Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression

Sang Muk Lee, Ji Hoon Jang, Jung Hwan Oh, Ji Kwang Kim, and Seung Eun Lee^{a)}

Dept. of Electronic Engineering, Seoul National University of Science and Technology, 232 Gongreung-ro, Nowon-gu, Seoul, 139–743, Korea

a) seung.lee@seoultech.ac.kr

Abstract: Hardware accelerators are being considered as important architectural components in the context of datacenter customization to achieve high performance and low power. Compression has played an important role in computer systems by enhancing storage and communication efficiency in the charge of extra computational cost. In this letter, we present a fully pipelined compression accelerator for the Lempel-Ziv (LZ) compression algorithm. The compression accelerator is verified by using FPGA and fabricated using 65 nm CMOS technology.

Keywords: data compression, LZ4, hardware accelerator, high throughput

Classification: Integrated circuits

References

- [1] A. M. Caulfield, *et al.*: “A cloud-scale acceleration architecture,” IEEE/ACM International Symposium on Microarchitecture (2016) (DOI: [10.1109/MICRO.2016.7783710](https://doi.org/10.1109/MICRO.2016.7783710)).
- [2] M. Staveley, “Applications that scale using GPU compute,” AzureCon (2015).
- [3] J. Ouyang, *et al.*: “SDA: Software-defined accelerator for large-scale DNN systems,” HotChips (2014) (DOI: [10.1109/HOTCHIPS.2014.7478821](https://doi.org/10.1109/HOTCHIPS.2014.7478821)).
- [4] AHA378 (2015) <http://www.aha.com/data-compression/>.
- [5] S. M. Lee, *et al.*: “A hardware scheduler for multicore block cipher processor,” PDP (2016) (DOI: [10.1109/PDP.2016.59](https://doi.org/10.1109/PDP.2016.59)).
- [6] B. Gu, *et al.*: “Biscuit: A framework for near-data processing of big data workloads,” ISCA (2016) (DOI: [10.1109/ISCA.2016.23](https://doi.org/10.1109/ISCA.2016.23)).
- [7] B. Reagen, *et al.*: “Minerva: Enabling low-power, highly-accurate deep neural network accelerators,” ISCA (2016) (DOI: [10.1109/ISCA.2016.32](https://doi.org/10.1109/ISCA.2016.32)).
- [8] A. Martin, *et al.*: “FPGA-based application acceleration: Case study with gzip compression/decompression streaming engine,” ICCAD (2013).
- [9] J. Fowers, *et al.*: “A scalable high-bandwidth architecture for lossless compression on FPGAs,” IEEE International Symposium on Field-Programmable Custom Computing Machines (2015) (DOI: [10.1109/FCCM.2015.46](https://doi.org/10.1109/FCCM.2015.46)).
- [10] M. S. Abdelfattah, *et al.*: “Gzip on a chip: High performance lossless data compression on FPGAs using OpenCL,” International Workshop on OpenCL (2014) (DOI: [10.1145/2664666.2664670](https://doi.org/10.1145/2664666.2664670)).
- [11] J. Ziv and A. Lempel: “A universal algorithm for sequential data compression,” IEEE Trans. Inf. Theory **23** (1977) 337 (DOI: [10.1109/TIT.1977.1055714](https://doi.org/10.1109/TIT.1977.1055714)).

1 Introduction

Hardware accelerators are being considered as important architectural components in the context of datacenter customization to achieve high performance and low power. Both GPUs and FPGAs have been deployed in datacenter thanks to its programmability because accelerators should be compatible with the target through its lifetime. By exploiting the reconfigurable nature of FPGAs, efficiency and performance of custom hardware can be delivered, without the complexity of deploying fully customized accelerators into the system. Microsoft proposed the Catapult FPGA platform by putting in FPGA cards both in I/O space as well as between a server's NIC and the local switch for datacenters and demonstrated significant performance improvement for the Bing web search [1]. GPU accelerated computing has been proposed in the variety of fields such as deep learning, analytics, and engineering applications, offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU [2, 3]. In addition to FPGAs/GPUs, application-specific hardware accelerators are being integrated into platforms for widely used workloads such as compression [4], cryptography [5], big data [6], and deep learning [7].

Over the past several decades, compression has played an important role in computer systems by enhancing storage and communication efficiency in the charge of extra computational cost. Compression is one of candidate for hardware acceleration, because a specialized accelerator should incorporate with variety of workloads through its lifetime. IBM implemented DEFLATE compression accelerator on FPGA [8]. Microsoft and Altera presented an LZ77 accelerators on FPGA [9, 10]. Microsoft introduced AHA presented GZIP compression accelerator ASIC, which supports up to 80 Gbps throughput [4]. These accelerators are useful for computational offloading, enhancing an order-of magnitude throughput over modern CPUs. However, they are resource intensive. In this letter, we present a fully pipelined compression accelerator for the Lempel-Ziv (LZ) [11] compression algorithm which finds duplicate strings in the data and replaces the strings with pointer. Compression algorithms are traditionally forced to make tradeoffs between throughput and compression quality. We adopt LZ4 compression algorithm because LZ4 compression speeds are similar to LZ0 and several times faster than GZIP while decompression speeds can be significantly faster than LZ0. We explore the tradeoffs between compression quality and hardware cost and optimize the architecture of the accelerator focusing on reduced hardware cost and throughput. The compression accelerator is designed in Verilog HDL and the functionality of the accelerator was verified on FPGA. Finally, the compression accelerator was fabricated using 65 nm CMOS technology. Experimental results show that we achieve up to 4 Gbit/sec of throughput with less area overhead. To the best of our knowledge, this is the first work with design of hardware accelerator for LZ4 compression on FPGA and ASIC.

2 Lempel-Ziv4 algorithm

Lempel-Ziv (LZ) compression algorithm exploits statistical redundancy in data patterns and applies a representation to eliminate the repeated literals. The LZ

algorithm finds repeated data strings from previous data and replaces it by a token, deploying the dictionary in order to find out the offset of previous literals and match length. The token is a pointer, which consists of a literal length and matched length with previous literals. In this occasion, the total file size decreases as the replaced literal size is bigger than the token. Fig. 1 illustrates the brief description of LZ4 compression algorithm. The token (6,4) describes that there are 6 bytes of uncompressed literals and 4 compressed data bytes. The offset value 5 represents that the literals, matched with previously compressed literals, have been appeared before the offset value.

Fig. 1. LZ4 compression algorithm

3 Micro-architecture of the compression accelerator

Fig. 2. LZ4 compression engine architecture

Fig. 2 illustrates the LZ4 compression engine architecture. Our hardware compression accelerator is composed of as follows: an architecture. 16 *dictionary*, a *compare_match*, a *compressed_data_rate*, a *manager*, a *position*, and a *FSM* module. The *allocator*, including sliding windows, performs the data allocation to other modules. Then, it allocates strings to each *dictionary*. The *dictionary* compares allocated literals with existing literals to find out the repeated data strings. Every *dictionary* is connected with work signal, which is matched with 1 bit for each *dictionary* in order to control *dictionary* separately. We can achieve the parallelism by deploying dictionary architecture for a hardware implementation. The *compare_match* finds the longest match length from each *dictionary* through 4

compare stages. Then, it deploys the best compression result. This is the most significant principle in the LZ4 compression algorithm. The *compare_data_write* builds the LZ4 data frames using compression result, which is matched in *compare_match* module. After then, it is stored in the output buffer. Therefore, the *compare_data_write* covers the uncompressed literals. The uncompressed literals are temporary saved into internal buffer until the occurrence of the match and LZ4 header. For the fine compression ratio, the enough buffer size would be better. However, the buffer size is limited to output the uncompressed literals for the stall-free architecture. Fig. 3 illustrates the fully pipelined architecture of the proposed hardware accelerator. The hardware accelerator receives 16 bytes of data from its input source every cycle and directs them into our stall-free latency pipeline. Thanks to the no-stall architecture, our hardware compression accelerator is 16 bytes/(20 cycles * period * number\ of cores). The proposed architecture is composed of four major functional components: *Fetch*, *Candidate match*, *Match selection*, *Write*. An operation of each stage is as follows:

- *Fetch*: The parallelization text (processed text at one iteration in parallel) is slid into the current window from the lookahead window, where the current window indicates the text processed in current iteration and the lookahead window indicates the text processed in next iteration. The parallelization text is prepared from plain text while other stages are conducted.
- *Candidate match*: The parallelization text is compared with each dictionary for candidate matching, where 16 match lengths data are calculated.
- *Match selection*: In order to obtain the best compression ratio, the best match length is found among the 16 match results. LZ4 frame composed of the token, the literals, the match length, and the offset is built by using the best match result.
- *Write*: The compressed data is fed to the output buffer through the write logic, which has an extra FSM.

Our hardware accelerator is based on a dictionary architecture. In the LZ4 algorithm, dictionary finds a first matching between the inside dictionary data and current window data. By exploiting both 16 bytes of current window and lookahead window as a buffer for matching with dictionaries, our LZ4 compression accelerator compresses the length of data up to 31 bytes. The first matching process

Fig. 3. Pipelined stages of LZ4 compression engine

is a major cause of overhead. Therefore, we designed the dictionary architecture to reduce a compression time and parallelize the LZ4 algorithm. The proposed dictionary has a short bucket bit to reduce the depth of memory. We used ASCII as a hash function, where the first character of parallelization text in current window data is used. Thanks to the hardware parallelism, we can achieve high compression throughput by exploiting 16 dictionaries architecture. With a single dictionary, the compression engine needs to repeat matching. However, our compression accelerator simultaneously compares current window data with 16 dictionaries. Thus, our hardware accelerator achieves the higher throughput than single dictionary based architecture. Fig. 4 is shown our dictionary architecture of LZ4 algorithm.

Fig. 4. Dictionary architecture of LZ4 compression engine

4 Implementation

To evaluate the proposed hardware accelerator, we implemented the hardware compression accelerator on a Xilinx VC706 evaluation board with a Virtex-7 FPGA. To measure compression throughput, we used a logic analyzer to detect signals indicating the start and completion of LZ4 compression. The hardware compression accelerator includes eight LZ4 compression cores. Total test log size was 256 Kbytes and the compression time was 520.965 μ s, where we measured the time between the start flags and completion flags. Total test log is divided into eight 32 KB logs and compressed by eight LZ4 compression cores. Compression throughput is measured through total test log size divided by compression time. The throughput of the compression accelerator was founded to be up to 4.0 Gbit/s. In terms of each LZ4 compression core, the compression throughput recorded of about 500 Mbit/s. Also, the compression ratio is measured up to 2.69. The compression ratio of hardware compression accelerator depends on the Parallelization Window Size (PWS). As the PWS emerges, the compression ratio is improved. However, the scalability of hardware becomes lower. For improving the compression throughput with many-core architecture, the scalability of compression IP is one of the essential element. The compression accelerator was fabricated using Samsung 65 nm CMOS technology and verified the functionality of LZ4 compression.

sion along with an FPGA. Fig. 5 shows our chip layout of LZ4 compression. The Synopsys tool chain provides some information such as critical path, area, and power consumption for the LZ4 compression. Our LZ4 compression operates up to 75 MHz, and has 392 K gate counts. Total power consumption of our compression chip is 75 mW.

Fig. 5. Layout of LZ4 compression core

5 Conclusions

In this letter, we presented a detailed design and implementation of the LZ4 compression accelerator, which supports up to 4 Gbit/s of throughput. The compression accelerator is designed in Verilog HDL and realized both on FPGA and in ASIC. We expect that accelerating compression will bring forth a new spectrum of novel usage models for data centers.

Acknowledgments

This work was supported by the IT R&D program of MSIP/IITP [R0101-15-0186, Development of open type Hadoop storage appliance to support more than 48 TB per single data node].