An FPGA approach for fast bitmap indexing

Xuan-Thuan Nguyen^{a)}, Hong-Thu Nguyen, and Cong-Kha Pham

The University of Electro-Communications,

1–5–1 Chofugaoka, Chofu, Tokyo 182–8585, Japan

a) xuanthuan@vlsilab.ee.uec.ac.jp

Abstract: In this paper, an efficient architecture of an FPGA-based bitmap index creation is proposed. The design utilizes a content addressable memory together with a bit-level transpose matrix to index multi-record documents by several given keywords. The experiments in a Cyclone V SX FPGA proved that our circuit could attain throughput of 330.6 million records per second while only using around 71% of embedded memory together with 45% of lookup tables and registers. In fact, achieved throughput is 2.8 times and 1.7 times as high as that of CPU-based and GPU-based design, respectively.

Keywords: bitmap index, CAM, transpose matrix, FPGA

Classification: Electron devices, circuits, and systems

References

- [1] A. Silberschatz, H. F. Korth and S. Sudarshan: *Database System Concepts* (2011) 6th ed. Chap. 11. 475.
- [2] R. Bayer: *Acta Inform.* **1** (1972) 290. DOI:10.1007/BF00289509
- [3] W. Litwin: *Int. Conf. Very Large Databases* (1980) 213.
- [4] P. O’Neil and D. Quass: *Proc. ACM SIGMOD Conf. Management of Data* (1997). DOI:10.1145/253260.253268
- [5] R. R. Sinha and S. Mitra: *Int. Parallel and Distributed Processing Symp.* (2006).
- [6] K. Stockinger and K. Wu: *IRM Press*, Chapter 7 (2007) 157.
- [7] E. O’Neil, P. O’Neil and K. Wu: *Int. Database Engineering and Applications Symp.* (2007) 72. DOI:10.1109/IDEAS.2007.19
- [8] F. Fusco, M. Vlachos, X. Dimitropoulos and L. Deri: *Conf. Internet Measurement Conference (IMC)* (2013) 327.
- [9] T. Zhong, K. A. Doshi and G. Deng: *Int. Conf. Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)* (2014) 233. DOI:10.1109/CyberC.2014.49
- [10] D. Heinrich, S. Werner, M. Stelzner, C. Blochwitz, T. Pionteck and S. Groppe: *Int. Symp. Reconfigurable Communication-centric Systems-on-Chip* (2015) 1.
- [11] R. Mueller and J. Teubner: *Int. Conf. Extending Database Technology* (2010) 721. DOI:10.1145/1739041.1739137
- [12] D.-H. Le, K. Inoue and C.-K. Pham: *IEICE Trans. Electron.* **E97-C** (2014) 65. DOI:10.1587/transele.E97.C.65
- [13] Altera: *Cyclone V SoC Development Kit - User Guide* (2015) 1.
- [14] K. Wu, E. J. Otoo and A. Shoshani: *ACM Trans. Database Syst. (TODS)* **31** [1] (2006) 1. DOI:10.1145/1132863.1132864
- [15] F. Deliege and T. B. Pedersen: *Int. Conf. Extending Database Technology*

- (2010) 228. DOI:10.1145/1739041.1739071
- [16] X.-T. Nguyen, H.-T. Nguyen and C.-K. Pham: IEEE Int. Symp. Circuits and Systems (ISCAS) (2015) 2908. DOI:10.1109/ISCAS.2015.7169295

1 Introduction

With the rapid growth of database systems nowadays, querying inside information has become increasingly difficult. Linear search is no longer suitable for the real-time requirement. To accelerate the query tasks, data indexing [1] must be performed in advance. With the created index values, the processing unit only looks information up in a narrow range of data sets, thereby reducing the search time significantly.

Several indexing methods have been introduced in the last decade such as B-tree index in 1972 [2], hash in 1980 [3], and bitmap index in 1997 [4]. Among them, bitmap index has gradually attained a great potential for scientific database [5] and data warehouse (DW) applications [6], where data are not frequently updated, even in the context of many concurrent systems. Moreover, a bitmap index is especially effective against multiple-key queries and typically requires a much smaller size than the database itself [7].

Many approaches for both software and hardware-based indexing have been proposed recently. For example, by exploiting the parallelism of a general processing unit (GPU), F. Fusco et al. [8] achieved an index throughput of up to 185 million records per second (Mrps). Besides, T. Zhong et al. [9] presented a multi-core-processor-based design that could index up to 116 Mrps by improving the cache line conflicts among processors (CPUs). Additionally, with the help of a hardware accelerator, D. Herinrich et al. [10] proved that computation time of a semantic web database system could be greatly reduced, i.e. up to 50% in comparison with software-only implementation.

It appears certain that FPGA has become a good fit for data-intensive database systems due to the new advancement of silicon technology so far [11]. For this reason, in this paper, we propose an FPGA-based architecture for fast bitmap indexing. Our design includes a content addressable memory (CAM) whose advantages were fully stated in previous research [12], together with a bit-level transpose matrix. Moreover, the output indexes allow the processing units behind it to answer users' query in parallel using basic logic circuits. The performance analysis proved that our design can feasibly index up to 330.6 Mrps with the clock rate of 150 MHz in a low-cost low-power Cyclone V FPGA [13].

The remainder of this paper is organized as follows. Section 2 briefly presents the motivation for this study. Section 3 describes in detail the CAM and transpose matrix architecture. Section 4 then compares the proposed design with the others regarding throughput and resource utilization. Section 5 finally gives the conclusion and future works.

Fig. 1. The basic bitmap index (a) and its transposition (b).

2 Motivation

2.1 Background of bitmap index

A basic bitmap index (BI) is simply a bit matrix used to index values in a set of data. Suppose that we have a list of values $DATA = (5, 4, 1, 0, 1, 3, 2, 1)$ numbered sequentially from zero to seven, as shown in Fig. 1. BI has six columns because DATA includes integer values ranging from zero to five. Additionally, the number of BI rows are equal to the number of values. The size of BI thus is 8×6 bits. The bit at row i and column j , i.e. $BI(i, j)$, is set to one if $DATA(i) = j$. All remaining bits, by contrast, are set to zero. BI has the advantage of answering multi-condition queries efficiently. For instance, to find all positions i , where $DATA(i) < 2$, we firstly perform bitwise *OR* operation between the zeroth and first column, i.e. 00010000 and 00101001, respectively. Lastly, we look for all high bits in the result 00111001. The second, third, fourth, and seventh bits of this result mean that $DATA(2)$, $DATA(3)$, $DATA(4)$, and $DATA(7)$ satisfy the condition above.

As can be seen in Fig. 1(a), to obtain a column k , we have to extract the correspondent bit k inside all rows of BI. Assume that in hardware, the rate of read process is one row per clock, eight clocks are needed to get a particular column. To reduce this processing time, we transpose BI as soon as all values are completely indexed. In other words, all the rows of BI are turned into columns and vice-versa, as illustrated in Fig. 1(b). With the BI transpose (BIT), only one clock cycle is necessary for obtaining the column k .

2.2 Overview of FPGA-based bitmap index

Based on the idea of BI, we proposed an FPGA-based bitmap indexing that aims to index N records stored in the data repository by M given keys. The records and keys are numbered from R_0 to R_{N-1} and from K_0 to K_{M-1} , respectively. The result BIT indicates the existence of keys in a certain record. The design consists of three main modules: content addressable memory (CAM), bitmap index buffer (BI), and transpose matrix (TM), as depicted in Fig. 2(a).

Fig. 2. The general block diagram of FPGA-based bitmap indexing (a) and its flow chart (b).

Fig. 2(b) shows the flow chart of hardware operation. Initially, the zeroth record R_0 is loaded to CAM. Upon completing, the CAM utilizes all keys to generate the corresponding index. Those indexes then are temporarily kept in the first row of BI. This workflow is repeated until all records are properly processed. Afterwards, BI is converted into BIT by TM module. This BIT can be compressed to save the storage space or be used to answer users' queries.

Some software implementation for high-throughput indexing has been studied recently. All designs exploited the advantages of either CPU or GPU, i.e. high frequency and multitasking capability, to significantly improve the execution time. In this work, however, we prove that an FPGA-based design can index many more records than other platforms despite its low-frequency system clock. The proposed architecture is described in detail in the next section.

3 Implementation

3.1 Content Addressable Memory

Content Addressable Memory (CAM) is a special type of memory that receives the input search data and returns the address of matching data inside its storage. Due to

Fig. 3. The hardware architecture of CAM unit (a) and CAM array (b).

this distinct feature, CAM is employed in a wide variety of high-speed searching applications such as image processing and information retrieval. Although there were many approaches to CAM research on FPGAs, dual-port-memory-based (DPM-based) CAM, in our best knowledge, is one of the fastest searching engines so far [12]. The architecture of a 32-word 8-bit DPM-based CAM unit is shown in Fig. 3(a). Port A consists of 8,192 addresses and each address stores 1-bit data. Port B, on the other hand, is composed of 256 addresses and each address contains 32-bit data. The memory size of port A and B, therefore, are $8,192 \times 1 \text{ bit} = 256 \times 32 \text{ bits} = 8,192 \text{ bits}$.

Initially, all 8-bit *data* and 5-bit *addr* are loaded into port A of CAM by *set* and *write_en* signal. After all *data* are loaded into CAM, *search_data* is sent to CAM in turn, together with the assertion of *match_en*. The 32-bit result is ready at *match_addr* upon the next clock cycle. In fact, the 1-valued bits of *match_addr* indicate the addresses *i* of matching data. For example, if *data* = 1 at *addr* = (2, 4, 7), the *match_addr* at *search_data* = 1 will be 0x00000094. Moreover, depending on the type of application, CAM size is expanded by connecting many CAM units, as illustrated in Fig. 3(b). The M10K embedded memory blocks are deployed for this CAM architecture to avoid consuming much of the combinational logics and registers.

In our preliminary analysis, the execution time t_{CAM} of each record, in clock cycle, is the sum of t_{CAM_1} - time to copy every record to CAM, and t_{CAM_2} - time to index that record by *M* given keys, as shown in Eq. (1). Because t_{CAM_1} replies on the method of inputting data, it will be analyzed later in Section 4.

$$t_{CAM} = N \times (t_{CAM_1} + t_{CAM_2}) = N \times (t_{CAM_1} + M) \text{ (clocks)} \quad (1)$$

3.2 Transpose matrix

The transpose matrix (TM) turns all the rows of BI into columns of BIT. By exploiting this BIT, the hardware accelerator of a database can presumably reply to all queries at very low latency. Fig. 4(a) depicts the three-step methodology of TM. To begin with, each $B \times B$ -bit block is captured in sequence from left to right. Subsequently, every B -bit column is transposed to the corresponding B -bit row, e.g. column *i* of BK_0 will become row *i* of BK_0^T , and so on. Upon completing, all new blocks are saved to BI in order, from top to bottom.

Fig. 4. The methodology (a) and hardware architecture (b) of bit-level transpose matrix.

The hardware architecture is composed of two main modules, CONTROL UNIT and TRANSPOSE UNIT, as shown in Fig. 4(b). The first module manages to read input from BI and write output to BIT in a predetermined order, whereas the second module focuses on the matrix transposition. After initialization, FINITE STATE MACHINE (FSM) translates the data of COUNTER ROW and COUNTER COL into the right read address - rd_addr , and read request - rd_req . BI then returns answer rd_q to all internal registers inside TRANSPOSE UNIT. Additionally, if reg_sel is equal to i , ENCODER allows corresponding register REG_i to update new rd_q promptly. As soon as all registers are filled up, FSM sends bit_sel to all multiplexers so as to select a certain bit of all registers, i.e. $bitmap_index_data$ or wr_data , in parallel. Simultaneously, FSM generates the correspondent write address wr_addr and write request wr_req for BIT.

Assume that L is the number of $B \times B$ -bit blocks in $M \times N$ -bit BI and t_{TM} is the total processing time in clock cycles. In our analysis, the execution time of a certain block consists of t_{TM_1} - the time to copy block data from BI to registers, t_{TM_2} - the time to dispatch all captured indexes to BIT, and t_{TM_3} - two additional clocks for initialize FSM, as shown in Eq. (2). Because of TM architecture, t_{TM_1} and t_{TM_2} each can be fully executed within B clocks.

$$t_{TM} = L \times (t_{TM_1} + t_{TM_2} + t_{TM_3}) = \frac{M \times N}{B \times B} \times (B + B + 2) \text{ (clocks)} \quad (2)$$

4 Performance analysis

The performance of our FPGA-based bitmap indexing was evaluated by its throughput and resources in comparison with previous works [8, 9], whose details

Table I. The bitmap indexing method, hardware platform, and throughput of previous works.

	Method	Hardware platform	THR
(A) [9]	WAH (Fastbit)	Intel Xeon E5-2680 2.7-GHz CPU with 64-GB DDR3 memory	6
(B) [8]	WAH	Intel i7-2600K 3.4-GHz CPU with 8-Mb cache	37
(C) [9]	ParaSAIL	Intel Xeon E5-2680 2.7-GHz CPU with 64-GB DDR3 memory	108
(D) [9]	ParaSAIL	Intel Xeon Phi 5110 accelerator card that contains 60-core 1-GHz CPU with 8-GB GDDR5 memory	116
(E) [8]	PLWAH	NVIDIA GTX-670 GPU	185

THR: Throughput (Mrps - million records per second)

are illustrated in Table I. In [8], the authors conducted two experiments (E) and (B) to prove the efficiency of GPU over the CPU. In [9], by avoiding cache conflict in many-core and multi-core CPU system, the authors stated that both experiments (C) and (D) achieved much-improved throughput than (A). Moreover, the description in both studies implied that the number of keys would be limited to 16. It is noted that WAH [14] and PLWAH [15] are two kinds of compressed bitmap indexing which can substantially reduce the index size. Because we only focus on the throughput in this study, the index size is not considered.

In order to draw a comparison, some parameters including the number of records in one document N , the number of keys M , and the size of block B were prepared in advance. Firstly, N was set at 256, or the proposed bitmap index creation (BIC) was responsible for 256 records each time, and every record contains 256 bytes. Several experiments on different N and record size showed that changing those numbers did not significantly improve the total throughput, whereas BIC had to consume much more of the resources. Secondly, M was selected as 16 so that the supported keys are similar to two referred works. Lastly, B was identical to M so as to reduce the transpose time. After being thoroughly simulated on Modelsim, the design was verified in a low-power Altera Cyclone V SX FPGA at a clock rate of 150 MHz.

Because the Altera DDR3 controller is capable of transferring 32-byte data from external memory to FPGA at 150-MHz clock, every 256-byte record can be loaded to CAM within eight clocks. In other words, t_{CAM_1} of Eq. (1) is seen as eight clock cycles. Besides, the processing time of BIC, t_{BI} , is the sum of t_{CAM} and t_{TM} , as prior described in Eq. (1) and Eq. (2), respectively. After replacing N , M , and B with three values mentioned above, t_{BI} becomes 6,688 clock cycles or approximately 44.59 μ s at 150-MHz frequency. This result also suggested that BIC could index approximately 5.7 million records per second (Mrps).

Moreover, the parallelism of FPGA is perfectly suited to deploy a large number of BICs in parallel, thereby increasing the total throughput substantially. Fig. 5 illustrates the architecture of a multi-BIC system (MBIC), in which every BIC takes

Fig. 5. The general block diagram of multi-BIC design.

Table II. The resource utilization and throughput of multi-BIC design.

	LUTs	Registers	Embedded Memory (Kbits)	THR
MBIC_2	710	874	136 (2%)	11.4
MBIC_7	2,481	3,062	544 (9%)	39.9
MBIC_19	6,745	8,303	1,292 (23%)	108.3
MBIC_21	7,455	9,177	1,428 (25%)	119.7
MBIC_33	11,715	14,421	2,244 (40%)	188.1
MBIC_58	20,823	25,315	3,944 (71%)	330.6

THR: Throughput (Mrps - million records per second)

charge of one certain document (DOC). Each input and output port should connect to a multi-port memory controller such as [16] so that MBIC can fully operate in parallel. Suppose that MBIC_ i indicates an MBIC with i BICs that are integrated within. For instance, at $i = 2$, MBIC_2 means two BICs are used simultaneously. As a result, the throughput of an MBIC_ i is considered as the product of i and the throughput of one BIC.

Table II depicts the hardware consumption and throughput in case of 2, 7, 19, 21, 33, and 58 BICs usage. As can be seen in this table, every design can correspondingly provide higher throughput than (A), (B), (C), (D), and (E). For example, the throughput of MBIC_2 and MBIC_33 are higher than (A) - 6 Mrps, and (E) - 185 Mrps, respectively. Moreover, at the minimum setting, MBIC_2, only 2% of embedded memory and 1% of lookup tables (LUTs) and registers are utilized. At maximum setting, the throughput of MBIC_58 can reach 330.6 Mrps while using 71% of embedded memory and 45% of LUTs and registers. In other words, the throughput of MBIC_58 is 2.8 times and 1.7 times as high as that of design on CPU - (D) and GPU - (E), respectively. Besides, a clumping technique was presented in [9] to expand the ParaSAIL throughput up to 473 Mrps. However, this method was only appropriate to particular data sets, and thus is beyond the scope of this comparison.

5 Conclusion

In this paper, we proposed an FPGA-based bitmap indexing that delivered higher throughput and consumed less of the resources than other powerful platforms. A CAM and a bit-level transpose matrix are two main components that form the fast bitmap index creator. The experiment of $N = 256$, $M = 16$, and $B = 16$ at 150-MHz clock rate on a Cyclone V SX FPGA proved that every single BIC could process approximately 5.7 Mrps. Moreover, if several BICs were deployed in parallel, the total throughput would increase significantly, i.e. approximately 330.6 Mrps in case of 58 BICs usage. In fact, attained throughput was 2.8 times and 1.7 times as high as that of design on CPU and GPU, respectively. In future works, the MBIC will be integrated into a database system to improve the indexing tasks.