Abstract—Web search engines are now using tens of thousands of index servers that consume huge amount of power. In this paper, we investigate FPGAs as the implementation platform for power efficient index serving. We propose the architecture of an FPGA-based inverted index search engine, as well as implementations of essential components, including decoder, matcher and ranker. We successfully boot up the FPGA-based search engine and run experiments on real-world data from a commercial search engine. The targeted FPGA-based hardware index server could achieve up to 19.52X power efficiency and 7.17X price efficiency over an Intel Xeon server with highly optimized software. This is the first complete work using FPGAs to implement query processing for Web search engines.

Index Terms—index server, query processing, hardware accelerator, FPGA, performance;

I. INTRODUCTION

Web search based Internet service is a huge business. To attract more users and gain larger margins, search engine companies are continuously improving their search quality and performance. The core service in a search engine is the index serving, which includes crawler, indexer and query processor. Modern search engines use inverted files to index documents (web pages) for efficient document retrieval in a common structure in modern search engines [1][2][3][4]. User queries are sent to an aggregator, which dispatches the queries to different Index Servers (ISs) in the computer cluster to balance the workload. Each IS, operating in a distributed manner, serves a subset of inverted indices from an index built pipeline. An IS processes a dispatched query as followings:

i) Invoke Stream Readers to read the inverted index files that contain the query words;

ii) Invoke a Matcher to find the Web pages that satisfy the input query (the matching list);

iii) Invoke a Ranker to calculate scores for the matched documents according to their relevances to the input query (L1 ranking), and select the top k Web pages to recalculate the scores with a more complex function (L2 ranking). The ranking functions are usually the combination of many Information Retrieval (IR) document features, which includes query dependent features, called dynamic features, such as BM25 [5], the frequency of the term (noted as tf), etc. and query-independent features, called static features, such as languages, PageRank [1], etc..

Then the aggregator combines the results across all ISs to generate the final result, including a short description of returned Web pages for the users.

With the increasing of Internet size and users, concerns about the energy consumption of clusters have been grown. Many efforts were taken to reduce the energy footprint of servers, as well as the total cost of ownership (TCO). Some work brings more efficient platforms to the web search, such as mobile cores [4] and GPUs (Graphical Processing Units) [3]. [4] employs Atom processors to reduce the power consumption and improve the throughput. They claims 5X more efficiency on Atom than on the server-class Xeon per core. However, it is observed that Atoms weakens the performance of web search. [3] proposes a GPU-based framework for IR query processing, and reports 3X throughput improvement compared to a CPU. While it shows a quad-core CPU still provides better cost efficiency than a GPU.

Other than CPUs and GPUs, FPGAs combine the performance and power advantage of dedicated hardware with the flexibility of reconfigurable device, and provide a promising implementation method. To provide a more power-efficient platform, we customized an FPGA-based accelerator board and developed a simple search engine on it. The major contributions of our work include: 1) to the best of the authors’
knowledge, our work is the first complete FPGA-based implementation of query processing for a Web search engine, which includes matcher, inverted list compression/decompression and ranker; 2) We successfully boot up the system with real-world data from a commercial search engine; and 3) we propose a query processing system that integrates FPGA, multiple independent Flash channels and DRAMs. It has the potential to achieve 19.52X power efficiency and 7.17X price efficiency over a highly optimized software version.

II. SYSTEM DESIGN AND HARDWARE IMPLEMENTATION

A. Pinaka: Customized FPGA-based Web Search Platform

We design a customized FPGA-based PCI Express accelerator board called Pinaka [7] to efficiently implement the query processing system described above. The access pattern to Web index is write once, and randomly read for a relatively long period of time until the next index update. Thus, it is very suitable for Flash memories, since Flash memories process read much faster that write. Besides, Flash provides much higher bandwidth and lower latency than hard disks, and lower cost than DRAMs. So Pinaka is designed to have daughter boards with more than 30 flash chips (up to 192 GB), and 36 independent flash channels to gain up to 1.9 GB/s bandwidth. A Xilinx Virtex-5 LXT FPGA (LX30T) is used as PCI Express interface. Two Altera Cyclone-III (EP3C120) FPGAs are selected as main computation engines for good performance/cost ratio. Besides, Pinaka has 4 GB DDR2 memory.

B. Hardware/Software Partition and Memory Hierarchy

We run the profiling on the index server software with an index of 8,240,027 documents and a query log of 1.767 queries, which are typical data sets from a commercial search engine. The software is implemented with C++ and highly optimized for speed. The computer is a 2.5GHz, 4-core Intel Xeon server with 16GB memory. It shows that matching and L1 ranking occupy 69% of total CPU time, and they are bounded by memory bandwidth. So we put them into Pinaka board for better bandwidth$/$. Initialization, L2 ranking and post processing are assigned to software since they contain many rules and may be changed frequently by software developers. Please be noted that the software part could be shared among many Pinaka boards. With this partition scheme, we design a memory hierarchy to support matching and ranking on Pinaka. Flash memories are used to store the major inverted index. DDR2 provides higher memory bandwidth and is used as storage for static features. FPGA on-chip memories stores the ranking models, the context of current query and other temporary results. A Nios-II processor implements the Flash Translation Layer [7], and manage the communication between Flash and the major query processing logics. Figure 2 presents the architecture of the system on Pinaka.

C. Major Functions

1) Inverted Lists Compression/Decompression: Usually inverted indices are represented by a sequence of integers. These integers are compressed to save more space. The most important criterions for index compression methods are the compression ratio and the decompression speed, since compression is one-time for an index and decompression occurs at each access. The most popular compression methods include Variable-Byte [8], PForDelta [6], and Rice [2].

We implement a Variable-Byte decoder for inverted lists with a new format. In Variable-Byte coding, an integer n is encoded as a byte-bounded bit sequence. In each byte, the seven lower bits are used to store the binary representation of the encoded integer and the highest bit is used as a flag to judge whether the next byte is also the coding result of n. Variable-Byte coding is known to offer a good tradeoff between time and space. And its decompression is also simple to implement with FPGA. In our format, an inverted list consists of document IDs (noted as docIDs) and two dynamic features (t f and DocCount). DocCount indicates the number of documents a term appears. Figure 3 shows a case of this format.

The decompression runs online when inverted lists are fetched. We implement a 13-state FSM (Finite State Machine) with the pre-fetch technique to ensure the decompression module could output one docID at each cycle.

2) Matching: Matching is to traverse the inverted lists and applies boolean operations (intersection, union and subtraction) to them for the matched documents according to the input queries. Meanwhile, matching is also making preparations for scoring. With the increasing number of indexed documents, the inverted list may be very long. Thus, some optimized techniques emerge to improve the throughput of matching, such as adding the skip lists for fast seeking, terminating the inverted list if the score from static features is lower than the threshold [2], etc. At the first step, to simplify this problem and narrow down the design exploration space, we focus on basic functions of matching.

Matching is implemented with a full binary tree to utilize the bandwidth of all available flash channels. From the statistics of
We use a neural network to score document functions, which could consume features in a streaming mode. We implement the full FPGA-based query processing system in Figure 2, which consists of all the discussed functions, a Nios-II CPU, memory controllers, and all necessary interconnection among them. The design is synthesized with Quartus II 10.0. Decoder and ranking take most of logic elements (LEs), and the full logic occupies 83.63% of all available LEs on an EP3C120. It is hard to fit all the 18 channel controllers in the current device. In practice we only use one flash channel to serve queries, and it becomes the system performance bottleneck in current design.

B. Experimental Setup

The baseline software-based index server is from a commercial search engine. This production level system and data sets are the same as mentioned in Section II-B. The proposed FPGA-based index server is composed of an Intel workstation (Pentium D dual-core 3GHz, 8GB memory, 64-bit Windows server 2008), and two Pinaka boards. The software that drives our accelerator boards is derived from the baseline software implementation.

C. Hardware Implementation and Performance

We implement the full FPGA-based query processing system in Figure 2, which consists of all the discussed functions, a Nios-II CPU, memory controllers, and all necessary interconnection among them. The design is synthesized with Quartus II 10.0. Decoder and ranking take most of logic elements (LEs), and the full logic occupies 83.63% of all available LEs on an EP3C120. It is hard to fit all the 18 channel controllers in the current device. In practice we only use one flash channel to serve queries, and it becomes the system performance bottleneck in current design.

D. QPS of the System

We first measure the performance of each critical function and compare it with the baseline software. For the decomposition function in hardware, the FPGA implementation processes 87349165.58 documents/s and achieves up to 3.58X bandwidth compared with the software implementation. Given that each query will touch 1508.8 documents on average, we can calculate that the performance for the decoder is: QPS_decoder = (Documents-per-sec / Documents-per-query) = 57893. Compared with the baseline system, the decoder is not the system bottleneck. For the matching function in hardware, the throughput of the hardware implementation is 43.75 MDoc/s and provides three orders of magnitude speedups over the software. Thus, we could directly calculate the QPS for matching tree as: 43.75e6/1508.8=28996.55. For the ranking function in hardware, the throughput of the hardware implementation is 282.49 KDoc/s and provides 4800X speedup in processing capability. It suggests that FPGA is fairly good at computation intensive applications. Also, the sustainable QPS is 282.49e3/178.89=1579.13.

Besides, the implemented DDR2 memory interface runs at 133 MHz and provides 2.133 GBps bandwidth (50% efficiency). Since DDR2 is mainly used as the storage for the static
features (48 Bytes/Doc), and each query will have 178.89 documents to be ranked, the DDR2 interface could provide a QPS of 2.133 GBps/(48*178.89)=266724.86. Each Flash channel could provide 50 MBps bandwidth to the 50MHz Flash chips. Since Flash is used at the storage for inverted lists, and each query will touch around 1.38 MB index, we can calculate that each flash channel can support a QPS of 50 MBps/(1.38 MB/query)=36.23. In current design, we fit only one Flash channel into the Cyclone III FPGA. Thus, the bottleneck for the whole system is the bandwidth of flash chips. With this accelerator, the QPS of the whole system is measured as 6.4. This number is much smaller that the theoretical bottleneck capability of hardwired decoder, matching tree and ranker is wasted due to the limit memory bandwidth to flash memories, since only one flash channel is used in the current design. The efficiency could be dramatically improved to the level of the targeted system by i) using latest high bandwidth Flash chips and larger FPGAs; and 2) further improving the effective bandwidth to the inverted index by utilizing the DDR2/DDR3 memory as the cache for flash memory. Now we are building the next version FPGA/Flash based accelerator system to achieve an even higher efficiency than the CPU-based query processing system.

IV. Conclusion

In this paper, we propose a novel architecture on a customized FPGA acceleration board for efficient query processing in a Web search engine. We explore the design space for essential components in query processing on the new platform and implement the whole system. It could be used in a real search engine. Furthermore, our experimental results show that the targeted system could achieve up to 19.52X power efficiency and 7.17X price efficiency compared with an Intel Xeon server.

ACKNOWLEDGMENT

The authors would like to thank Zeng-Lin XIA, Bo-Jun HUANG, Jing-Jing XIAO for their support in this work.

REFERENCES