IEICE TRANS. INF. & SYST., VOL.E101–D, NO.12 DECEMBER 2018

LETTER *Special Section on Parallel and Distributed Computing and Networking*

Hardware Based Parallel Phrase Matching Engine in Dictionary Compressor

Qian DONG[†a)], ***Student Member***

SUMMARY A parallel phrase matching (PM) engine for dictionary compression is presented. Hardware based parallel chaining hash can eliminate erroneous PM results raised by hash collision; while newly-designed storage architecture holding PM results solved the data dependency issue; Thus, the average compression speed is increased by 53%.

key words: *parallel chaining hash, dictionary compression, phrase match, hash collision, hardware algorithms*

1. Introduction

In order to cope with the explosion of data along with the popularity of cloud computing, a number of real time dictionary compression scheme are proposed by exploiting the parallel computing ability of hardware structure [1]–[10]. The central part of the dictionary compression consists of two serial steps: Phrase matching (PM) step positions the phrases in the dictionary that matched to the data to be compressed (raw data); replacement (RP) step analyses PM results and chooses the longest phrase to encode raw data. The dictionary, also known as the sliding window, changes during compression reflecting newly encoded raw data.

The most difficult issue of dictionary compression is an efficient search for duplicated phrases in PM step, both FPGA and ASIC based PM acceleration engines were proposed. Parallel matching algorithms based on content addressable memories (CAMs) and systolic arrays were proposed for optimal PM speeds [3]–[6], parallel dictionaries approaches based on generic hardware resources (distributed registers and on-chip memories) were also proposed and enhanced compression speeds to some extent [7]–[10]. These PM engines are resource intensive with trade-offs between compression speeds and hardware resources.

We propose a resource-efficient PM engine in this letter. The accuracy of PM results was advanced efficiently by using parallel chaining hash method; while a FIFO structure holding PM results was introduced for implementations that support the full parallel operation of RP and PM step. Experimental data shows that our parallel PM engine accelerates average compression speed by 53% compared with the traditional approaches based on a single hash function [1], [2].

Manuscript received December 6, 2017.
Manuscript publicized September 18, 2018.
[†]The author is with School of Integrated Circuits, Southeast University, Nanjing, 210046, P.R. China.
a) E-mail: ic_qiand@seu.edu.cn
DOI: 10.1587/transinf.2018PAL0001

2. Parallel Chaining Hash

Hash function is often used to simplify the PM step. The phrases in dictionary are arranged in linked lists indexed by the hash values of their prefix (first-3-byte); thus, according to raw data's hash value, a linked list of the duplicated phrases can be selected for RP step. The selected linked list also needs to be maintained reflecting newly encoded raw data.

A hash function maps M entries to N buckets [11]. In dictionary compression, N is defined smaller than M to save memory resource, but it induces a negative effect of erroneous PM results. For instance, the hash function in LZ77 can be denoted as Eq. (1) or (2), M and N are 24 and 15 in bit width. It is obvious that the hash value h in Eq. (2) does not change with the high 3-bit input: $P[23:21]$ Based on this, the phases that do not really match each other could be assigned to a same linked list according to the same hash value, resulting in erroneous PM results, which is commonly known as wrong matches induced by hash collision.

$$h[14:0] = H(P[23:0]) \tag{1}$$

$$h = ((P[23:16] << 10) \oplus (P[15:8] << 5) \oplus (P[7:0]))\&15'b1 \tag{2}$$

The parallel chaining hash method is proposed to remove the wrong matches for accelerating the data compression, which is based on an assumption that the phrase can be reconstructed from its two hash values. For instance, H_a and H_b, which outputs all the odd and even bits of the input data respectively, is such a pair of hash functions. Thus, the sufficient and necessary condition of $p_i = p_j$ is shown in Eq. (3).

$$\begin{cases} H_a(p_i) = H_a(p_j) \\ H_b(p_i) = H_b(p_j) \end{cases} \tag{3}$$

Under the guidance of such pair of hash functions, each phrase in dictionary is parallel assigned into two groups of linked lists. In a PM loop, one linked list can be selected from each group according to raw data's hash value, and a coincidence of an element in the two selected linked lists suggests that the corresponding phrase matches with the raw data.

A didactic example is shown in Fig. 1: p_a is the prefix of raw data at 201; linked list A is maintained based on the hash function H_a, while linked list B is according to H_b.

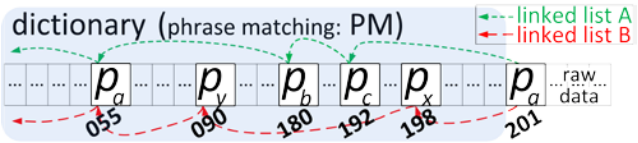

Fig. 1 Two selected linked lists in parallel chaining hash method.

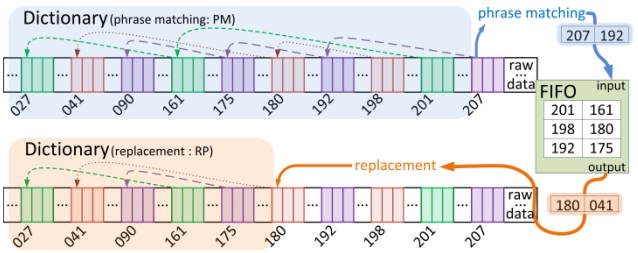

Fig. 2 The FIFO memory used for storing PM results.

Suppose that because of hash collisions, $H_a(p_a) = H_a(p_b) = H_a(p_c)$ and $H_b(p_a) = H_b(p_x) = H_b(p_y)$. In the PM loop for the raw data at 201, the same element 055 can be queried by traversing linked list A and B, although wrong matches exist in each selected linked lists. As a result, p_a at 055 is found as the correct matched phase.

3. Storage Mode of Phrase Matching Results for Parallel Compression

When the matched phrase is found, a maintained linked list that consists of the phrases' direct addresses will be exported to support RP step. Typically, PM and RP steps can only run serially and alternately. Any PM step ahead of schedule is likely to overwrite useful information required by RP step, and this data dependency limits the full parallelization of data compression.

Careful observation of PM results in the linked list format reveals certain degrees of freedom, which can be exploited to achieve fine-grain parallelism in dictionary compression. These observations lead to the following key idea: the newly generated PM result is inserted in the head of the old linked list, without changing the pre-existing linking relationships; it's like the new nodes grow out from the old linked lists' heads.

In our design, a FIFO memory is added to save the head of the linked list that is being maintained, and guarantee RP step to index an appropriate fragment in the maintained linked lists. As shown in Fig. 2, even if PM step has processed raw data up to 207, at the same time, RP step for raw data at 180 still can find the accurate linked list indexed by the output of the FIFO. In this way, PM and RP steps are fully parallel and no longer restrict each other.

4. Hardware Architecture of Phrase Matching Engine

A pipelined architecture of dictionary compressor is shown in Fig. 3, which includes PM engine and RP block. Raw data

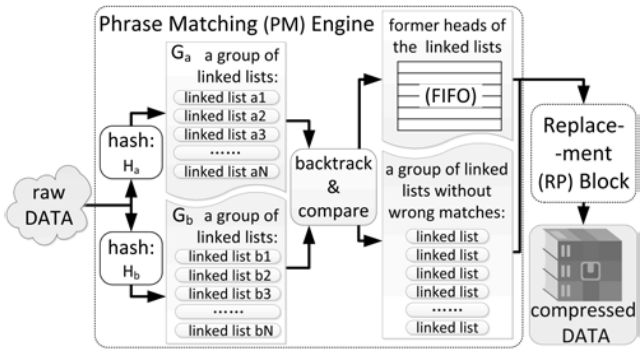

Fig. 3 Diagram of hardware based dictionary compressor.

is processed by two hash modules of PM engine simultaneously and independently, which employ two hash functions in Eq. (4). The two groups of the linked lists (G_a and G_b) generated by H_a and H_b are processed by the backtrack & compare module, producing a group of linked lists without wrong matches as the PM result, which are indexed by the heads of linked lists storing in the FIFO. Based on this, RP block pipelined selects the longest match, encodes raw data into compressed data.

$$\begin{cases} H_a: \ h = ((P[23:16] << 10) \oplus (P[15:8] << 5) \\ \qquad \oplus (P[7:0])) \,\&\, 15'b1 \\ H_b: \ h = \begin{pmatrix} (P[23:16] * 0.618 << 10) \\ \oplus (P[15:8] * 0.618 << 5) \\ \oplus (P[7:0] * 0.618) \end{pmatrix} \&\, 15'b1 \end{cases} \tag{4}$$

5. Experimental Results

The evaluation of our parallel PM engine leads to the resource utilizations for Xilinx Virtex-6 FPGA devices (XC6vlx240tffU56-l): 917 (0.30%) registers, 1681 (1.12%) LUTs, and 120 (28.84%) block RAMs. Dictionary compression (LZ77) tests were performed using Canterbury Corpus, the performance measures are wrong match ratio (WMR), compression ratio (CR), and compression speed (CS), which are expressed as Eqs. (5), (6) and (7).

$$\text{WMR} = \frac{\textit{number of wrong matches}}{\textit{number of all matches}} \times 100\% \tag{5}$$

$$\text{CR} = \frac{\textit{compressed data size}}{\textit{raw data size}} \times 100\% \tag{6}$$

$$\text{CS} = \frac{\textit{raw data size}}{\textit{total time consumed in compression}} \tag{7}$$

For illustration, Table 1 summaries obtained results with our parallel PM engine compared to those using traditional PM methods with only a hash function in [1], [2]. The average WMR sharply decreased from 16.39% to 0.10%, when the traditional PM methods changed to parallel PM engine. WMR reaches 0.10% indicates that almost all the wrong matches were eliminated. At the same time, a very

Table 1 Comparative results in terms of wrong match ratio (WR), compression ratio (CR), and compression speed (CS).

Test Source File Name	With proposed **parallel** phrase matching engine			With **traditional** [1,2] phrase matching methods		
	WMR (%)	CR (%)	CS (MBps)	WMR (%)	CR (%)	CS (MBps)
alice29.txt	0.08	52.33	86.57	6.85	52.25	58.96
asyoulik.txt	0.09	56.60	84.01	6.18	56.53	57.04
cp.html	0.00	42.30	130.48	19.24	42.63	83.59
fields.c	0.00	36.81	119.43	5.24	37.31	82.19
grammar.lsp	0.00	42.14	143.52	7.09	43.43	95.77
kennedy.xls	0.04	28.09	118.75	41.65	32.45	63.27
lcet10.txt	0.18	49.47	88.07	7.48	49.38	60.10
plrabn12.txt	0.13	60.67	77.39	6.77	60.47	52.54
ptt5	0.54	14.68	136.5	39.76	14.67	94.76
sum	0.00	42.59	108.13	35.43	42.64	61.65
xargs.1	0.00	52.80	133.17	4.56	54.32	90.76
average	**0.10**	**43.50**	**111.45**	**16.39**	**44.19**	**72.78**

few number of wrong matches infiltrated the PM result; we believe the reason is that there is still a slight correlation between the hash function in Eq. (4). The more accurate PM results and the advantages of parallel computing can accelerate dictionary compression. Table 1 also shows the CS of the dictionary compressor using the two PM engines under the same test condition. The average improvement of CS using our parallel PM engine is 53.12%. In the previous studies of accelerating data compression, the great improvement of CS was usually at the expense of CR [5], [6]. In our study, the situation is different though; a slightly better CR is also obtained, which is shown in the 3th and 6th columns of the Table 1.

6. Conclusion

We present a hardware based PM engine for dictionary compression, which uses two hash functions to guide the maintenance of two groups of linked lists, and then eliminates the wrong match by recursively comparing the linked lists. Moreover, the storage mode of PM results enables the full parallel operation of PM and RP steps. Coupled with the advantages of pipeline structure, the PM results with few numbers of wrong matches is extremely advantageous to improve CS in the hardware implementations.

Acknowledgments

This work is supported by the national natural science foundation of China (No. 61571116).

References

[1] B. Li, L. Zhang, Z. Shang, and Q. Dong, "Implementation of LZMA compression algorithm on FPGA," Electron. Lett., vol.50, no.21, pp.1522–1524, 2014.
[2] S. Rigler, W. Bishop, and A. Kennings, "FPGA-based lossless data compression using Huffman and LZ77 algorithms," IEEE Canadian Conference on Electrical and Computer Engineering (ICCECE), pp.1235–1238, 2007.
[3] D.-H. Le, K. Inoue, and C.-K. Pham, "Design of a parallel CAM-based multi-match search system using 0.18-μm CMOS process," IEEE Fifth International Conference on Communications and Electronics (ICCE), pp.336–339, 2014.
[4] P. Manikandan, B.B. Larsen, and E.J. Aas, "Design of embedded TCAM based longest prefix match search engine," Microprocessors and Microsystems, vol.35, no.8, pp.659–667, 2011.
[5] M.A.A.E. Ghany, A.E. Salama, and H.A. Khalil, "Design and implementation of FPGA-based systolic array for LZ data compression," IEEE International Symposium on Circuits and Systems (ISCAS), pp.3691–3695, 2007.
[6] S.-A. Hwang and C.-W. Wu, "Unified VLSI systolic array design for LZ data compression," IEEE Trans. Very Large Scale Integr. Syst. (VLSI), vol.9, no.4, pp.489–499, 2001.
[7] S.M. Lee, J.H. Jang, J.H. Oh, J.K. Kim, and S.E. Lee, "Design of hardware accelerator for Lempel-Ziv 4 (LZ4) compression," IEICE Electronics Express, vol.14, no.11, 20170399, 2017.
[8] J. Fowers, J.-Y. Kim, D. Burger, and S. Hauck, "A scalable high-bandwidth architecture for lossless compression on FPGAs," IEEE International Symposium on Field-Programmable Custom Computing Machine (FCCM), pp.52–59, 2015.
[9] M.S. Abdelfattah, A. Hagiescu, and D. Singh, "Gzip on a chip: High performance lossless data compression on FPGAs using OpenCL," International Workshop on OpenCL 2013 & 2014 (IWOCL'14), no.4, 2014.
[10] K. Liao, M. Petri, A. Moffat, and A. Wirth, "Effective construction of relative Lempel-Ziv dictionaries," International Conference on World Wide Web, WWW'16, pp.807–816, 2016.
[11] Z. Shi, C. Ma, J. Cote, and B. Wang, "Hardware implementation of hash functions," Introduction to Hardware Security and Trust, pp.27–50, Springer New York, 2012.