Streaming Sorting Network Based BWT Acceleration on FPGA for Lossless Compression

Baofu Zhao, Yubin Li, Yu Wang, Huazhong Yang
Electronic Engineering Department, TNLIST, Tsinghua University, Beijing, China
{zbf15,liyb13}@mails.tsinghua.edu.cn {yu-wang, yanghz}@tsinghua.edu.cn

***Abstract*—The Burrows-Wheeler Transform (BWT) has received special attention due to its effectiveness in lossless data compression algorithms. Because BWT is a time-consuming task, the efficient hardware accelerator that can yield high throughputs is required in real-time applications. This paper presents a novel BWT accelerator based on the streaming sorting network. The streaming sorting network performs the suffix sorting of large amount of data which is the most difficult task in BWT. Our BWT accelerator is implemented on a NetFPGA board. Experimental results show that it achieves 14.3X speedup compared with the state-of-art work when the data block size is 4KB. Furthermore, we design and implement a lossless data compression system based on the proposed BWT accelerator. The hardware system is composed of Burrows-Wheeler Transform module, the move-to-front encoding module, the run length encoding module, and the canonical Huffman encoding module. We evaluate the system performance on a NetFPGA board at the frequency of 155MHz. The throughput of the system could reach 179 MB/s on board when we use only one streaming sorting network for a 4KB block. The system throughput can be linearly improved up to 537 MB/s in simulation on a Virtex UltraScale xcvu440 chip if we use three streaming sorting networks to compute BWT.**

I. Introduction

With the explosive growth of global data, data transmission and storage have been a critical problem in data centers. This increases the demand for efficient compression systems that can remove redundant features in raw data and improve the efficiency of data transmission and storage. However, the great amount of calculation during compression will bring a heavy burden to the CPU and will introduce a significant delay to the traditional transmission and storage system. In order to better address the CPU load and delay problems in compression, the dedicated compression accelerator becomes one of the possible solutions. Using FPGA or ASIC as a carrier, and combining highly concurrent hardware design, the accelerators can solve these problems, and maximize the availability of compression algorithms in a large amount of data transmission and storage scenarios.

The Burrows-Wheeler Transform (BWT), invented by Burrows and Wheeler in 1994 [1], also known as block sorting, is a high compression ratio method that originally used for lossless compression of text [2]. Although BWT itself does not compress the data, it changes the context features of the data so that the same characters are clustered together as much as possible. Combined with subsequent run-length encoding and entropy coding [3], it can achieve a very high compression ratio. This approach has been adopted in bzip2 compression [4], and software tests have shown that the compression ratio is much higher than gzip.

This work was supported by National Natural Science Foundation of China (61373026, 61622403, 61621091), National Key R&D Program of China (2017YFA0207600), Huawei Technologies Co. Ltd, and Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608).

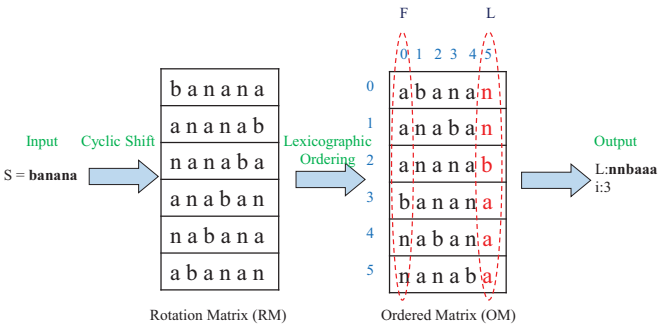

Fig. 1. An example of BWT on a string [1].The BWT result of the string banana is nnbaaa and 3.

The basic idea of BWT is straightforward. As shown in Figure 1, we first cyclically shift the text block that needs to be converted, one character at a time. For a text block of length N, we get N strings of length N. Then, we sort these strings in the lexicographic order and store them in a Ordered Matrix (OM). The last column L of matrix OM along with i is the BWT output of the text block, where i represents the position of the original block in OM.

In this paper, we first propose a novel FPGA design for BWT acceleration and then apply it to lossless data compression to achieve a complete BWT-based compression system. The main contributions of this paper include:

- We for the first time propose an architecture to accelerate BWT on FPGA by using the streaming sorting network, and the evaluation on the NetFPGA shows that it achieves 14.3X speedup compared with the state-of-art work when the BWT block size is 4KB.
- We design a BWT-based lossless data compression system with high throughput and low latency. The maximum clock frequency of the entire system can reach 155 MHz on the Xilinx NetFPGA board. The throughput of the system could reach 179 MB/s on board when we use only one streaming sorting network to compute BWT. It can be linearly improved up to 537 MB/s in simulation on a Virtex UltraScale xcvu440 chip when we use three

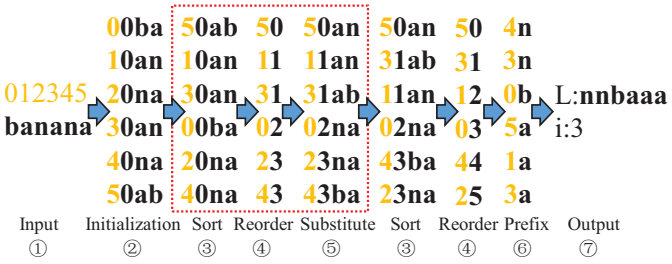

Fig. 2. Compute BWT based on the suffix sorting approach [5]. The yellow numbers represent the index and do not participate in sorting.

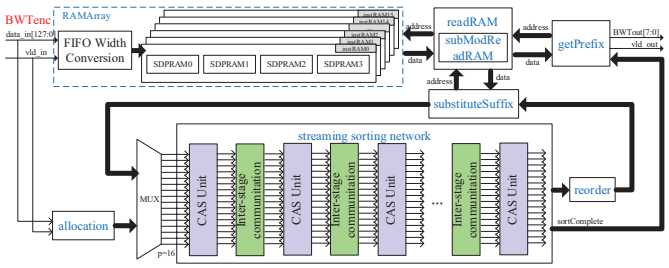

Fig. 3. The implementation of BWT for a single block.

streaming sorting networks in parallel to compute BWT.

In order to describe our approach, the remainder of this paper is organized as follows: Section II discusses previous hardware implementations. Section III details the proposed approach and architecture to accelerate BWT with the streaming sorting network. Section IV presents the BWT-based lossless data compression system. Section V shows the experimental results and analyses. Finally, conclusions are presented in Section VI.

II. Related Work

There is some previous work about the hardware implementation of BWT. The main difference focused on the approaches taken to sort the input strings, which is one of the most time-consuming tasks in BWT.

Mukherjee [6] adopts a Weavesorter-based architecture to sort the strings on FPGA. The Weavesorter begins by right shifting a string and comparing it during each step. The sorted string is obtained by left shifting the stored data. However, the same characters should be substituted via software operations in order to complete BWT, and the whole string should be sorted again. Martinez [5] uses a parallel sorting strategy to compute BWT. Although strings can be sorted in *n/2* steps, the ordered string should be output completely and sorted again with substituted data. By using a linear sorter, Prez-Celis [7] proposes a new BWT architecture on FPGA. Their architecture consists of control logic and several identical comparison units (CU) which are used to store and compare data. Compared with the previous designs, this architecture can significantly reduce the computing cycle of BWT, but it can only use one suffix character to substitute the same characters at a time, thus resulting in using more iterations to compute BWT.

All work mentioned above have implemented BWT on FPGA, but their main drawback is that the text block size that their BWT designs can process is too small while the time used in transformation is too long. In their experiments, it needs 1128 cycles or more to compute BWT of 128-byte block. In order to increase the data block size, while reducing the transformation time, we propose three solutions. The first is using the streaming sorting network to sort a large amount of data, thus increasing the block size. The second is using multiple suffix characters to substitute the same characters at a time, thus reducing the number of iterations. The third is supporting BWT computation of multiple text blocks in parallel, thus reducing the total sorting time.

III. BWT Hardware Accelerator

A. Implement BWT for a Single Block

Direct implementation of BWT on hardware requires lots of resources, especially memory resources to store string rotation matrix. In order to avoid the generation of the rotation matrix, the suffix sorting approach [5] is often used to carry out BWT in practice. Figure 3 shows the hardware framework of our proposed BWT accelerator that performs BWT of a 4KB data block. The BWT accelerator is composed of an **RAMArray** module for storing data, a **ReadRAM** module for reading data, the streaming sorting network, and several character control modules.

The streaming sorting network is composed of $(\log N)(\log N + 1)/2$ stages of compare-and-swap (CAS) units and inter-stage communication modules [8] and it can sort an *N*-key sequence by merging the bitonic sorting network recursively. We use a third party IP [9] for the streaming sorting network implementation on FPGA. Its parallelism is 16 in our architecture, and it needs 1247 cycles to sort a 4KB sequence of arbitrary width.

These character control modules are used to allocate data and control operations among data to compute BWT as the steps showing in Figure 2. The **allocation** module initializes 16 data sequences according to the input and the statistical index and sends them to the sorting network. The **reorder** module is used to determine each data's serial number in the data sequence from the streaming sorting network, thereby changing their serial numbers which all are zero on initialization, and determines whether the suffix sorting is done based on whether there is data of the same serial number. If the suffix sorting is not completed, the **substituteSuffix** module replaces these data with their four suffix characters according to the index, and then send the new data to the network for reordering. Otherwise, the **sortComplete** signal will be set to high level and the data will be fed into the **getPrefix** module, and then the corresponding prefix characters are taken according to the index to obtain the final BWT result. The BWT result is outputted byte-by-byte from the **getPrefix** module by a total of 4098 bytes, of which the first 4096 bytes are the result of BWT of the 4KB block, and the last two bytes represent the position of the first character of the block.

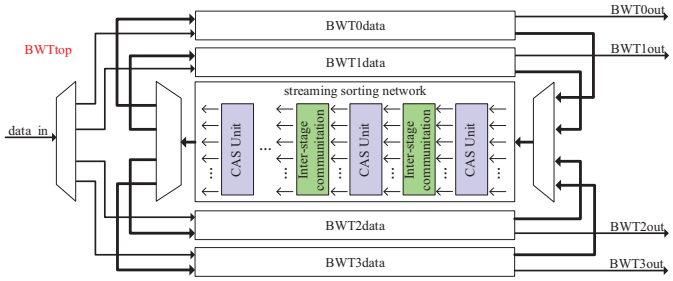

Fig. 4. The implementation of BWT for multiple blocks.

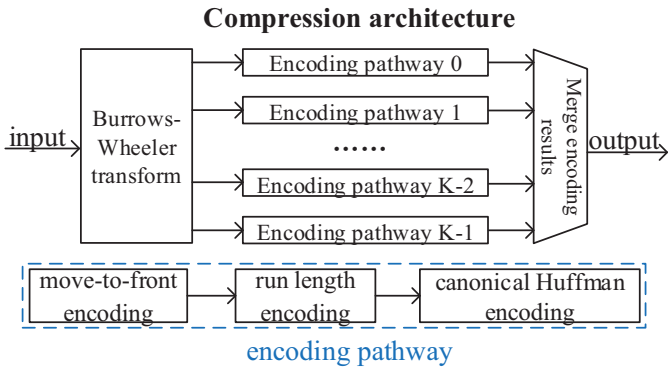

Fig. 5. The hardware architecture of BWT-based lossless data compression.

Because the data parallelism p of the sorting network is 16, we store the 4KB block data in 16 RAM groups. The contents in each RAM group are the same in the **RAMArray** module, which is convenient to read 16 different data simultaneously. Each RAM group has four RAM output ports, and it can output four data once a time so that we can substitute four consecutive suffix characters in the **substituteSuffix** module.

The **ReadRAM** module is responsible for reading data from the **RAMArray** module. It first calculates the RAM address from the output index of the **substituteSuffix** module or the **getPrefix** module and then transfers the corresponding read data back to the corresponding module.

B. Implement BWT for multiple blocks

Since the sorting network is fully streaming and its latency is 1247 cycles, but it takes only 256 cycles to receive a 4KB data block, so we can consecutively send four 4KB data blocks to the sorting network when it is sorting the previous block. Based on this, we can implement BWT of four 4KB data blocks in a pipeline on hardware by sharing one streaming sorting network. Thus, the bandwidth of the sorting network can be fully utilized, and the throughput of the module can be improved.

As shown in Figure 4, the **BWTtop** module can processes four 4KB blocks in the pipeline. There are four same **BWTdata** modules and one sharing streaming sorting network, and the streaming sorting network is time-division multiplexed by four **BWTdata** modules. Each **BWTdata** module processes one data block, and it includes the **RAMArray** module, **readRAM** module and character control modules described in Figure 3. After the previous data block is fed into **BWTtop**, the next data block can be fed. After all four data blocks are received, the streaming sorting network outputs the results of the first block, then the second. If the suffix sorting of the first block is not completed, these output data are replaced with their suffix characters and sent to the streaming sorting network again. This phase keeps on until the sorting task of all four blocks is completed. BWT computation on different blocks may need different numbers of sort times. We set the number of sort times to a fixed value, which satisfies that it can complete all BWT processing of the four blocks.

IV. Lossless Data Compression Based On BWT

We design a lossless data compression hardware system based on the proposed BWT accelerator. The compression system performs different encoding algorithms, including Burrows-Wheeler Transform, move-to-front coding, run length encoding and the canonical Huffman coding [10]. Each encoding function in the compression system is reversible and can be used to decompress data. We use the Silesia Corpus data set to evaluate our compression system. The software testing shows that the compression ratio increases with the growth of the block size in BWT when other coding methods remain unchanged. It is worth noting that our algorithm's compression performance outperforms the gzip level 9 when the block size is greater than 50K bytes.

The corresponding hardware compression framework is shown in figure 5. The text data carry out the Burrows-Wheeler transform block-by-block in the **BWTtop** module. When the **BWTtop** module uses one streaming sorting network to compute the Burrows-Wheeler transform, it can output $\mathrm{K} = 4$ BWT results almost simultaneously. If the **BWTtop** module uses three streaming sorting networks to compute the Burrows-Wheeler transform in parallel, K can be set to 12. All BWT results of different blocks will be sent to the posterior encoding pathway, and move-to-front encoding, run length encoding and canonical Huffman encoding are preformed on them successively. After the encoding is completed, the **Merge-encoding-results** module combines the encoding results in sequence and produces the compressed result.

V. Experimental Results

A. Results of our BWT accelerator

The execution time of BWT on processing a data block depends on the number of sort times, which is related to the length of the longest repeat substring in the data block. The number of sort times is similar to the number of iterations "k" in [7], but they can only replace one character at a time while we can substitute four characters every time. As a result, the number of sort times we used is 1/4 of the number of iterations in [7]. The LSBWT in [7] is only capable of sorting a string of a length equal to the number of CUs, so the text block size

TABLE I
COMPARISON OF THE EXECUTION STEPS AND MAXIMUM CLOCK FREQUENCY OF BWT ARCHITECTURES.

Architecture	Expression of cycles	Cycles for n=4096,k=8	Steps count (normalized)	Maximum Frequency	Device
Weavesorter [6]	2n(k+1)	73728	18.10	45MHz	Virtex xcv300
Parallel Sorter [5]	(3k/2+1)n	53248	13.15	51.6MHz	Virtex-2 xv2v2000
Linear Sorter [7]	n(k+1)+2k	36880	9.11	152MHz	Kintex-7 XC7K70T
Sorting Network we proposed	1262(k/4)+1525	4049	Base line = 1	239MHz	Virtex-7 xc7vx690t

for BWT in [7] is limited. In their experiments, the text block size is only 128 bytes. However, we implement BWT of a 4KB block by using the streaming sorting network, and can improve the compression ratio from 1.55 to 2.49 when used in lossless compression system based on BWT.

Table I compares the number of cycles taken by the architectures proposed in [6], [5] and [7] and their maximum clock frequencies. Table I also shows the increase in the number of steps in relation to the proposed approach. From the results shown in Table I, it can be seen that the sorting network based BWT accelerator we proposed outperforms other BWT architectures in the execution cycles and the maximum clock frequency. Our execution cycles are only one-ninth of executions steps of the state-of-art work [7]. Our clock frequency can reach 239 MHz, and it is a 1.57X speedup compared with [7]. On the whole, our BWT accelerator achieves 14.3X speedup compared with the state-of-art work when the data block size is 4KB.

B. Performance analysis of the lossless compression system

We have synthesized and implemented the overall compression system with RTL on a Virtex-7 xc7vx690t chip and verified it on the NetFPGA board. Due to the resource constraints, the compression system only uses one streaming sorting network and performs BWT of four 4KB blocks in a pipeline in the **BWTtop** module. The clock frequency of the whole system can reach 155MHz, and the throughput of the whole system is 179MB/s. All the part except move-to-front encoding can run at 200MHz or more.

Based on the system above, we use three streaming sorting networks in parallel in the **BWTtop** module to compute 12 ways of the Burrows-Wheeler transform, and the posterior encoding pathway is also copied 12 copies accordingly. We did the behavioral simulation on a Virtex UltraScale xcvu440 chip, and the performance analysis of each part is as follows:

1) The execution time of BWT for a 4KB block is $(1262 * (k/4) + 1525)$ cycles. Assuming $k/4$ is ten that suits most of the text blocks, the entry performance of the BWT part can reach $\frac{4096*12}{1262*10+1525} = 3.47$ bytes/cycle. The maximum clock frequency of BWT part can be 239 MHz, and the corresponding throughput is 829 MB/s.
2) Both the MTF part and the RLE part can process one character per cycle, so it can reach 12 bytes/cycle when using 12 copies in parallel. The clock frequency of RLE can be up to 251MHz, but the MTF part can only reach 164 MHz which reduces the overall system clock frequency.
3) Both data input and encoding are performed character-by-character in the canonical Huffman encoding module, so it takes about 9728 cycles to encode 4KB data. The 12 same modules can achieve $\frac{4096*12}{9728} = 5.05$ bytes/cycle, and the clock frequency is 257MHz.

It can be seen that the overall compression performance can reach 3.47 bytes/cycle by using three streaming sorting networks to execute BWT. The corresponding throughput can be linearly improved up to 537MB/s in simulation when the system runs at 155MHz on a Virtex UltraScale xcvu440 chip.

VI. CONCLUSION

This paper presents a novel Burrows-Wheeler transform accelerator based on the streaming sorting network. The results show that the architecture can significantly increase the size of the BWT block while significantly reducing the BWT latency. Furthermore, we design and implement a BWT-based lossless data compression system on the NetFPGA board at the frequency of 155MHz. The throughput of the system could reach 179 MB/s on board when we use only one streaming sorting network for a 4KB block. The throughput can be linearly improved up to 537 MB/s in simulation on a Virtex UltraScale xcvu440 chip if we use three streaming sorting networks to compute BWT.

REFERENCES

[1] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.
[2] P. Fenwick, "Block sorting text compression–final report," Department of Computer Science, The University of Auckland, New Zealand, Tech. Rep., 1996.
[3] http://marknelson.us/1996/09/01/bwt/.
[4] http://www.bzip.org.
[5] J. Martinez, R. Cumplido, and C. Feregrino, "An fpga-based parallel sorting architecture for the burrows wheeler transform," in *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on*. IEEE, 2005, pp. 7–pp.
[6] A. Mukherjee, N. Motgi, J. Becker, A. Friebe, C. Habermann, and M. Glesner, "Prototyping of efficient hardware algorithms for data compression in future communication systems," in *Rapid System Prototyping, 12th International Workshop on, 2001.* IEEE, 2001, pp. 58–63.
[7] J. A. Pérez-Celis, J. Martínez-Carranza, A. Morales-Reyes, C. Feregrino-Uribe, and R. Cumplido, "An fpga architecture to accelerate the burrows wheeler transform by using a linear sorter," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 2016, pp. 156–161.
[8] R. Chen, S. Siriyal, and V. Prasanna, "Energy and memory efficient mapping of bitonic sorting on fpga," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 240–249.
[9] http://www.spiral.net/hardware/sort/sort.html.
[10] J. Matai, J.-Y. Kim, and R. Kastner, "Energy efficient canonical huffman encoding," in *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*. IEEE, 2014, pp. 202–209.