Efficient Data Transfer Scheme Using Word-Pair-Encoding-Based Compression for Large-Scale Text-Data Processing

Hasitha Muthumala Waidyasooriya, Daisuke Ono, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi,980-8579, Japan
Email: {hasitha@, ono1831@, hariyama@, kameyama@}ecei.tohoku.ac.jp

***Abstract*—large-scale data processing is very common in many fields such as data-mining, genome mapping, etc. To accelerate such processing, Graphic Accelerator Units (GPU) and FPGAs (Feild-Programmable Gate-Array) are used. However, the large data transfer time between the accelerator and the host computer is a huge performance bottleneck. In this paper, we use a word-pair-encoding method to compress the data down to 25% of its original size. The encoded data can be decoded from any position without decoding the whole data file. For some algorithms, the encoded data can be processed without decoding. Using Burrows-Wheeler algorithm based text search, we show that the data amount and transfer time can be reduced by over 70%.**

Keywords: Succinct data structures, big data, data compression

I. Introduction

Recently, large-scale data processing is very common in many fields such as data-mining [1], genome mapping [2], etc. Graphic Accelerator Units (GPU) and Feild-Programmable Gate-Arrays (FPGA) are used to accelerate such processing in works such as [3] and [4]. These accelerator boards contain a large off-chip memory (global memory). As shown in Fig.1, data are transferred from the host computer to the global memory of the accelerator board. Then those data are accessed from the global memory and processed in the accelerator chip. The processing power of the accelerators are usually very large. However, the time required to transfer data between the host processor and the accelerator, and the time required to transfer the data between the global memory and the accelerator are very large. Moreover, data storage limitations and data access bandwidth limitations are also exists.

Data compression is an effective way of dealing with these problems. However, some compression methods could bring other set of problems. One such problem is not allowing random access. The whole data file should be decompressed to its original state in order to access the data from an arbitrary position. Data decompression requires processing time and also storage space. To solve this problem, Succinct data structures are introduced in works such as [5], so that the complete decompression is not required. In this paper, we consider a relatively old text compression algorithm based on byte-pair encoding (BPE) [6] which allows access from any position without decompressing the whole text.

In this paper, we propose an efficient data transfer scheme for large-scale text processing in hardware accelerators. This data transfer scheme considers; (a) the data transfers between the host computer and the global memory of the hardware accelerator, (b) the data transfer between the accelerator chip and the off-chip global memory. The reduction of data transfer time between the host computer and the global memory is achieved simply by data compression. The data transfer time reduction between the accelerator chip and the off-chip global memory is achieved by accessing the compressed data and processing them without decompressing. We shows an example of text searching, where the data size and data transfer time are reduced by 70%. We also shows that the compressed data can be accessed from any arbitrary position and can be used without decompressing. Since the compressed data are smaller in size compared to that of the original data, the memory bandwidth also can be reduced.

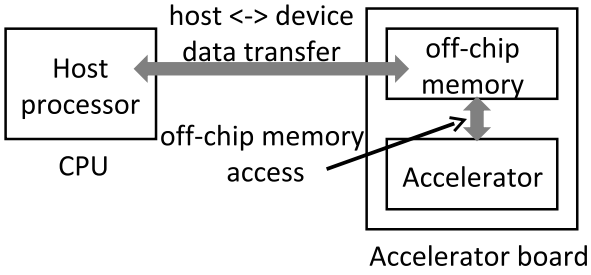


Fig. 1. Data transfers

II. Word-pair encoding based text data compression

A. Word-pair encoding

BPE [6] is a simple data compression method that the most common pair of consecutive bytes of data is replaced with a byte that is not been used already in the compressed data file. Figure 2 shows an example of BPE. A symbol is stored as a 8-bit word or a byte. At the encoding stage 1 in Fig.2, the most common byte-pair, "AB", is replaced by a new byte "E". Then the same process continues in stage 2 by replacing the most common byte-pair of stage 1, which is "EC". The result at stage 2 has a small number of bytes compared to the input data. The result can be decompressed from any arbitrary position. The decoding is done by the table look-up. In this paper, we consider the data compression of words with arbitrary number of bits. Therefore, we call it word-pair encoding.

978-1-4799-5230-4/14/$31.00 ©2014 IEEE

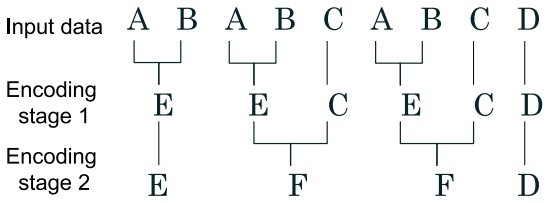


Fig. 2. Byte-Pair encoding

B. Data compression of text search using Burrows-Wheeler algorithm

This section shows an example of text search using Burrows-Wheeler algorithm, which is widely used in genome mapping [7]. Note that, the genome matching is basically a text search, where the number of symbols are limited to four. In this algorithm, Burrows-Wheeler (BW) transform is applied to a text string. The text search uses an FM-index [8] based method that computes the rank and select. Since the intention of this paper is to show how to use the word-pair compressed data, we omit the explanation of BW algorithm. Interested readers can refer the work in [2] and [7] to get a detailed knowledge of Burrows-wheeler algorithm based text search.

Let us consider the BWT based text search using Fig.3. We have a reference string X shown in Fig.3(a). We want to find if a phrase W exists in X. To search in string X, we need inputs $C(.)$ shown in Fig.3(b), and BW array of X shown in Fig.3(c). The BW array is constructed by applying BW transform [7] to the reference genome X. The number of symbols that are lexicographically smaller than a is given by $C(a)$ where $a \in \{A, B, C, ...\}$. The number of occurrences of the symbol a in string $B[0, i]$ is given by $O(a, i)$. The string B is the BW array of the string X. BW algorithm uses the "exact matching" method explained in [8]. According to [8], if a string W is a substring of the string X and $L(aW) \leq U(aW)$, string aW is also a substring of X where aW equals the string $\{a, W\}$. The terms L and U, given by Eqs.(1) and (2) respectively, are the lower and upper bounds of the suffix array (SA) interval of X.

$$L(aW) = C(a) + O(a, L(aW) - 1) \quad (1)$$

$$U(aW) = C(a) + O(a, U(aW)) - 1 \quad (2)$$

Note that, the suffix array shown in Fig.3(c) shows the corresponding positions of the elements after the BW transform to the reference string.

Searching of the phrase W in X is shown in Fig.4. The phrase W is shown in Fig.4(a). The search starts from the last symbol of W which is "A". Since, $L(A) \leq U(A)$, a match is found. Then we move to the next symbol which is at the left side of the searched symbol. According to Fig.4(a), it is "G". Since $L(GA) \leq U(GA)$, we move to the next symbol which is "H". As shown in Fig.4(b), $L(HGA) \leq U(HGA)$. At this time, all the symbols in W are searched and the SA interval is [6,6]. Referring the SA in Fig.3(c), we find that the phrase W is in the position 2 in text string X.

Figure 5 shows the comparison of the architectures that use non-encoded and encoded data. In BW algorithm based search method, computing Eqs.(1) and (2) is very important. For this,

Position	0	1	2	3	4	5	6
Ref. string (X)	E	E	H	G	A	G	$

(a) Reference string X

a	A	E	G	H
$C(a)$	1	2	4	6

(b) Number of characters

Suffix array (SA)		BWT array
	Position in X	
0	6	G
1	4	G
2	0	$
3	1	E
4	5	A
5	3	H
6	2	E

(c) BW array of X

Fig. 3. Text searching using BWT

we need to access the BW array from the global memory and count the number of occurrences of the symbol a in string $B[0, i]$. Figure 5(a) shows the conventional counting method. As shown in Fig.5(a), all the symbols up to the required position (i^{th} position) should be countered. Figure 5(b) shows the proposed counting method using encoded data. When encoding data, we keep a table that shows which symbols in the original data are encode to a new symbol. In this table, we also store the number of original symbols that are represented by the new symbol. When we encounter an encoded symbol, we refer the table and find the number of occurrences of the original symbols. Therefore, we do not have to decode the data to its original form. We can just add the number of occurrences in the table. As a result,decoding time is zero and counting time is reduced. Moreover, since we access smaller number of symbols, the data transfer time between the global memory and the accelerator is also reduced.

III. Evaluation

Table I shows the compression ratio of the data encoding of the English text. The original text data use one byte for a single character. The compressed data size is the percentage of data compared to the original data size. The data size decreases when the number of bits used to represent an encoded word increases. Although larger word sizes increase the encoding time, they do not have any major effect on the decoding time. The last column of Table I shows the data size of the BW transformed text. It also gives an impressive data size reduction, so that we can use it with text searches effectively.

Position	0	1	2
Phrase (W)	H	G	A

(a) Phrase W

L({}) = 0, U({}) = 6
L(A) = C(A) + O(A, 0 -1) = 1+0 = 1
U(A) = C(A) + O(A, 6) - 1 = 1 + 1 - 1 = 1

L(GA) = C(G) + O(G, 1-1) = 4 + 1 = 5
U(GA) = C(G) + O(G, 1) - 1 = 4 + 2 -1 = 5

L(HGA) = C(H) + O(H, 5-1) = 6 + 0 = 6
U(HGA) = C(H) + O(H, 5) - 1 = 6 + 1 - 1 = 6

SA[6,6] = string position 2

(b) Text search

Fig. 4. Text searching using BWT

TABLE I. ENCODING OF THE ENGLISH TEXT (THE BIBLE)

Word length (bits)		Compressed data size %	BW transform compressed data size %
Original	Encoded		
8	8	48.0	34.9
8	9	42.9	32.4
8	10	38.1	31.1
8	11	34.1	30.3
8	12	30.5	29.8
8	13	27.5	29.3
8	14	25.1	

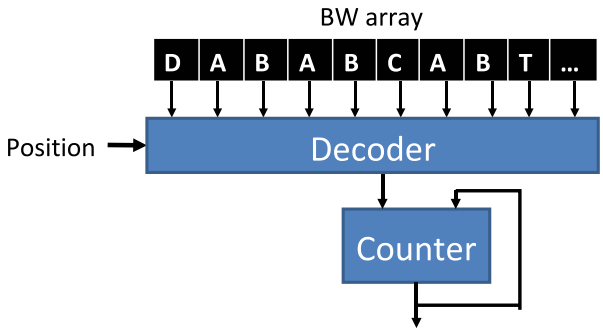
Table II shows the compression ratio of the data encoding of the Japanese text. The original text data use two byte for a single character. This also reduces the data size to 28.8%.

Table III shows the comparison with other compression algorithms using English text (The Bible). In fact, the proposed method is better than "zip, gzip", etc. Only "bzip2" has a better reduction ratio. Since these conventional compression techniques require whole file to be decompressed to find a particular position, the decompressing time and the memory required to hold the decompressed data are very large. Since the proposed method can decompress from any arbitrary position, fast decompression is possible.

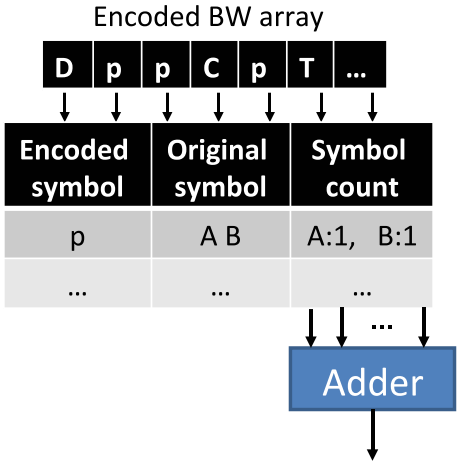
We estimated the amount of memory required for the BW-based text search. We consider the text search of the Bible example. The BW array of the text string is stored in the

TABLE II. ENCODING OF THE JAPANESE TEXT

Word length (bits)		Compressed data size %
Original	Encoded	
16	16	37.2
16	17	32.4
16	18	26.2
16	19	26.4
16	20	28.8



(a) Architecture of character occurance count



(b) Architecture of character occurance count after encoding

Fig. 5. Text searching using BWT

TABLE III. COMPARISON WITH OTHER COMPRESSION ALGORITHMS

Compression algorithm	Compressed data size %
zip	29.06
gzip	29.06
bzip2	20.44
tar + gzip	29.06
Proposed	25.10

memory as a 8192 bit words. Each word contains a header and a text string as shown in Fig.6. The header shows the number occurrences of each symbol in the previous words. The text string is a segment of the BW array. Using this method, we can count the number of symbols in the BW array segment and add to the header to count the total number of symbols up to that position. Therefore, we don't have to count for the symbols in all the words. This method is used for genome sequence mapping in [9]. The results are shown in Table IV. After encoding the BW array data, we achieved a 70.19% memory size reduction. Since the data size has a direct relationship with the data transfer time, we can say that the data transfer time from the host computer to the accelerator board also reduces by over 70%. Since more symbols are included in a single word after encoding, the accelerator accesses more symbols in a single memory read. This will reduces the data transfer time between the accelerator and its global memory. Moreover, the data can be read from any position of the memory and process

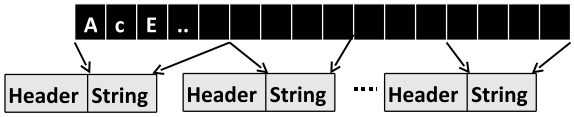


Fig. 6. Storing the BW array

TABLE IV. MEMORY REQUIRED FOR THE BW-BASED TEXT SEARCH

Before encoding (bits)		After encoding		Memory
Header	Data	Header	Data	reduction %
512	7,680	512	1930	70.19

without decompressing, for some algorithms.

IV. CONCLUSION

This paper evaluate a data encoding scheme based on BPE that can be used to compress the data and to reduce the data transfer time. For some algorithms such as BWT-based text searching, the encoded data can be used without decompressing. According to the experimental results, we achieved a compression ratio of 25% for English text and 29% for Japanese text. For the BWT-based text search example, we can achieve over 70% reduction of memory amount and data transfer time. Therefore, the proposed encoding method can be used to reduce the data transfer time between host computer to accelerator memory, and also from the accelerator memory to the accelerator logic.

ACKNOWLEDGMENT

This work is supported by MEXT KAKENHI Grant Number 24300013.

REFERENCES

[1] Wei Fan and Albert Bifet, “Mining big data: current status, and forecast to the future”, ACM SIGKDD Explorations Newsletter, Vo.14 Issue 2, pp.1-5, 2012.

[2] Heng Li and Richard Durbin, “Fast and accurate short read alignment with Burrows-Wheeler transform”, Bioinfomatics, Vol.25, No.14, pp.1754-1760, 2009.

[3] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi and Y. Shi, “Parallel data mining techniques on Graphics Processing Unit with Compute Unied Device Architecture (CUDA)”, Highly-Efficient Accelerators and Reconfigurable Technologies, Journal of Supercomputing, Vol. 64, Issue 3, pp 942-967, 2011.

[4] H. M. Waidyasooriya, M. Hariyama, M. Kameyama, “Implementation of a custom hardware-accelerator for short-read mapping using Burrows-Wheeler alignment”, Conf Proc IEEE Eng Med Biol Soc., pp.651-654, 2013.

[5] Thomas C. Conway, and Andrew J. Bromage, “Succinct data structures for assembling large genomes”, Highly-Efficient Accelerators and Reconfigurable Technologies, Bioinformatics, Vol.27, No.4, pp.479-486, 2011.

[6] Philip Gage, “A New Algorithm for Data Compression”, C/C++ Users Journal, 12(2), pp23-28, 1994.

[7] M. Burrows and D.J. Wheeler, “A block-sorting lossless data compression algorithm”, Technical report 124, Palo Alto, CA, Digital Equipment Corporation. 1994.

[8] P. Ferragina and G. Manzini, “Opportunistic data structures with applications”, Proc. of 41st Symp. on Foundations of Computer Science, pp.390-398, 2009.

[9] Hasitha Waidyasooriya, Masanori Hariyama and Michitaka Kameyama, “FPGA-Accelerator for DNA Sequence Alignment Based on an Efficient Data-Dependent Memory Access Scheme”, Highly-Efficient Accelerators and Reconfigurable Technologies, pp.127-130, 2014.