International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 6 (June 2014), PP.16-21

Quad-Byte Transformation Performs Better Than Arithmetic Coding

Jyotika Doshi, Savita Gandhi
GLS Institute of Computer Technology, Gujarat Technological University, Ahmedabad, INDIA
Department of Computer Science, Gujarat University, Ahmedabad, INDIA
jyotika@glsict.org
drsavitagandhi@gmail.com

Abstract:- Arithmetic coding (AC) is the most widely used encoding/decoding technique used with most of the data compression programs in the later stage. Quad-byte transformation using zero frequency byte symbols (QBT-Z) is a technique used to compress/decompress data. With k-pass QBT-Z, the algorithm transforms half of the possible most-frequent byte pairs in each pass except the last. In the last pass, it transforms all remaining possible quad-bytes. Authors have experimented k-pass QBT-Z and AC on 18 different types of files with varying sizes having total size of 39,796,037 bytes. It is seen that arithmetic coding has taken 17.488 seconds to compress and the compression rate is 16.77%. The compression time is 7.629, 12.1839 and 16.091 seconds; decompression time is 2.147, 2.340 and 2.432 seconds; compression rate is 17.041%, 18.906% and 19.253% with k-pass QBT-Z for 1, 2, and 3 pass respectively. Decoder of QBT-Z is very fast, so significant saving is observed in decompression with QBT-Z as compared to arithmetic decoding. Both these techniques require computing the frequencies before starting compression. It is observed that QBT-Z is more efficient than arithmetic coding giving higher compression with very small execution time.

Keywords:- Data Compression, better compression rate, faster execution, quad-byte data transformation, substitution using zero-frequency byte symbols, arithmetic coding

I. INTRODUCTION

Arithmetic coding (AC) is formulated by Elias [11] and was first implemented by Rissanen[12]. It is the most widely used entropy coding method used in the later stage of most of the data compression techniques like LZ methods, JPEG, MPEG etc. Grammer-based codes [10] and recent generation image and video standards including JPEG2000 [20] and H.264 [21]-[24] utilize arithmetic coding instead of Huffman coding. The main drawback of arithmetic coding is its slow execution. Many researchers [13-19] have tried to improve the speed. Authors of this paper have also suggested faster implementations of arithmetic coding [8,9].

Authors of this paper have introduced block-wise k-pass quad-byte transformation using zero-frequency bytes (QBT-Z) [5], where k is the number of repeated passes as specified by user. In each pass, it computes frequency of quad-bytes, sort them to find most frequent quad-bytes and also find unused (zero-frequency) byte symbols in a data block. In the next stage, transformation takes place. Here, more than one most-frequent quad-byte is encoded with zero-frequency bytes at a time in each pass. In the first k-1 passes, half of the possible frequent quad-bytes are transformed in each pass. In the last pass, all remaining possible quad-bytes get transformed at a time.

Performance of faster arithmetic coding suggested by author [8] is compared with various numbers of passes of QBT-Z. It is found that it is more efficient than arithmetic coding. Especially, QBT-Z decoder is too small and executes very fast.

With 3-pass QBT-Z, when compressing different types of files from different domains with a total size of nearly 40MB data, it is found to compress 2.5% more in nearly same time as compared to arithmetic coding. Decompression time of QBT-Z is extremly low, just about 2.4 seconds as compared to 27 seconds of arithmetic decoding time.

Thus, QBT-Z with fewer passes can be considered in place of arithmetic coding in programs like those based on LZ algorithms, JPEG, MPEG where entropy encoder is used in later stage.

II. LITERATURE REVIEW

Arithmetic coding is an entropy encoding technique where compression rate cannot be improved without changing the data model due to its entropy limitation. So improvement is suggested by researchers in its execution time [8, 9, 13-19]. Its decoder is quite time consuming.

Researchers have worked on encoding pair of adjacent bytes. Byte-Pair Encoding (BPE) [1], digram encoding [3, 4] and Iterative Semi-Static Digram Coding (ISSDC) [2] are such algorithms. These algorithms are intended for text files. However, they can be applied to any type of source when the alphabet size is small. These are all dictionary based algorithms.

Authors of this paper have proposed QBT-I [6] and BPT-I [7] dictionary based transformation techniques. They are intended to introduce redundancy in the data for second stage conventional data compression techniques. This algorithm forms logical groups of most frequent quad-byte or byte-pair data in dictionary and encodes data using variable-length codeword made up of group number and index position of data within group. Thus a quad-byte or byte-pair is encoded with less than 16-bits.

Byte pair encoding (BPE) algorithm proposed by P. Gage [1] compresses data by finding the most frequent pair of adjacent bytes in the data and replacing all instances of this pair with a byte that was not in the original data. The algorithm repeats this process until no further compression is possible, either because there are no more frequently occurring pairs or there are no more unused bytes to represent pairs.

The variation of BPE is where algorithm is applied on data blocks in stead of entire file. In this paper, it is referred to as m-pass QBT-Z. It increases the chances of getting unused bytes in a block to achieve compression in any type of source. But, due to multiple iterations, BPE takes very long time to encode the data.

III. RESEARCH SCOPE

Digram encoding, ISSDC and BPT-I are all dictionary based techniques using index in encoding byte pair. Digram encoding and ISSDC are better only when applied to small size alpahbet source. QBT-I and BPT-I are intended for introducing data redundancy for second stage data compression like arithmetic coding or huffman coding.

The problem with BPE is very large execution time due to many repeated passes. An improvement in execution time is seen in k-pass BPT-Z.

Further improvement is possible with encoding quad-byte data instead of byte-pair data using zero-frequency byte in a data block. In k-pass QBT-Z (Quad-Byte Transformation using Zero-frequency bytes), block-wise quad-byte transformation is performed in fewer passes where number of passes k may be specified by user to experiment. This reduced number of passes and transforming 4 bytes at a time help to reduce execution time.

Arithmetic coding is found to execute very slow, especially its decoder. It is not possible to get more compression due to its entropy limit. Arithmetic coding is used very widely in most of the compression techniques in the later stage.

Research scope is in considering k-pass QBT-Z in place of most widely used entropy based arithmetic coding technique. QBT-Z decoder is very fast. If k-pass QBT-Z gives better compression and takes less encoding time, it can be considered to replace arithmetic coding.

IV. BRIEF INTRODUCTION TO K-PASS QBT-Z

QBT-Z is applied on data blocks and quad-bytes are substituted with unused bytes. The process of substitution may be repeated multiple number of times, referred to as pass. Each pass of QBT-Z involves two stages:

1. Determine unused bytes and frequent quad-bytes
2. Transform frequent quad-bytes by substitution with available zero-frequency bytes

Here, it needs to store substitution information for decoder. This information contains the number of quad-bytes transformed and the substitution pairs of quad-byte and unused byte used to encode. At each pass, this is the additional cost of data to be stored as header information with transformed data. So only quad-bytes with specific minimum frequency (say 4) are worth to be considered while transforming.

Thus, at each pass, the algorithm determines how many quad-bytes to transform based on minimum of avialable zero-frequency bytes and number of more frequent quad-bytes.

When k=1, all possible quad-bytes are transformed in single pass only. So, its header contains number of quad-bytes transformed, say n; n substitution pairs and size of transformed data block. Then the transformed data block is written.

When k=2, it transforms only half of the possible quad-bytes in the first stage. It stores the headaer information and transformed buffer in memory itself. This data block is considered for encoding in the second pass. While writing the resulting buffer in file, it needs additionally to store the number of passes also.

Thus, for k>1, it transforms only half of the possible quad-bytes in first k-1 passes. In the last pass, it transforms all possible quad-bytes. While writing the output, it writes number of passes also for decoder to know.

With m-pass QBT-Z, only one most frequent quad-byte is transformed in each pass. This process is repeated maximum possible number of times, say m, until there are no more frequent quad-bytes or no more unused (zero-frequency) bytes in the data block. Here, the header information contains only one substitution pair details in each pass.

V. EXPERIMENTAL RESULTS AND ANALYSIS

Programs for Arithmetic coding (AC) with multi-bit processing [8] using shift, m-pass QBT-Z and k-pass QBT-Z [5] are written in C language and compiled using Visual C++ 2008 compiler. QBT-Z is implemented with the data block is of size 8KB and the data structure used to store quad-byte and its frequency is binary search tree.

Programs are executed on a personal computer with Intel(R) Core(TM)2 Duo T6600 2.20 GHz processor and 4GB RAM.

Experimental results are recorded using average of five runs on each test files. Most of the test files are selected from Calgary corpus, Canterbury corpus, ACT web site. Test files are selected to include all different file types and various file sizes as shown in Table 1.

Table 1 gives the overall compression rate (% of saving in compressed or transformed file size), BPS (Average number of Bits used Per 8-bit Symbol), total execution time of compression and decompression of all test files.

Table I. Experimental Results of AC, m-pass QBT-Z and k-pass QBT-Z

No.	Source File Name	Source size (Bytes)	Compressed File Size (Bytes)					
			AC	m-pass QBT-Z	1-pass QBT-Z	2-pass ABT-Z	3-pass QBT-Z	4-pass QBT-Z
1	act2may2.xls	1348036	789951	730560	824736	765411	761401	760442
2	calbook2.txt	610856	367017	397802	474945	490188	490108	490183
3	cal-obj2	246814	194255	163030	198080	189119	186066	185127
4	cal-pic	513216	108508	81331	184599	112377	99506	97813
5	cycle.doc	1483264	891974	574132	812832	665628	633909	628959
6	every.wav	6994092	6716811	6996644	6996644	6997508	6998362	6999216
7	family1.jpg	198372	197239	198431	198446	198472	198497	198522
8	frymire.tif	3706306	2200585	1212183	1710533	1450974	1421200	1416383
9	kennedy.xls	1029744	478038	335045	389700	350533	346246	346290
10	lena3.tif	786568	762416	786243	786798	786858	786955	787052
11	linux.pdf	8091180	7200113	5973207	6617753	6397862	6358141	6354790
12	linuxfil.ppt	246272	175407	179391	197036	186334	184066	183416
13	monarch.tif	1179784	1105900	1130232	1153911	1166198	1166354	1166499
14	pine.bin	1566200	1265047	1110988	1271663	1242818	1232091	1226568
15	profile.pdf	2498785	2490848	2493687	2495262	2494758	2495009	2495315
16	sadvchar.pps	1797632	1771055	1729980	1751322	1744673	1743477	1743391
17	shriji.jpg	4493896	4481092	4479601	4488212	4487614	4487836	4488131
18	world95.txt	3005020	1925940	2170758	2461967	2544857	2544829	2545171
Total Size (Bytes)		**39796037**	**33122196**	**30743245**	**33014439**	**32272182**	**32134053**	**3211326 8**
Overall Compression Rate (%)			**16.770**	**22.748**	**17.041**	**18.906**	**19.253**	**19.305**
Overall BPS (Bits Per Symbol)			**6.658**	**6.180**	**6.637**	**6.4887**	**6.460**	**6.456**
Total Compression Time (Sec)			**17.488**	**268.434**	**7.629**	**12.183**	**16.091**	**20.994**
Total Decompression Time (Sec)			**27.158**	**4.697**	**2.147**	**2.340**	**2.432**	**2.543**

Compresstion rate and Bits Per Symbol (BPS) given in Table 1 shows that m-pass QBT-Z gives the best compression as compared to AC and k-pass QBT-Z, but the compression time is significantly very high.

Compression rate of AC is 16.77% as compared to 17.041%, 18.906%, 19.253% and 19.305% rate achieved using 1-pass, 2-pass, 3-pass and 4-pass QBT-Z respectively. Here, m-pass (maximum possible passes) QBT-Z performs the best with 22.748% saving. Refer Figure1. As k increases, compression rate increases.

Fig. 1: Compression Rate of AC, m-pass QBT-Z and k-pass QBT-Z

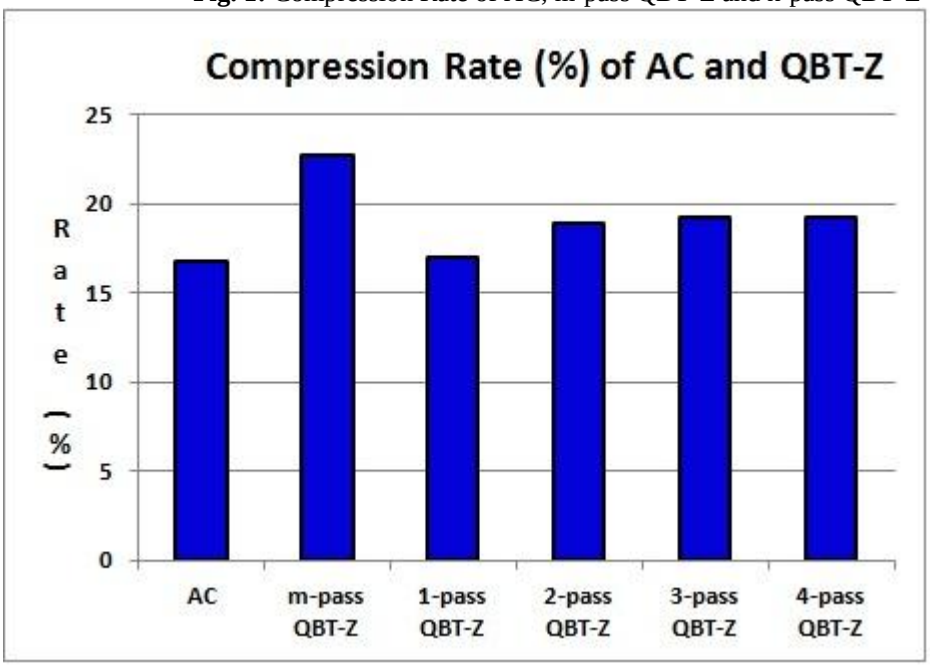
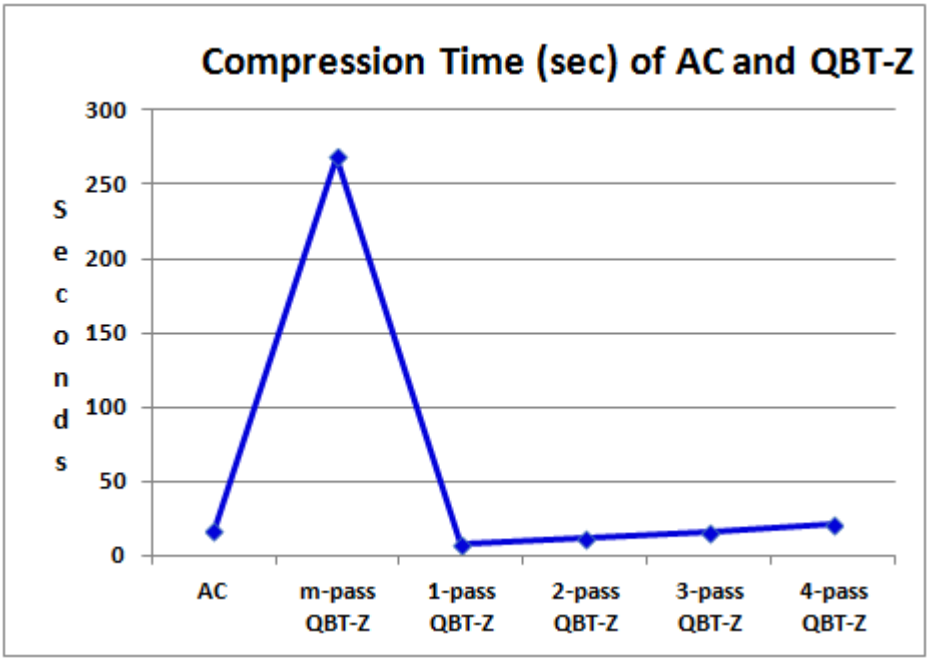


Figure 2 represents compression time. With k-pass QBT-Z, as k increases, it takes larger time to compress but the compression is improved. The cost involved in compression time using m-pass QBT-Z is significantly very high. It is 268.434 seconds as compared to nearly 17 seconds for AC and 3-pass QBT-Z.

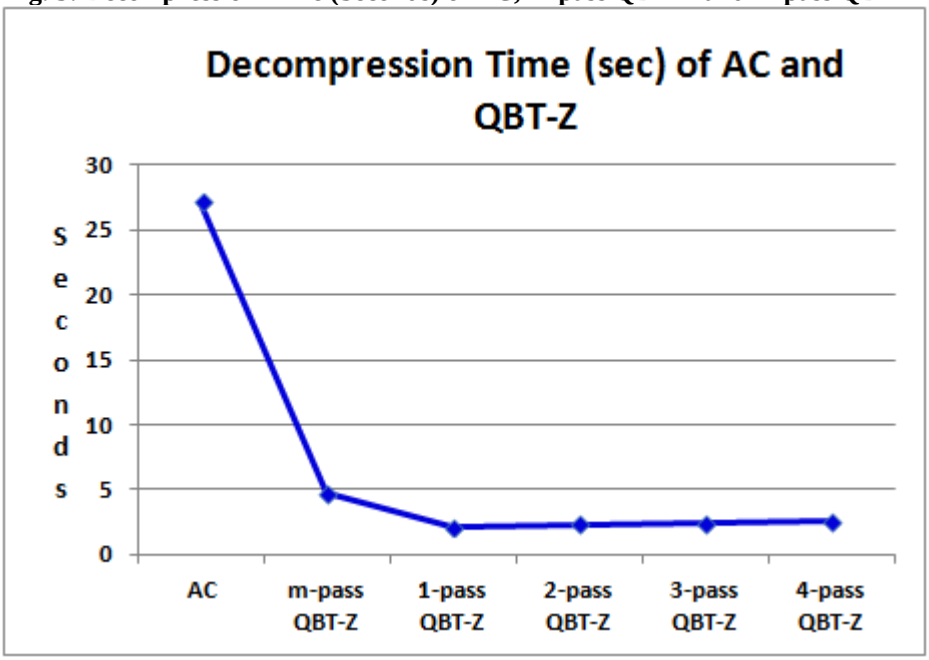
Fig. 2: Compression Time (Seconds) of AC, m-pass QBT-Z and k-pass QBT-Z



Decompression using QBT-Z reverse transformation is extremely fast as expected. It decodes in nearly 2 to 2.4 seconds and there is no considerable difference observed in decoding time of k-pass QBT-Z for k varying from 1 to 4. With m-pass QBT-Z also, 4.7 seconds is not that high as compared to 27 seconds decompression time of arithmetic coding. Refer Figure 3.

When comparing arithmetic coding with k-pass QBT-Z, it is found that 3-pass QBT-Z is better giving 2.5% more reduction in file size taking less time. After 3-pass, improvement in compression is not that significant, but increase in encoding time if seen to be linear with k.

Fig. 3: Decompression Time (Seconds) of AC, m-pass QBT-Z and k-pass QBT-Z



VI. CONCLUSION

QBT-Z with 3 numbers of passes performed better than arithmetic coding. It compresses more in less time. With increased number of passes, compression rate is improved. After 3 passes, improvement rate is not that significant. Compression time is observed to be linear with k. So, increasing number of passes will increase compression time at constant rate. QBT-Z decoder is considerably very fast even with m-pass QBT-Z. It seems to be beneficial to use 3-pass QBT-Z in place of arithmetic coding.

REFERENCES

[1]. Philip Gage, "A New Algorithm For Data Compression", The C Users Journal, vol. 12(2)2, pp. 23–38, February 1994

[2]. Altan Mesut, Aydin Carus, "ISSDC: Digram Coding Based Lossless Dtaa Compression Algorithm", Computing and Informatics, Vol. 29, pp.741–754, 2010

[3]. Sayood Khalid, "Introduction to Data Compression",2nd edition, Morgan Kaufmann, 2000

[4]. Ian H. Witten, Alistair Moffat, Timothy C. Bell, "Managing Gigabytes-Compressing and Indexing Documents and Images", 2nd edition, Morgan Kaufmann Publishers, 1999

[5]. Jyotika Doshi, Savita Gandhi, "Quad-Byte Transformation using Zero-Frequency Bytes", International Journal of Emerging Technology and Advanced Engineering, Vol. 4 Issue 6, June 2014

[6]. Jyotika Doshi, Savita Gandhi, "Quad-Byte Transformation as a Pre-processing to Arithmetic Coding", International Journal of Engineering Research & Technology (IJERT), Vol.2 Issue 12, December 2013

[7]. Jyotika Doshi, Savita Gandhi, "Article: Achieving Better Compression Applying Index-based Byte-Pair Transformation before Arithmetic Coding", International Journal of Computer Applications 90(13):42-47, March 2014.

[8]. Jyotika Doshi and Savita Gandhi, "Computing Number of Bits to be processed using Shift and Log in Arithmetic Coding", International Journal of Computer Applications 62(15):14-20, January 2013

[9]. Doshi, J.; Gandhi, S., "Enhanced arithmetic coding using total frequency in power of 2 & processing multi-bits at a time," Sixth International Conference on Contemporary Computing (IC3), August 2013
[10]. J. C. Kieffer and E. H. Yang, "Grammar-based codes: A new class of universal lossless source codes", IEEE Trans. Inform. Theory, vol. 46, pp. 737–754, 2000
[11]. F. Jelinek, "Probabilistic Information Theory", Mc-Graw Hill, New York, 1968, 00. 476-489.
[12]. J. Rissanen, "Generalized kraft inequality and arithmetic coding", IBM J. Res. Develop., vol. 20, pp. 198–203, May 1976.
[13]. G. G. Langdon, Jr., and J. Rissanen, "Compression of black-white images with arithmetic coding", IEEE Trans. Commun., vol. COMM-29, pp. 858–867, 1981.
[14]. C. B. Jones, "An efficient coding system for long source sequences", IEEE Trans. Inform. Theory, vol. IT–27, pp. 280–291, 1981.
[15]. I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression" Commun. ACM, vol. 30, pp. 520–540, 1987.
[16]. P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression", Proc. IEEE, vol. 82, pp. 857–865, 1994.
[17]. A. Moffat, R. Neal, and I. Witten, "Arithmetic coding revisited," ACM Trans. Inform. Syst., vol. 16, no. 3, pp. 256–294, July 1998
[18]. Boris Ryabko and Jorma Rissanen, "Fast Adaptive Arithmetic Code for Large Alphabet Sources With Asymmetrical Distributions", IEEE Communications Letters, 7(1), January 2003 pp. 33-35
[19]. J. Jhang, X. Ni, "An improved bit-level arithmetic coding algorithm", Journal of Information and Computer Sciencs, 5(3), 2010, pp 193-198
[20]. D. S. Taubman and M. W. Marcellin, JPEG2000: "Image Compression Fundamentals, Standards and Practice", Norwell, MA: Kluwer Academic, 2002.
[21]. T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst.Video Technol., vol. 13, no. 7, pp. 560–576, Jul. 2003.
[22]. Detlev Marpe, Heiko Schwarz, and Thomas Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard", IEEE Trans. On Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 620-636, July 2003
[23]. M. Dyer,D. Taubman, and S. Nooshabadi, "Improved throughput arithmetic coder for JPEG2000", Proc. Int. Conf. Image Process., Singapore, Oct. 2004, pp. 2817–2820.
[24]. R. R. Osorio and J. D. Bruguera, "A new architecture for fast arithmetic coding in H.264 advanced video coder", Proc. 8th Euromicro Conf. Digital System Design, Porto, Portugal, Aug. 2005, pp. 298–305.