Reducing Symbol Search Overhead on Stream-based Lossless Data Compression

Shinichi Yamagiwa[1], Ryuta Morita[2], and Koichi Marumo[2]

[1] Faculty of Engineering, Information and Systems
[2] Department of Computer Science
University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, JAPAN
yamagiwa@cs.tsukuba.ac.jp, {morita, marumo}@padc.cs.tsukuba.ac.jp

Abstract. Lossless data compression is emerged to utilize in the BigData applications in the recent days. The conventional algorithms mainly generate a symbol lookup table to replace a frequent data pattern in the inputted data to a symbol, and then compresses the information. This kind of the dictionary-based compression mechanism potentially has an overhead problem regarding the number of symbol matchings in the table. This paper focuses on a novel method to reduce the number of searches in the table using a bank separation technique. This paper reports design and implementation of the bank select method on the LCT-DLT, and shows the performance evaluations to validate the effects of the method.

Keywords: Lossless Data Compression · Stream-based Data Compression · Dictionary-based Compression · Interconnection.

1 Introduction

Increasing the demands for handling BigData applications, it becomes one of the important techniques to processing a huge size data in computer systems. Because it is impossible to reduce the fast generation rate of the BigData itself, one of the ideal solutions is to minimize the data by applying the lossless data compression algorithm.

Lossless data compression algorithm has been studied for these three decades regarding mainly the *dictionary-based* compression. The LZW [3] is one of the well-known algorithms. Those algorithms exploit the frequent data patterns and create a dictionary that contains replacement rules to smaller data. Using the rules, the compressor generates a compressed data sequence. Decompressor decodes the sequence using the dictionary. However, it has fatal disadvantage due to the processing style. When we consider to apply the algorithm to a data stream such as continuous sensor data, we need to prepare a data buffer for the dictionary. The size of the dictionary is not deterministic and decompression is blocked until the dictionary is prepared fully after the compression processes entire input data. Moreover, the data stream must be terminated in chunks to

DOI: 10.1007/978-3-030-22750-0_59

generate the dictionary. Finally, it can not continuously compress and decompress in pipeline manner. Thus, the conventional dictionary-based algorithms are not suitable for realtime compression for the continuous data stream.

We have developed a new compression algorithm called *LCA-DLT* [2]. It is able to compress data stream without buffering and blocking inputted data to the compressor and the decompressor. The compression algorithm applies an idea that any data stream can be expressed by a binary tree, and compresses a data pair to a compressed symbol. Cascading modules of the two-to-one compression, it compresses long data patterns. Using a dynamic histogram management for the symbol lookup table under the limited number of entries in the table, it can compress data stream in a pipeline manner and hides information to reproduce its symbol lookup table in compressed data. In decompressor side, when it receives any compressed data, it reproduces a histogram of the symbol lookup table and decompresses the data to originals. This mechanism does not need any buffer for creating entire symbol lookup table due to the hided reproducible information for the histogram. Moreover, it can compress a continuous data without scattering original data stream. This mechanism is suitable for hardware implementation due to availability of parallel pattern matching in the symbol lookup table. However, when we consider software implementation, it needs to search compressing or decompressing symbol pairs from the symbol lookup table sequentially. When the number of patterns in the table is N, it never avoids the overhead of the search operations from $O(N)$. When the number of cascaded modules increases, it linearly increases. Thus, using LCA-DLT algorithm, this paper will try to reduce the searching overhead in the dictionary-based lossless compression mechanism.

2 Stream-based lossless data compression algorithm: LCA-DLT

The table of LCA-DLT has any number N of entries and the i-th entry E_i includes a pair of the original symbols ($s0_i$, $s1_i$), a compressed symbol S_i, and frequent counter $count_i$. The compressor side uses the following rules: 1) reading two symbols ($s0$, $s1$) from the input data stream and if the symbols match to $s0_i$ and $s1_i$ in a table entry E_i, after counting up the $count_i$, it outputs S_i as the compressed data, 2) if the symbols do not match to any entry in the table, it outputs ($s0$, $s1$) and register an entry $(s0_k, s1_k, S_k, count_k = 1)$ where S_k is the index number of the entry, and 3) if all entries in the table are used, all $count_i$ where $0 \leq\ i < N$ are decremented until any count(s) become zero and the corresponding entries are deleted from the table.

In the decompressor side, assume that a compressed data S is transmitted from the compressor, the subsequent steps are equivalent to the compressor's, but the symbol matching is performed based on S_k in an entry. If the compressed symbol S matches to S_k in a table entry, it outputs ($s0_k$, $s1_k$). If not, it reads another symbol S' from the compressed data stream and outputs a pair of (S, S') and then the pair is registered in the table. When the table entry is full, the same operation as the compressor side is performed. These operations provide a reproducible dynamic histogram on a limited number of lookup table entries.

Here, let us focus on overhead of the symbol matching operation in the lookup table when we implement LCA-DLT in software. LCA-DLT supports two-to-one comparisons. It allows a fixed number of matching operations in the table entries. However, the number of entry search operations in the symbol lookup table is not predictable because it is inevitably performed by sequential search from the top of the table. Unfortunately, it occupies a large part of the entire compression and decompression operations.

When we consider to divide the symbol lookup table into multiple groups, we can expect to reduce the number of search operations in the table. However, it is not so simple in the case of the conventional dictionary-based compression algorithms because the matching complexity of the search operations is expected in the worst case as $O(NM)$ where the N is the number of entries in the table and the M is the length of matching symbols in an entry. Both of M and N are very variable according to the inputted data. Besides, the number of symbol matching is fix to two in LCA-DLT. Therefore, the complexity of the search operation $O(2N)$ where N is the number of the table entries and also is defined as a fixed value. This means that we can expect to reduce the entire compression and decompression processing times drastically if we divide the number of the table entries by the number of groups (i.e. N/k where the k is the number of groups). In this paper we will propose the technique to divide the symbol lookup table into multiple groups and the technique will effectively reduce the number of search operations. We will call the groups the *banks of symbol lookup table*, and will propose the speedup technique using the LCA-DLT based on the bank select method for the symbol lookup table.

3 Bank select method for stream-based lossless data compression

Now, let us explain the bank select method using LCA-DLT. The technique is quite simple. The symbol lookup table is divided into N_b banks. When a data pair arrives, it is associated to one of the banks in the table. Then LCA-DLT algorithm is applied to the bank. When the number of the table entries is N, the number of entries related to the compression is N/N_b. For example, when the table has 256 entries and the number of banks is 16, LCA-DLT will manage the dynamic histogram operation in a bank with 16 entries. Thus, in the case of LCA-DLT, the complexity of the number of search operations becomes $O(N/N_b)$. The decompressor has the same organization in symbol lookup table.

Let us consider how a bank is selected. Here, we employ two simple selection methods; order major and data major selection. One is the sequence index i of the inputted data stream. In this case the *hash* function becomes an equation with i, for a simple example, such as the round-robin selection $n_b = i \bmod N_b$. This hash function has the characteristics that there is no relationship to the inputted data itself. Therefore, the bank indices are totally selected among the entire table

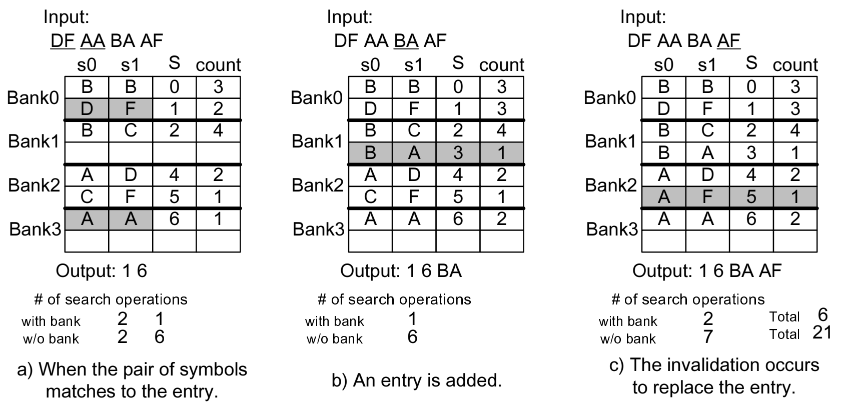

Fig. 1. Compression example of LCA-DLT with the bank select method.

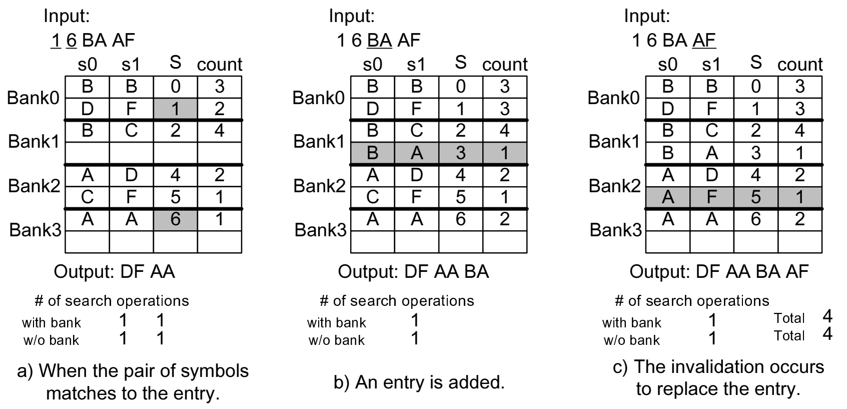

Fig. 2. Decompression example of LCA-DLT with the bank select method.

entries even if the entropy of the data is dynamically changing. This causes multiple assignments of the table entries of the same data pair among multiple banks. Another function uses a part of the inputted data itself. In this case, the selected bank indices are spread among the entire table entries according to the entropy of the inputted data stream. Let us introduce a simple example to keep fast hash calculation such as $n_b = \sum_{k=0}^{K/2-1} 2^k \cdot s_{0,k} + \sum_{k=K/2}^{K-1} 2^k \cdot s_{1,k-K/2}$ where $K = \log_2 N_b$ and $s_{i,j}$ is a bit of s_i. n_b is combined by $\log_2 K$ bits made from the LSBs of inputted symbols (s_0, s_1). Here, the combined bits can be generated by picking up the least significant $K/2$ bits from s_0 and s_1 respectively. In this case, the bank selection will be depending on the entropy of the entire data elements. This would cause data concentration in several banks if the number of data combinations is few (i.e. the entropy of the data is low). We will evaluate the effects of the hash methods in the performance evaluation.

We shall show examples of the compression and the decompression flows. Fig. 1 and Fig. 2 illustrate examples under the conditions: N and N_b are 8 and 4 respectively, and the hash function is the n_b of the data major approach. We assume that the inputted data stream consists of ASCII characters in 8bits. The combined K bits are organized by the least significant bit of each ASCII code of the input data pair. In this case, the least significant bit of each symbol in the pair is combined and finally the two-bit bank number n_b is decided.

Regarding the compression operation, we assume that after several data has been processed and then the part of the stream "DFAABAAF" is inputted to the compressor. Fig. 1 a) shows the case when the first two pairs match to the table entries. Here, we count the numbers of search operations with/without the bank select method. In the case of "DF", the number of search operations is two in both cases. However, in the case of "AA", because the bank number is 3 decided by the combination of the LSB of "A"s (0x41). Therefore, the bank select method finds the entry from the first search in the bank. However, in the case without banks, it searches the entry from the top of the table. It needs six search operations. Fig. 1 b) shows the case when the pair is inserted to a bank. In order to know that the inputted data pair must be inserted to an empty entry, the case without bank needs to search all the entries in the table, and then it inserts a new entry. Therefore, it needs 6 search operations. Besides, the case with bank performs just a search operation in a bank. Finally, Fig. 1 c) shows the case of invalidation operation to replace the entry. It is the same as the case of the insertion operation. It needs to count maximally the number of entries in a bank. However, without the bank, it needs to search all the entries in the table and finds an entry to replace it. According to the total number of the search operations in the compression example, the one without banks becomes almost four times more search operations than the one with banks. Thus, the complexity of the search operation becomes approximately $O(N/N_b)$.

On the other hand, regarding the decompression operation, the decompressor reads an inputted symbol and searches the entry associated by the symbol value as the index of the table. In the case when the compressed symbols match to the entries as shown in Fig. 2 a), the numbers of search operations for '1' and '6' are both one time. This is the same when the case without bank. The new entry insertion is performed for the table such as the case of "BA" as shown in Fig. 2 b), after the decompressor searches the index 'B' in the table. However, 'B' as the index of the table is more than the number of entries. It knows that there is not any matched entries and then insert it to a new entry. Therefore, the number of search operations is always one. Finally, in the case of invalidation of the entry as shown in Fig. 2 c), the number of search operations is the same as the insertion. Thus, the numbers of search operations with/without the bank select method are totally the same.

As we have seen in the examples above, we have confirmed that the bank select method has the effective speedup technique for the compression. Even if the processing speed is fast, it is not a novel technique if the effective compression ratio degrades than the compressor without the bank select method. In the next

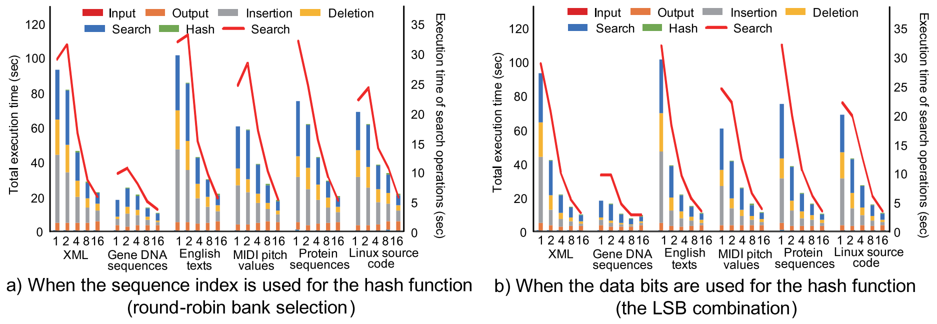

Fig. 3. Comparisons of compression times of LCA-DLT with the bank select method.

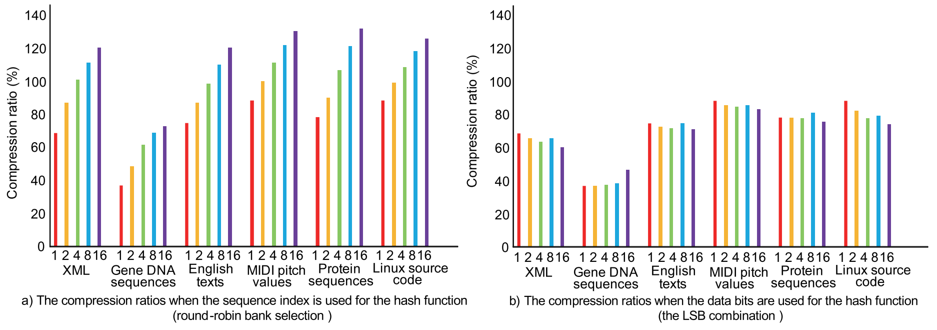

Fig. 4. Comparisons of compression ratios of LCA-DLT with/without the bank select method among different hash functions.

section, we will discuss the actual computing speed and the compression ratio to validate the proposed technique.

4 Performance evaluation

We used several patterns of 10Mbyte benchmark data of a text collection available from [1]. We have implemented compressor/decompressor with/without the bank select method using C#. The execution time measurement has been performed in a 64bit Windows 8 PC with Intel Core i7 2.7GHz and 8GByte memory.

First, we measured the elapsed times of individual operations in the compression process: *Insertion* is the time for inserting a new symbol pair entry to the table, *Deletion* is the one for deleting an entry(s) from the table by the invalidation operation, and *Search* is the one for searching operation for the input pair, as shown in Fig. 3 when four compressor modules are cascaded. The bars show the execution times of the individual operations with varying the number of banks from 1 to 16. The total number of entries in the table is configured to 256. In the case when the number of banks is 1 as shown in the graph, it is precisely equal to the case without banks. The line shows the speedup ratio where the execution time of search operations is divided by the one without banks. We have evaluated the speedup applying different kinds of the hash function: one is the hash function based on sequence index, and another is based on a bit combination of inputted data.

According to the graphs, in both cases of the different hash functions, we have confirmed that the search operation occupies a large part of the total execution. Therefore, when the number of banks is increased, the time of the search operations is linearly decreased for two times when it is changed to the twice number of the banks. The total execution times of Fig. 3 a) are larger than the ones of Fig. 3 b). This means that insertion and deletion times are larger because the compression ratios become worse as we will discuss in the next evaluation.

Next, Fig. 4 shows the compression ratios. The compression ratios in the case when the hash function uses the sequence index become worse when the number of banks is increased. The hash function selects all the banks fairly. Because the invalidation operations occur frequently among all the banks, the table search often does not match the input pair to the corresponding one in the table. On the other hand, in the case when the hash function uses a bit combination of symbol pair. it results better compression ratios than the round-robin case. When the hash function uses the data itself, the bank selection is affected directly by data entropy. Therefore, the corresponding symbol pair is always mapped to the same bank. This causes high probability to match the inputted symbol pair to the entry in the bank.

5 Conclusion

We have proposed a novel method to speedup lossless compression operation called the bank select method. It divides the symbol lookup table to multiple banks. The banks are selected by the hash function. We applied it to the stream-based lossless data compression algorithm called LCA-DLT. The evaluation has shown the drastic improvement of execution time. We have confirmed that the method is effective to reduce the search operations in compression.

Acknowledgement

This work is partially supported by JSPS KAKENHI Grant Number 15H02674 and JST CREST Grant Number JPMJCR1402.

References

1. http://pizzachili.dcc.uchile.cl/
2. Yamagiwa, S., Marumo, K., Sakamoto, H.: Stream-based Lossless Data Compression Hardware using Adaptive Frequency Table Management. In: Proceedings of the VERY LARGE DATA BASES / BPOE 2015, LNCS 9495. Springer (2015)
3. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23**(3), 337–343 (May 1977). https://doi.org/10.1109/TIT.1977.1055714