EFFICIENT HARDWARE SEARCH ENGINE FOR ASSOCIATIVE CONTENT RETRIEVAL OF LONG QUERIES IN HUGE MULTIMEDIA DATABASES

Christophe Layer, Hans-Jörg Pfleiderer

Department of Microelectronics, University of Ulm,
Albert-Einstein-Allee 43, D-89081 Ulm, Germany
christophe.layer@e-technik.uni-ulm.de

ABSTRACT

Due to the enormous increase in the stored digital contents, search and retrieval functionalities are necessary in multimedia systems. Though processor speed for standard PCs (Personal Computers) is experiencing an almost exponential growth, the memory subsystem handicapped by lower frequencies and a physical I/O (Input/Output) limitation reflects the bottleneck of common computer architectures. As a result, many applications such as database management systems, remain so dependent on memory throughput that increases in CPU (Central Processing Unit) speeds are no longer helpful. Because average bandwidth is crucial for system performance, our research has focused especially on techniques for efficient storage and retrieval of multimedia data. This paper presents the realization of a hardware database search engine based on an associative access method for textual information retrieval. It reveals the internal architecture of the system and compares the results of our hardware prototype with the software solution.

1. INTRODUCTION

Information retrieval has changed considerably with the expansion of the WWW (World Wide Web) and the advent of modern and inexpensive mass storage components. With the large ubiquity and mobility of multimedia enabled devices, UMA (Universal Multimedia Access) [4] emerges as an fundamental part within new applications, for which text search in mass data has become a functionality of growing importance. As a matter of fact, digital documents have become nowadays one of the most common and practical means of communication and memorizing information [1]. But as their quantity has dramatically increased, finding them rapidly has become a real challenge of the utmost importance that search engines intend to overcome. Hence their goal is to direct a user to a small selection of documents that simply contain words that match a set of searched terms. Even though millions of documents may be returned, the search engines also try to sort the results so that only a handful of the most relevant matches are presented first. Searching through always cheaper modern electronic storage media like magnetic or optical disks involves the moving of an enormous quantity of data between the memory and the processor, and therefore implies a very long query time.

The solution is to provide a hardware support for the error tolerant storage and retrieval of electronic objects described by a set of binary features. Depicted in Figure 1, our system is a high performance database search engine bound to the main processor of a computer and thus releasing it for intricate query tasks. The

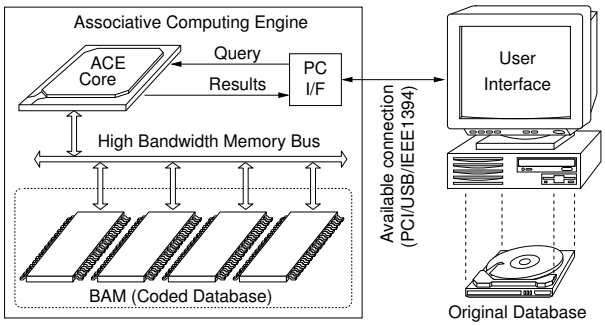

Fig. 1. The hardware solution proposal for a query accelerator is able to offload data processing onto a dedicated daughter board.

ACE (Associative Computing Engine) presented here is based on a unified approach that applies to different data structures databases, ranging from multimedia applications to Internet data. Based on a textual paradigm, queries are performed locally inside a coded database known as BAM (Bit Attribute Matrix) [5, 8] which is an image of an original multimedia container. Following this introduction, Section two gives an overview of the query procedure and reviews the associative computation method. Section three presents the internal architecture of the ACE that makes it so fast. The last Section discusses the performance of our hardware prototype compared to the software solution.

2. TRIGRAM BASED ASSOCIATIVE MATCHING

2.1. Retrieval Process

Created by hashing texts extracted from the original database which it refers to, the BAM [8, 9] contains textual properties of each page in form of binary signatures. Those are generated by hashing each trigram within a confined text to a single bit position in the bit string. Hashing techniques are used to significantly reduce the memory overhead associated to the operation with large sparse matrices. Moreover, they permit a non exact pattern matching and an extremely flexible retrieval suitable for many object types. In our database system, textual documents are segmented in pages and their contents are analyzed and stored as a set of binary features in the BAM. A row of the matrix formed by repeating this procedure for all the text pages corresponds to a page signature. Each column index codes the availability of one or a group of bi-

0-7803-9332-5/05/$20.00 ©2005 IEEE

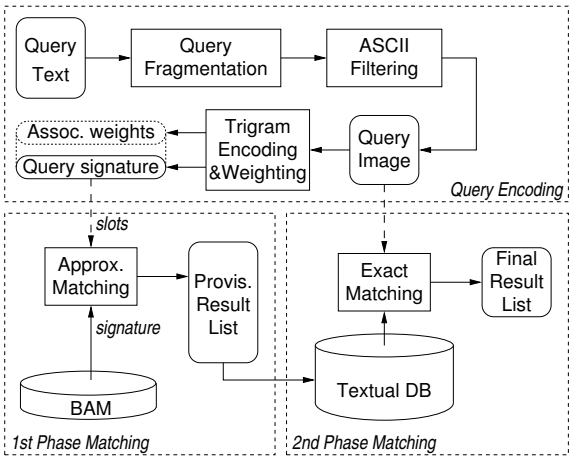

Fig. 2. The two searching phases of the retrieval process after the signature encoding of the query string.

nary features with a "1" or its absence with a "0" [8]. The length of the bit string and the hash function are chosen so that approximately half of the bits are set to "1", maximizing the entropy of the matrix since this is theoretically the most efficient arrangement.

Depicted in Figure 2, the retrieval algorithm includes two complementary phases (approximate and exact matching) which are performed after the preprocessing of the query text, including fragmentation, trigram encoding into query-slots (which form the query signature) and the weighting of important words in a sentence. Partitioning long queries into fragments permits a better matching at the string level and allows to recover documents in which parts of the original query appear. The approximate search phase, in which the query signature is compared with the attributes of the BAM to identify the records having a high probability to contain the desired query information [5], is a rough but very fast procedure. It is used in connection with the BAM to exclude a large number of non-relevant documents from the search. Afterwards, the remaining candidates are sorted in descending order of relevance and passed on to the second stage. Having for purpose to refine these results, the second phase is much more complex but can be very short if the previous one was selective enough. Records from the first phase matching at the bit attribute level are then filtered using more rigid pairing procedures.

2.2. Accelerated Matching Computations

Since the first phase is the longest one considering the enormous amount of page-entries in the BAM which all have to be treated, there is a particular need to optimize the memory accesses as well as the processing method. Therefore we decided to implement a hardware accelerator performing the approximate matching computations which correspond to calculation of the distance between two strings, giving an appreciation of their similarity. Based on the information present in the page-signatures of the BAM (the availability of a certain trigram in the page), we punish bad matching pages more or less badly, according to the weights given to the different words composing a query fragment. A "mismatch penalty" is assigned if a page does not have a binary feature which is present in the query. A "switch penalty" is given when features appear or disappear along the searching procedure, because as the query-slots are processed sequentially, when successive features are found in the page-signature, there is a fair chance that they belong to the same sequence in the original page, and the opposite applies. The smaller a penalty, the bigger the amount of similarities to one query fragment found in the BAM page. Over the processing of one fragment i, the pealty yields

$$\mathcal{P}_{Fi} = \sum_{s} (switch + mismatch) \times \mathcal{W}_s \qquad (1)$$

where $\mathcal{W}_s$ is the weight associated to the query-slot s. The penalties acquired for one BAM-page are combined together in order to provide an overall rating. The score $\mathcal{S}_{Page}$ is adjusted after each query-fragment i by summing up the logarithm of the penalty $\mathcal{P}_{Fi}$ complemented to a constant $\mathcal{K}$ as in

$$\mathcal{S}_{Page} = \sum_{i} \left(\mathcal{K} - \log\left(\mathcal{P}_{Fi}\right)\right). \qquad (2)$$

The purpose of the logarithm is to sum up big values only for very small fragment penalties. In case of a linear function, large penalties of bad matching fragments would impact the result and suppress the small penalties of good matching fragments [8]. The remaining processing in phase one is the sorting of all the pages according to their score $\mathcal{S}_{Page}$ and the selection of the best ones.

3. ASSOCIATIVE COMPUTING ENGINE

The development of a new kind of hardware search engine was motivated by the advantages offered by associative processors having a high system scalability, without being bounded by the range of the address bus [7]. In such implementations, the processors participate in an operation only on the basis of the data they hold.

3.1. Internal Hardware Architecture

Data parallelism is a basic concept used for associative searching [12] applied to the ACE extending the data parallel paradigm to a complete computational model. The regular matrix shape of the BAM, as seen in Figure 3, allows an efficient access to the data, in form of a bit vector, which is read vertically [5]. For each access, the basic information is then one bit per BAM page. In other words, reading N bits from the BAM affords the processing of N pages at the same time by the ACE. If the evaluation of the data afterwards can be rapid enough, the bottleneck of the system

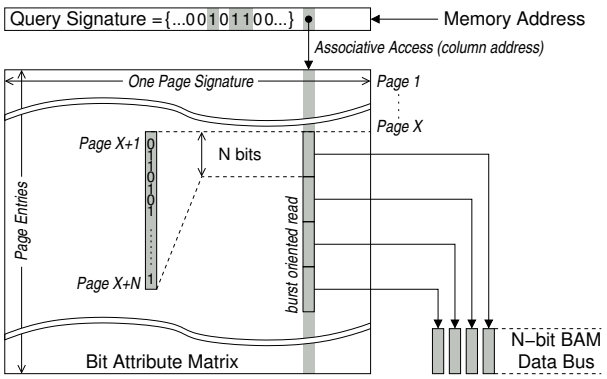

Fig. 3. Vectorized data accesses to the Bit Attribute Matrix.

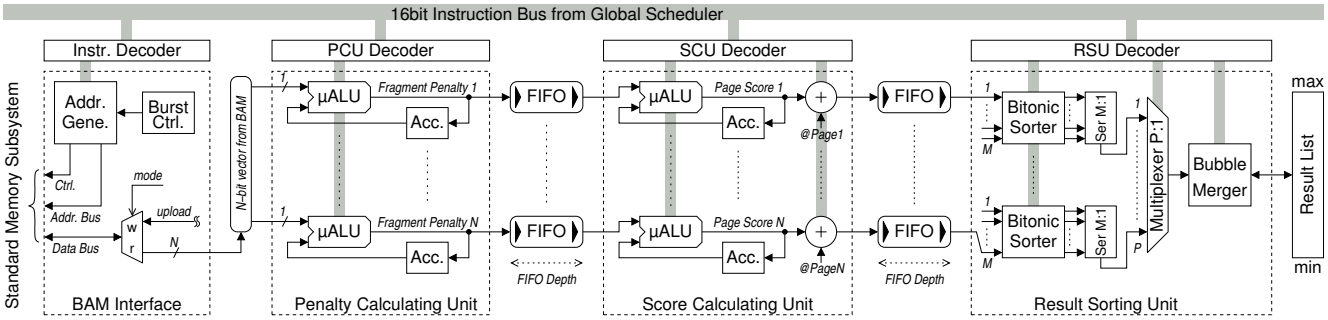

Fig. 4. Internal hardware architecture of the ACE showing the 3 modules (PCU, SCU and RSU) performing the first phase of the search algorithm plus a BAM interface that ensures efficient data exchanges, independent from the type of memory used.

will remain at the BAM interface. Therefore, we had to design a processing unit able to handle the data as quickly as possible, thus ensuring a maximization of the BAM throughput.

As seen in Figure 4, the core of the ACE is made of a very regular structure of small μALUs (micro-Arithmetic and Logic Units) allowing a vertical scalability among N parallel sections. Each section processes one bit of the BAM vector and follows the equations related to the pattern matching arithmetic described previously. It performs the first phase of the search algorithm, the realization of which is divided into many sub-stages corresponding to the parallel processing of slots, fragments and pages, i.e. respectively the PCU (Penalty Calculating Unit), the SCU (Score Calculating Unit) and the RSU (Result Sorting Unit).

The BAM interface is responsible for fetching the correct lines and rows of the matrix from the memory subsystem into the processing units. Depending on the mode set, it allows either a rapid transfer of the data from the host computer to the BAM container bypassing the hardware accelerator in write configuration (mode `w`) or a burst oriented associative access to the BAM for the querying activities in read configuration (mode `r`). Moreover, it serves as an interfacing adaptor between a physical memory system and an abstract scalable associative computing engine. The PCU calculates the penalties yielded for each query-fragment and actualizes the result in its accumulator, according to equation (1). At the end of each fragment, the result consisting of N fragment-penalties of the query in N pages is passed to the next processing unit. The SCU converts these penalties into N page-scores for all the fragments in the query string, according to equation (2). The accumulation of the logarithms calculated sequentially with a very compact hardware circuit as in [11] occurs during the processing of the next fragment by the PCU. When PCU and SCU calculate the scores of N new pages, the RSU, last unit in the processing chain, keeps the best matching pages from the N previously calculated ones and merges them in the result list.

3.2. Lower Bounds on Modular Parallel Processing

As depicted in Figure 4, the highly pipelined internal architecture of the ACE allows a very fast parallel processing of the data read from the BAM. Because many physical media are suitable for storing the BAM, the ACE has been designed very flexibly, so that it can interface different memory types. Therefore, the FIFO (First-In First-Out) buffers inserted between the processing units are meant to handle burst accesses performed by the memory subsystem. Moreover, they synchronize the data exchanges and permit an efficient scheduling referenced to the PCU. They allow, without any input multiplexer, to reuse the same computing hardware in the PCU and in the SCU with time-multiplexed uncorrelated data. Let BL be the length of a burst. Considering that the PCU has to read data from the BAM continuously, we must have the execution time t in clock cycles such that

$$t_{SCU} \cdot BL \leq t_{PCU} \cdot BL \quad \text{and} \quad t_{RSU} \cdot BL \leq t_{PCU} \cdot BL. \quad (3)$$

As seen in Figure 4, the sorting method of the RSU is partially based on a recursive implementation of an M input parallel Bitonic sorting algorithm [2, 6] with a complexity of $O(\log_2^2 M)$, followed by a serial Bubblesort merger of depth R with a linear complexity of $O(R)$. Using P serializers (with $P = N/M$) and one multiplexer, the merger inserts a new value from the output list of each bitonic sorter only if it is larger than the minimum score in the result list. Moreover, the RSU has the possibility to slow down the PCU if it needs more time to sort the results in order to avoid an overflow in the FIFOs. But this case occurs only at the very beginning of the search process when the result list is empty.

When the result list is full, no new pages from the SCU are likely to be inserted. This is a probabilistic remark that allows us to design a much simpler system which has the ability, at very low costs in terms of delay and latency, to adapt the calculation intensive starting phase on a reduced architecture. With these considerations, the time for the RSU module converges to

$$t_{RSU} = BL \cdot \left(\log_2^2 M + P \right) \quad (4)$$

after a small amount of time. Thus, two parameters play an important role from the hardware point of view: i) the increasing of the width N of the architecture, which slows down the RSU, increases the core area of the ACE but reduces the overall search time. ii) the length BL of the burst accesses, which increases the depth of the FIFOs, but makes the ACE suitable for more media types. By setting the depth of the FIFO to one, the architecture might only support direct addressing memory devices, though its area would be significantly reduced.

4. PERFORMANCE MEASUREMENT

For testing the architecture and the hardware implementation, we used a prototyping platform for a reconfigurable SoC (System on Chip) design. The board is equipped with generic components providing the required communication infrastructure to link the ACE to a personal computer through a PCI interface. It includes

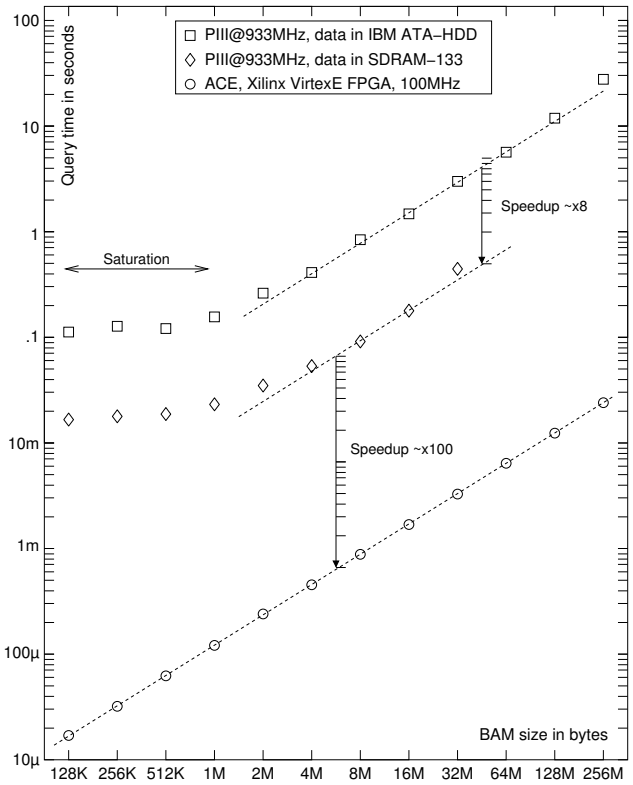

Fig. 5. Performance graph of the search algorithm running on a PC with the BAM is located in SDRAM or in HDD compared to the ACE implemented in a Xilinx FPGA.

an FPGA from the Xilinx Virtex-E series which can emulate circuits with a complexity of four million system gates and 128 MB of SDRAM (Synchronous Dynamic Random Access Memory) accessible over a 32 bit bus. The model works with a 100 MHz clock frequency and can process 32 pages per clock cycle. On the other hand, the software model runs under Linux with a Pentium™III with 256 MB SDRAM. For the simulation of queries in databases of different lengths, the PC encountered problems for BAMs bigger than the available memory space. Dynamic allocation failed and the search process was aborted.

Figure 5 shows the time needed for the searching of one sentence of text for both the PC and the ACE. It appears clearly as expected that the search time varies linearly with the size of the BAM. The saturation zone on the left part of the figure reflects some side-effects for the queries performed on the PC only, where the BAM was stored on the HDD (Hard Disk Drive) and in the RAM. On the one hand, the operating system needs to administer some tasks with a higher priority than the search in a multi-tasking environment. On the other hand, the access time to a hard disk is quite long, most of all because of the positioning of the heads at the right disk sector. This intrinsic constraint leads to the fact that the simulated HDD-based queries could not run under 100 ms.

Further measurements proved the linearity of the method to the length of the query as well as the size of the BAM. The second phase of the search – dealing with an exact matching of the retrieved documents, as explained in Section 2 in this paper – was not considered in our benchmark. Its duration is very short and depends exclusively on the amount of matching entries returned by the first phase of the search. Nonetheless, it has been experienced that, depending on the quality of the feature engineering, the second phase might even be entirely omitted.

5. CONCLUSIONS

We proposed here a fast scalable architecture for the computing of search algorithms based on approximate matching. With its 100MHz clock frequency, the ACE brings a speedup factor of two and three orders of magnitude compared to the actual processor implementation where data is read from the SDRAM and from the HDD of the PC respectively.

The performance of the ACE is directly linked to the bandwidth of the BAM interface. Increasing the throughput of the memory device would consequently diminish the searching time. Compared to a previous work described in [10], the system presented in this paper proposes an architecture suitable for burst oriented storage-media types.

In addition, the ACE offers an extremely low power consumption compared to a PC running at many GHz. Due to its low cost and very reliable high speed design, the ACE is meant to be used in multimedia applications devices with reduced or limited power supply, requiring efficient and fast data mining.

6. REFERENCES

[1] R. Baeza-Yates, B. Ribeiro-Neto, “Modern Information Retrieval”, *Addison Wesley*, 1999, ISBN: 0-201-39829-X.

[2] K. E. Batcher, “Sorting Networks and Their Applications”, *AFIPS Proc. Spring Joint Comp. Conf.*, pp. 307-314, 1968.

[3] K. E. Batcher, “Bit-Serial Parallel-Processing Systems”, *IEEE Trans. on Comp.*, Vol. C-31, No. 5, pp. 377-384, 1982.

[4] P. van Beek, J. R. Smith, T. Ebrahimi, T. Suzuki, J. Askelof, “Metadata-Driven Multimedia Access”, *IEEE Signal Processing Magazine*, Vol. 20, No. 2, pp. 40-52, 2003.

[5] S. Berkovich, E. El-Qawasmeh, G. M. Lapir, “Organization of Near Matching in Bit Attribute Matrix Applied to Associative Access Methods in Information Retrieval”, *Proc. 16th IASTED Conf. on Applied Informatics*, pp. 62-65, 1998.

[6] D. E. Knuth, “The Art of Computer Programming, Vol. 3, Sorting and Searching”, 2nd Edition, *Addison Wesley*, 1997.

[7] A. Krikelis, C. C. Weems, “Associative Processing and Processors”, *IEEE Computer*, Vol. 27, No. 11, pp. 12-17, 1994.

[8] G. M. Lapir, “Use of Associative Access Method Information Retrieval Systems”, *Proc. 23rd Pittsburgh Conf. on Modeling and Simulation*, Vol. 23, No. 2, pp. 951-958, 1992.

[9] G. M. Lapir, H. Urbschat, “Associative Memory”, *International Patent*, IPN. WO 02/15045A2, WIPO, 2002.

[10] C. Layer, H.-J. Pfleiderer, P. Ruján, G. Lapir, “High Performance System Architecture of an Associative Computing Engine”, *Proc. IFIP WG10.5 VLSI*, pp. 74-79, 2003.

[11] C. Layer, et al., “A Scalable Compact Architecture for the Computation of Integer Binary Logarithm Through Linear Approximation”, *Proc. ISCAS*, pp. 421-424, 2004.

[12] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, C. Asthagiri, “ASC: An Associative Computing Paradigm”, *IEEE Comp.: Associative Proc.*, pp. 19-25, 1994.