmic extensions", IBM Journal of Research and Development, 42(6), 733 – 745, November 1998.

[5]. Tremaine, R.B., et. al. "IBM Memory Expansion Technology (MXT)", IBM Journal of Research and Development, 45(2), 271-285, March 2001.

[6]. Franaszek, P. A., Lastras, L. A., Peng, S., and Robinson, J. T., "Data Compression with Restricted Parsings", dcc, 203-212, Data Compression Conference (DCC'06), 2006.

[7]. Nunez, J. L., et. al. "X-MatchPRO: A ProASIC-Based 200 Mbytes/s Full-Duplex Lossless Data Compressor", Lecture Notes in Computer Science, 2147/2001, 613-617, January 2001.

[8]. Christina Zeeh Seminar "The Lempel Ziv Algorithm" January 16, 2004