FPGA Implementation of GZIP Compression and Decompression for IDC Services

Jian Ouyang，Hong Luo, Zilong Wang，Jiazi Tian, Chenghui Liu and Kehua Sheng
System Group, Baidu Inc.
ouyangjian@baidu.com, tianjiazi@baidu.com, liuchenghui@baidu.com, shengkehua@baidu.com
Electronic Engineering Department, Tsinghua University
Beijing, China
hongluo@tsinghua.edu.cn, zilongwang@mails.tsinghua.edu.cn

Abstract—**In the large scale data processing of Internet, data compression and decompression is a very important technology which can significantly improve the valid capacity of the disk and the valid bandwidth of IO, which can reduce the costs of IDC and accelerate application programs.**

This paper describes a low-cost FPGA hardware architecture of GZIP compression and decompression, which has been applied to IDC services successfully. Depending on different applications, the disk IO utilization has been improved by 300 to 500 percent, and the programs are accelerated by 30% to 200%, while 1 to 3 CPU-core resources could be released.

Keywords- IDC; FPGA; GZIP;

I. INTRODUCTION

Data compression technology is widely used in the field of Internet, databases and storage [1-2]. GZIP is the most widely used compression algorithm, which has a good compression ratio and compression/decompression bandwidth. For example, both apache and IIS compress the web pages by GZIP to improve the access response [3].

In traditional applications, software-based solutions are widely used in data compression and decompression. However, a large amount of CPU and memory resource is consumed in the large-scale data compressing and decompressing. For example, on the platform deploying a 2.66GHz CPU, the compression bandwidth is 50MB/s, while the decompression bandwidth is 200MB/s. Therefore, the effective transaction capability is confined to the remaining CPU and memory resource.

This paper proposed a GZIP hardware accelerator, which can offload the CPU consumption, and significantly improve the capacity of system transaction per second. Our innovative hardware accelerator distinguishes itself from the software one in the following aspects:

1. The FPGA accelerator uses two separate RAM blocks to store and manage historical data, thus for concurrent operation, while software implementation just uses linked lists.

2. In software implementation, the data is stored and read sequentially. But in hardware implementation, we use 16 distributed and individually addressable RAMs to read data simultaneously.

3. The hardware architecture can handle 4 matching search simultaneously, while only one matching search can be launched by the software implementation at the same time.

The rest of this paper is organized as follows. In Section II, we first review GZIP algorithm and the software implementation. In Section III, FPGA is applied in GZIP compression and decompression. The computing results and comparisons are made between software and hardware implementation in Section IV. Section V shows the application examples of this FPGA accelerator, and Section VI concludes this paper.

II. REVIEW OF GZIP

GZIP compression is based on DEFLATE algorithm [4], which consists of two parts: Lz77 [5] and Huffman coding [6]. Lz77 algorithm compresses the raw data by removing the redundant parts. Huffman coding is to encode the results of Lz77.

Software implementation of Lz77 algorithm is to find the best match and thus eliminating redundant data string through the “sliding window”, and a typical window size is 32KB. Such a window should be traversed to find all occurrences of the matching, thus the processing speed is very slow. Reducing the size of the window (e.g. 16KB) could increase the speed, but the compression ratio would be degraded.

Huffman coding in GZIP compression is the standard one. Lz77’s output will be divided into two parts: matched length part, and matched distance part, and these two parts will be encoded separately.

III. FPGA IMPLEMENTATION

In IDC Data services, one machine handles multiple tasks simultaneously, thus compression and decompression should be task-level-parallelism. Therefore, we designed a hardware board with four FPGA chips. As shown in Figure 1, these four cyclone-III chips are connected to a bridge chip through PCI, and the bridge chip communicates with the host through PCI-e X4 interface. The photo of the FPGA board is shown as Figure 2. Each FPGA chip can be configured as a compression engine or a decompression engine.

978-1-4244-8982-4/10/$26.00 ©2010 IEEE

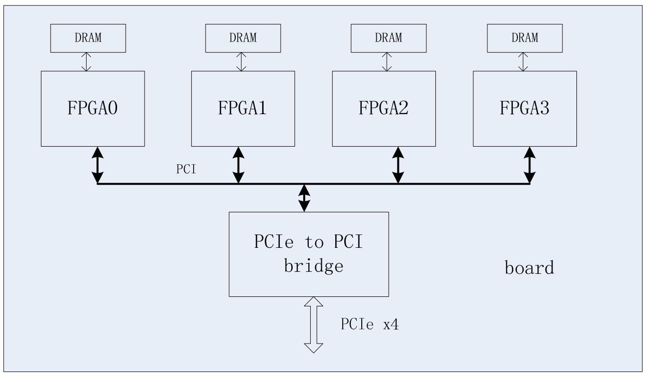

Figure 1 the architecture of FPGA board

Figure 2 the FPGA board

A. *FPGA-based GZIP compression*

According to Section II, in GZIP compression algorithm, the LZ77 encoding is the most critical part for both compression ratio and performance. Thus, our FPGA-based implementation focuses on efficient LZ77 matching search.

There are several difficulties for simultaneously handling multiple matching searches on low-cost FPGA such as ALTERA cyclone-III 80 series [7]:

1. The characters sharing the same hash value must be taken out within a cycle, so a new storage structure should be applied instead of the linked list.

2. Because of multiple read requests to access the historical data window in one cycle, the new storage structure should avoid the conflicts of read ports.

This paper presents an innovative architecture to solve the problems, which can achieve two-time higher compression bandwidth than the software implementation, with a neglect loss of compression ratio.

1. Our hardware architecture uses two separate RAM blocks to store and manage literal index in historical data window instead of linked lists in software.

2. In software implementation, the data is stored in sequence In hardware implementation, the historical data is stored in 16 distributed and individually addressable RAMs, and the data is stored and managed as “interleave”, which reduce the read conflicts caused by data locality.

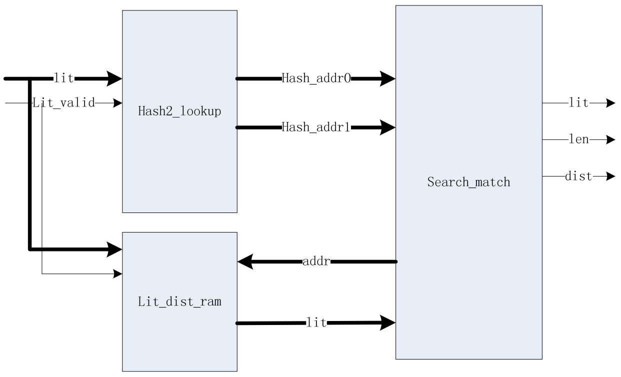

Figure 3: framework overview

3. Only one matching search can be issued by the software implementation at the same time. The hardware architecture can implement 4 matching search simultaneously, which can significantly speed up the searching match processing.

As shown in Figure 3, lz77 hardware architecture is divided into the following three modules:

1. Hash2_lookup: This module will calculate the hash values of each literal, store and manage of the literals’ index based on the hash value. In order to improve concurrency of matching search, two literal indexes can be read each cycle, as shown in Figure 4.

2. Lit_dist_ram: The distributed RAM module has been used to store historical data in order to improve the concurrency of read access. To realize full concurrency of matching search, the hardware architecture described in this paper made the following optimization to store historical data:

a. Each RAM can be individually addressed, and has two independent read ports., In our architecture, the 16KB historical data window uses 16 separate RAMs, thus it can support 32 concurrent read accesses in each cycle. Distributed storage structure is shown in Figure 5.

b. Historical data is stored in interleave way. The experiment has showed the locality of the data, so most of the read accesses will be centralized in the recent 1KB data block. In order to prevent from the resource conflicts, the interleave-way memory layout is applied to eliminate the data locality.

c. Data ports change from 1Byte into 4Byte, and each cycle 16-Byte of data will be read from the 4 RAM, which can reduce the number of read requests to the original 1/16.

3. Search_match: As shown in Figure 6, this module supports 4-way concurrent matching and searching in each cycle, and it supports the lazy matching to improve the compression ratio.

B. *FPGA-based GZIP decompression*

Huffman decoding in software implementation is based on multi-level look-up table [8], which has the advantage of less memory usage, but the parallelism of the algorithm is much limited. In this paper, we propose a new hardware architecture to explore more parallelism for GZIP decoding, and overall architecture is shown as Figure 7.

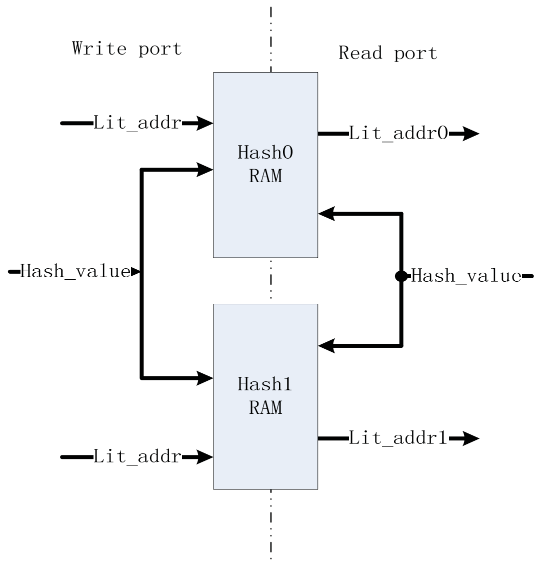

Figure 4: storage architecture of literal address

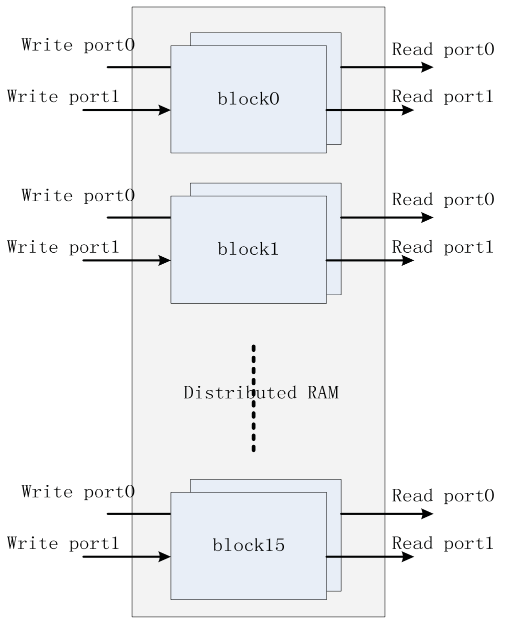

Figure 5: distributed storage architecture

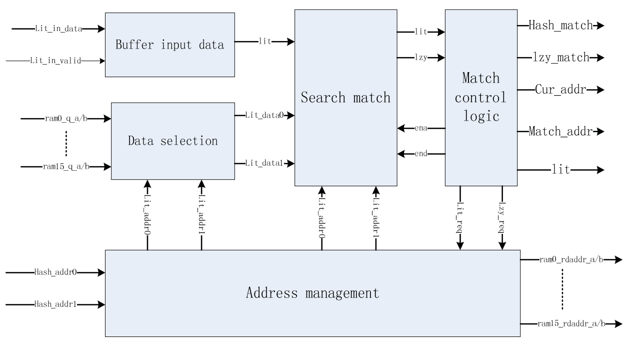

Figure 6: search match architecture

A "tables for each length" method to search 15 tables in parallel way is proposed for hardware implementation.

Firstly, we use highly efficient memory layout method to utilize ram resource as efficient as software implementation. The longest length of Huffman code in GZIP is 15. As it is standard Huffman coding [9], the code of the same length is continuous, which means that the symbol of the same code length can be stored in contiguous memory space. Huffman table needs an efficient index. This paper uses the memory layout to guarantees a minimum of storage resources and provides the fastest lookup. In order to quickly search, the initial address of each table needs to be saved in the register, and as there are 15 tables, 15 registers are needed. The Huffman table layout method is shown as Figure 8.

Secondly, this paper describes an efficient FPGA implementation of "tables for each length" [7] Huffman decoding algorithm, which achieves a full parallel Huffman decoding. This implementation concurrently looks up multiple tables (up to 15) in one cycle, calculates the code length of the current input encoded data, decodes and outputs symbols. The micro-architecture of Huffman decoding is shown as Figure 9.

In many applications, IO read requirement is much greater than IO write, it means that the decompression bandwidth must much higher than compression. In order to improve the decompression bandwidth and task-level concurrency, we also designed a multi-core decompression hardware architecture, which integrates several decompression engines into one low-cost FPGA chip. The 4 cores architecture is shown as Figure 10.

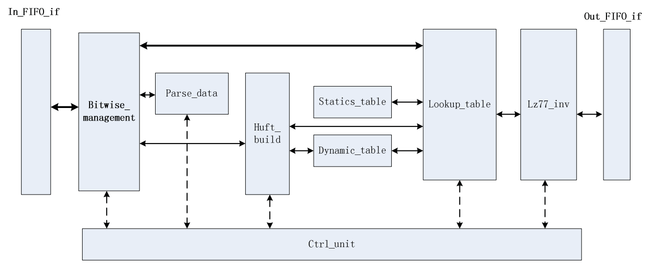

Figure 7: decompression architecture

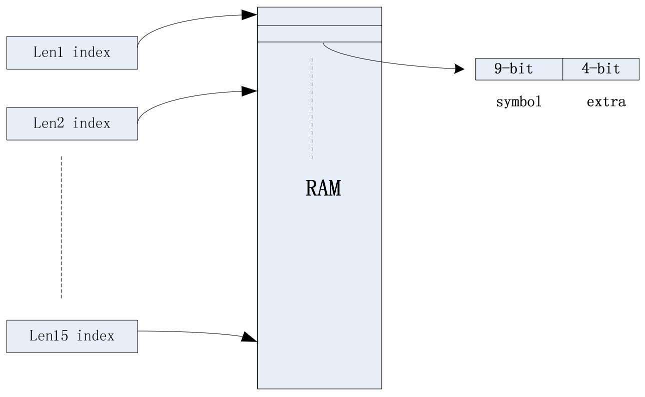

Figure 8: Huffman table memory layout

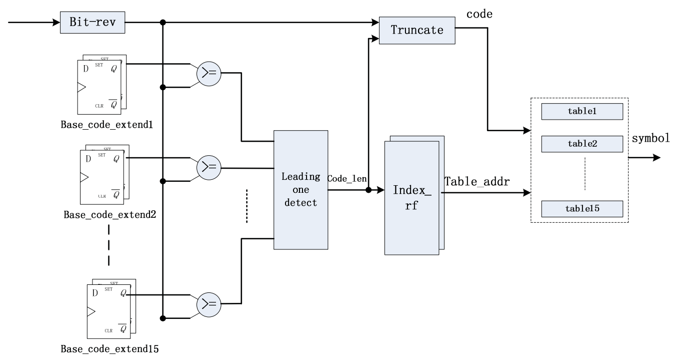

Figure 9: Micro-structure of "tables for each length"

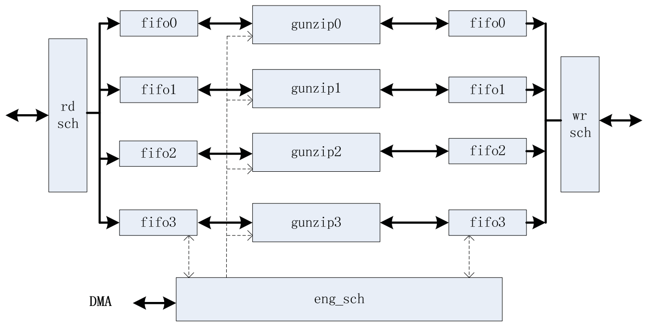

Figure 10: multi-core decompression

IV. Results and Comparisons

In this section, some data files including text and web pages are used for evaluating our FPGA accelerator. In our experiments, the single FPGA core runs at 132MHz, while the compared CPU core is 2.66GHz.

Table 1 shows the comparisons between CPU compression and FPGA compression. The results in the same row represent the same data. The FPGA accelerator sacrifices a little compression ratio in order to achieve much higher compression bandwidth. Meanwhile, FPGA can achieve a compression bandwidth up to 110MB/s. The adoption of FPGA accelerator leads to on average 270% improvement of compression bandwidth, while the maximum improvement is about 550%.

Table 2 shows the decompression bandwidth comparisons between FPGA decompression engine and CPU engine in different compression ratio. The FPGA can achieve up to 300MB/s bandwidth, and the maximum bandwidth gain is about 58%, while the average improvement is 38%.

Table 1: software and hardware compression comparisons

CPU Compression Ratio (%)	CPU Bandwidth (MB/s)	FPGA Compression Ratio (%)	FPGA Bandwidth (MB/s)
1.58	84.695	2.37	110.704
4.90	58.182	5.04	110.778
11.20	36.635	14.52	82.488
25.11	29.193	27.10	85.543
31.38	10.304	34.74	62.748
35.43	26.573	35.06	62.531
36.17	9.643	38.76	62.523
72.47	9.462	81.54	61.291

Table 2: software and hardware decompression comparisons

Compression Ratio (%)	CPU Bandwidth (MB/s)	FPGA Bandwidth (MB/s)
21.17	241.1	306.1
23.67	218.3	300.0
27.94	213.8	286.8
34.48	164.3	254.8
34.67	160.7	254.5
35.22	224.7	266.3

V. Application Examples

The hardware compression and decompression boards are widely deployed in IDC. There are two kinds of practical applications.

In Apache Web front end, we use hardware compression to replace the software compression. The average response time is decreased by 21%, and every 100 requests can save 7% of the CPU consumption. The ultimate capability of signal node's transaction achieves one time improvement, and the network bandwidth decreases to 30%.

In distributed large-scale data processing, with the use of hardware compression and decompression, disk capacity utilization is increased by three times without extra burden on CPU, and the running speed of application layer is increased by 30% to 40%.

VI. Conclusion

FPGA accelerator of GZIP compression and decompression could improve the disk and network utilization without extra burden on CPU, so many applications can be benefited from this FPGA accelerating solution. Our architecture is optimized for low-cost FPGA, which is also efficient and can reduce the budget of IDC significantly. At the same time, compared to general CPU, FPGA has much lower power, which effectively improves the energy utilization, and provides a feasible scheme for Low Carbon Computing.

Acknowledgment

The authors would like to thank Xin Zhang, Fei Wang and Huijun Tang for their important contributions for this work.

References

[1] Jeff Dean, "Challenges in building large-scale information retrieval system", WSDM09.
[2] Ghemawat, Gobioff, and Leung, "Google file system", SOSP 2003.
[3] http://httpd.apache.org/.
[4] RFC 1952 - GZIP file format specification version 4.3.
[5] Jacob Ziv and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, 23(3), pp. 337–343, May 1977.
[6] Michael Schindler, http://www.compressconsult.com/huffman/.
[7] http://www.altera.com/products/devices/cyclone3/cy3-index.jsp.
[8] Jean-loup Gailly and Mark Adler, http://www.gzip.org/.
[9] David Salomon, "Data Compression: The Complete Reference", 3rd edition.