Bit-Sliced Signature Files for Very Large Text Databases on a Parallel Machine Architecture

George Panagopoulos and Christos Faloutsos[⋆]

Department of Computer Science
and Institute for Systems Research (ISR)
University of Maryland, College Park, MD 20742

Abstract. Free text retrieval is an important problem which can significantly benefit from a parallel architecture. Signature methods have been proposed to answer text retrieval queries in parallel machines [Sta88, LF92], under the assumption that the main memory is sufficient to hold the entire signature file. We propose the use of a Parallel Bit-Sliced Signature File method on a SIMD machine architecture when the size of the signature file exceeds the available memory. We propose that we need not examine all the bit slices; instead we use a *partial fetch* slice swapping algorithm. This method achieves graceful performance degradation according to the database size. We provide formulae for the optimal number of signature slices to fetch and match with the query signature. Arithmetic examples show that our method can handle a 128GB database with a 2sec response time on a machine with the characteristics of the Connection Machine.

1 Introduction

Free text retrieval in large text databases is an important problem involved in numerous applications, for example electronic office filing [TC83], computerized libraries [SM83], and electronic encyclopedias. Text databases are traditionally large, unstructured and archival in nature. Retrieval methods based on inverted indices introduce large space overhead (typically 50%–300% [Has81]) and large insertion cost. Signature-based text retrieval methods [CF84, FC87, Fal90] constitute an alternative between fast but space-expensive inversion methods and full text scanning. Signature methods have been shown to have a modest space overhead (typically 10%–15%), to be efficient in text retrieval and insertion, and to be well suited to the archival nature of text databases.

The signature methods also seem to be well suited to the concept of parallelism introduced with the development of massively parallel machine architectures. In signature methods, queries are answered using a two-stage retrieval

⋆ This research was sponsored partially by the Institute for Advanced Computer Studies (UMIACS), by the National Science Foundation under the grants IRI-8719458, IRI-8958546 and IRI-9205273, by a donation by EMPRESS Software Inc., and by a donation by Thinking Machines Inc.

mechanism: first, the query signature is compared with the set of stored document signatures in a bit matching operation which identifies the documents that have a high probability of containing the required query words; then a detailed character matching procedure is invoked for the documents that qualify in the first stage. Large text database applications would benefit significantly by parallelism, especially in the bit matching step, as previous work has suggested [Sta88, Lin92].

The paper in [Sta88] presented a parallel signature-based algorithm for high-speed interactive querying of a text database on a SIMD computer, the Connection Machine. The results were based on the assumption that the main memory was sufficient to hold all the document signatures. In fact the algorithm does not extend well to the case of larger databases that do not fit in the Connection Machine memory. The Sequential Signature File method used in the paper requires the loading of the entire signature file, which makes retrieval prohibitively expensive for interactive applications.

Our motivation was to investigate the applicability of signature methods in SIMD architectures for very large text database applications, in which the signature file size exceeds the memory capacity of the computer system. The main contribution of this paper is the presentation of a parallel signature method that fits better to a SIMD architecture for large database applications. This method is based on the Bit-Sliced Signature File (BSSF) method as described in [FC88]. When the signature size exceeds the available main memory of the computer system in use, we propose the use of *partial fetch slice swapping*. The idea is to examine the bit slices already in memory and to fetch as many from the disk as necessary. Another contribution is a derived formula for the optimal number of signature slices to fetch and match against the query. Furthermore, we provide a performance analysis of the proposed method and we investigate it for various database sizes and other parameters of the application and the system architecture. We show that the proposed scheme on a machine with the characteristics of the Connection Machine can handle interactive text retrieval on databases of size in the order of 50GB. Finally, we show that the method is general enough to be also applicable on uniprocessor machines.

This paper is organized as follows. Section 2 is an introduction to the terminology used and a review of the Bit-Sliced Signature File method. Section 3 contains a description of the parallel system architecture model used and the proposed Parallel BSSF method. Section 4 contains the results of an analytic performance evaluation of the proposed retrieval algorithm. In section 5 we discuss the performance of the method for various application and system parameters. Finally, section 6 contains our conclusions.

2 Background

In this section we will present our terminology and review the Bit-Sliced Signature File text retrieval method [FC88]. The algorithms for insertion and retrieval will be presented.

For the purposes of signature construction, each document is considered here as a set of words. The words that appear in queries are called *terms* and are the only ones that take part in the construction of the document signatures. Each term in a document is hashed by m independent hash functions to m bit positions (not necessarily distinct) in a bit vector of length F. The corresponding bits will be set to 1, all the others being set to 0. The resulting value of the bit vector is the *term signature*. The *document signature* is constructed by superimposing the term signatures, in other words by logically OR'ing them together. The following table illustrates the method for an example document that consists of two terms, with $F = 9$ bits per signature and $m = 3$ bits per word:

Word	Signature
`free`	`001 000 110`
`text`	`000 010 101`
Document	`001 010 111`

Suppose that there are N documents. The collection of the N document signatures forms an $F \times N$ bit matrix, called the *signature matrix*, which will be stored in the *signature file*. The signature file contains only the document signatures. The actual document text data are stored in a *text file*. For every document the method maintains a *posting*, which is a pointer to the beginning of the document in the text file. All postings are stored in a separate *postings file*.

The *insertion algorithm* simply constructs a signature for each document inserted by superimposing the signatures of the contained terms, and appends the resulting document signatures to the signature file.

The *retrieval algorithm* answers queries, which are boolean formulae with terms as literals. During the first step of the retrieval algorithm, the query signature is constructed by applying the same hashing functions to the terms of the query. Then a bit matching operation is performed with the signatures stored in the signature file. A document potentially qualifies with respect to a query term if all the signature bits set in a query term are also set in the document signature. Every query term produces a list of documents that potentially contain this term. Then these lists are merged according to the boolean operators contained in the query.

The exact algorithms for insertion and retrieval of the documents depend on the organization of the signature file. In the *Sequential Signature File (SSF)* method, which is the basic organization, signatures are stored sequentially. This organization is useful for small database applications only, because retrieval requires the sequential search of the entire signature file. In the *Bit-Sliced Signature File (BSSF)* method [FC88], on the other hand, the signature matrix is vertically partitioned in F bit slices. Each bit slice is stored in consecutive disk blocks. Since only the bit slices relevant to the query need to be examined, the BSSF method permits the loading of only the relevant disk blocks of the signature file. For example, only at most m bit slices are required for a single word query. BSSF is more efficient than SSF in retrieval because of the smaller amount of data that needs to be loaded from the signature file.

The signature methods effectively perform a filtering of the entire set of the documents, eliminating the ones whose signature does not match the query signature. By an efficient application of this method, most of the non-qualifying documents will be eliminated without being read at all. Some of these documents may pass the filter, though, because of the superimposed coding. In the signature example above, if a word happens to have a signature "`001 010 100`", a query on it will retrieve the illustrated document in the answer, even if the word is not contained in the document. These cases are called *false drops*. The retrieval algorithm must eliminate the false drops by retrieving the actual documents and checking them against the query. This false drop elimination step can impose a very serious performance overhead if the design parameters allow for a large proportion of false drops. Therefore a major design criterion for the efficiency of a signature method is a small false drop probability. The latter is defined as the fraction of irrelevant documents whose signatures match the query signature.

3 The Proposed Parallel BSSF Method

In this section we will describe the application of the Bit-Sliced Signature File method on a SIMD parallel machine architecture. We will first present the architecture model of the underlying hardware. Then we will describe the application of the Parallel BSSF method on the architecture model used.

3.1 The System Architecture Model

The architecture model that we will use follows the parallel processor model in [Sto87]. The model corresponds to a SIMD machine architecture like the Connection Machine. It contains an array of P *parallel processors*, each with its own local memory of size M bits. The operation of the parallel processors is synchronized by a *host processor*, which also works as a front-end for the users. The host processor is equipped with its own (serial) I/O system.

The model includes a *parallel I/O bus* and a *parallel I/O system*. A file stored in this I/O system is not a single stream of bits, like in a serial I/O architecture, but a multiple stream of bits, one stream per processor. Such a *parallel file* is assumed to be formatted in a way that reflects the processor array structure. It can be depicted as a stream of P-bit vectors; each vector corresponds to a parallel bit array stored along the P processor memories. During an I/O transfer, each processor p will access bit p of the bit vector at the current file position. The file is said to have *width* equal to P, the size of this bit array.

In the following, we will use the term *processor* to mean one of the parallel processors, unless explicitly stated otherwise. These processors can execute *local operations* (e.g. ALU-associated), *global operations* (global maximum, global logical AND, broadcast and other similar operations over the entire array of processors) and *I/O operations* (parallel transfer from or to the parallel I/O system).

3.2 The Application Model

In this paper we are mostly concerned with conjunctive queries. The desired result is a list of documents that contain all the query terms. It is easy to extend the basic retrieval algorithm to accommodate more general boolean queries with no negated terms. One can simply convert the query in disjunctive normal form and merge the lists of documents that qualify for each conjunct. As in previous papers (see for example [FC88]) we will assume that the database consists of N documents with D terms on the average. When very long documents exist in the database, they can be broken down into "logical blocks", each with D terms.

3.3 The Data Structures

The modification of the serial BSSF method to the parallel architecture of our system model is straightforward. The text file will be kept in the host processor I/O system, while the signature file will be stored in the parallel external storage in bit-slice form, and appropriate portions of it will be loaded into the processor memories on demand, by a "slice replacement" algorithm.

We will assume that M bits of every processor's memory can be allocated for storage of signature bits[2]. Figure 1(a) illustrates our proposed implementation with an example consisting of a database with $N = 8$ documents and a system with $P = 4$ processors. Each processor has $M = 4$ bits of memory available for signatures, and will be assigned $V = \lceil N/P \rceil$ document signatures.

A simplified view of the allocation scheme is to consider each processor as a *physical processor* that corresponds to V *virtual processors*, which are assigned one document each. These V virtual processors share the resources of the physical processor, i.e. they execute in round-robin fashion and each has $R = \lfloor M/V \rfloor$ bits of memory available for signatures. This scheme is equivalent to the virtual processor scheme of the Connection Machine. It permits us to view the system as if it really consisted of $V \cdot P$ processors, each having R bits of main memory. In the following we will adopt this scheme and by *processor* we will mean *virtual processor*, except when we explicitly state *physical processor*. In figure 1(b) we denote the virtual processors by VP_0, VP_1, etc. Note that the physical processor P_0 corresponds to the virtual processors VP_0 and VP_4. In general, if there are P physical processors, the physical processor P_i corresponds to the virtual processors VP_i, VP_{i+P}, etc. Since the virtual processors run in successive fashion and not in parallel, the algorithm steps below are meant to iterate V times in each physical processor. We will call V the *virtual processor ratio* of the system.

At every instance of time during text retrieval, we keep R bit slices of the signature file resident in main memory. A resident slice occupies one bit of each processor's memory. We will refer to the set of these bits storing a single slice as a *frame*. There are R frames in the parallel computer system. In figure 1(b) we have only $R = 2$ frames; frame 0 holds slice 0 and frame 1 holds slice 3.

[2] The algorithm requires some additional parallel memory for other data structures, but this additional requirements are small enough to be safely ignored in our performance analysis.

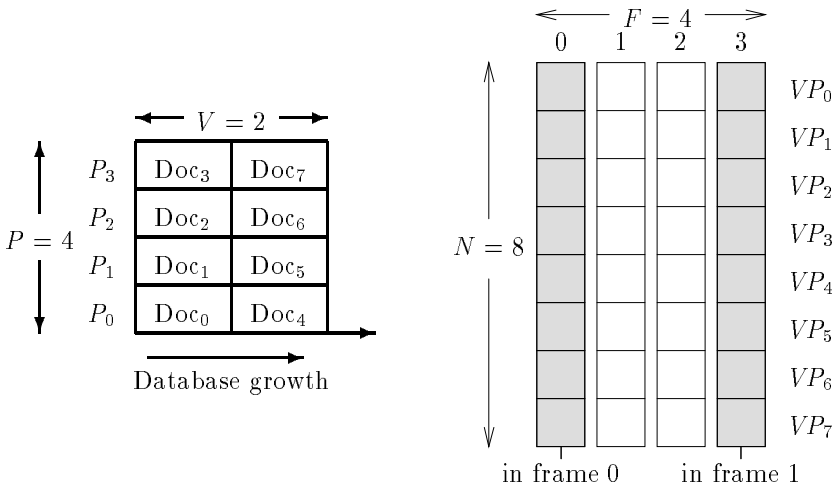

Fig. 1. Signature file in PBSSF method ($N = 8$ documents, $F = 4$ bits per signature, $P = 4$ physical processors, $M = 4$ bits per processor): (a) Allocation of documents to processors; each physical processor is assigned $V = 2$ documents (b) Signature file storage in parallel memory; 8 virtual processors, $R = 2$ resident slices (slices 0 and 3)

The resident bits of the signature file in each processor p form a *slice bit vector* of length R, which we will denote by $B_p[R]$. The relevant bits of the slice bit vectors are matched in each processor with the relevant bits of the query signature into a *response bit* R_p. The response bit R_p in processor p will be set at the end of the computation if and only if the signature document p matches the query signature.

3.4 The Parallel BSSF Algorithms

The *insertion algorithm* for the Parallel Bit-Sliced Signature File method is more efficiently executed in batches. It simply constructs the new document signatures and then appends the new bits to the signature file, slice by slice.

The *retrieval algorithm* is shown in figure 2. Given a conjunctive query, it produces a list of documents that match the query signature. It proceeds by constructing the query signature, then matching it against the document signatures stored in the database; finally, false drops are eliminated.

Step [R3] involves a decision of how may slices to fetch in memory, as well as a slice victim selection and replacement policy. We will call this procedure the *fetch policy*. If we consider the set of resident slices as the *state* of the memory, the fetch policy decides the next state given the current one and a set of slices references in the last query. The fetch policy can be quite general. It may take account of some or all of the past references and states, in addition to the current memory state.

Algorithm Parallel BSSF Retrieval

[R1] CONSTRUCT QUERY SIGNATURE: For every term w_i in the query, construct term signature s_i; then superimpose all term signatures on query signature S

[R2] INITIALIZE PARALLEL MEMORY: Initialize all response bits R_p to 1

[R3] MATCH DOCUMENT SIGNATURES: Decide how many and which, if any, slice faults to service by fetching the corresponding bit slices in main memory *(fetch policy)*; For every signature bit i that is resident in some frame r and must be matched, if the document signature bit $B_p[r]$ is 0, clear the response bit R_p; For every signature bit slice i that must be loaded, invoke a slice replacement algorithm to choose a frame for replacement and load slice i in that frame; then proceed as above to match the document signature bit with the query signature bit

[R4] OUTPUT MATCHING DOCUMENT ID'S: Find all processors p with $R_p = 1$

[R5] ELIMINATE FALSE DROPS: For each document to be retrieved, read document text from text file and scan text to check it against query

Fig. 2. The Parallel BSSF retrieval algorithm

We will call a *total fetch (TF)* policy one that always fetches all the referenced non-resident slices in memory and performs a complete signature match on all bits referenced by the query. An alternative policy is a *partial fetch (PF)* policy, which considers fetching a subset of these slices, in order to minimize the expected response time by reducing the cost of step [R3].

4 Performance Analysis

In this section we will present the results of an analytic evaluation of the expected response time of the retrieval algorithm for a query of c terms. The details of the analysis can be found in [Pan92].

We will assume that the cost of a local operation in each processor is t_{op} and the cost of a global operation is t_{glob}. The cost t_{slice} of an I/O operation that reads a slice of V bits in each processor's memory is:

$$t_{slice} = t_{seek} + V \cdot P \cdot t_{xfer}$$

where t_{seek} is the seek time for the parallel I/O storage device and t_{xfer} is the reciprocal of the maximum sustained transfer rate (measured in bits/sec) between the processor memories and the I/O system. For the host processor, let us define t_{hash} to be the cost of one hash function call, t_{scan} the scan cost per document block, and t_{doc} the cost of an I/O operation in the host processor that reads the text of one document from the text file, where:

$$t_{doc} = t_{seek} + b \cdot t_{blk}$$

where b is the average size of a document in blocks and t_{blk} is the cost of loading a disk block in memory. Here we assume similar characteristics for the serial and parallel disk systems.

Let s_q be the number of distinct bits set in the query signature, s_{res} the number of slice hits caused by the query, s_{fetch} the number of slices that the fetch policy decides to fetch into memory, and s_{match} the number of bits on which the signature match is performed (thus $s_{match} = s_{res} + s_{fetch}$). On the average:

$$s_q = F \cdot (1 - (1 - \frac{1}{F})^{cm}) \simeq c \cdot m$$
$$s_{res} \simeq c \cdot m \cdot \frac{R}{F}$$

The approximations work for $c \cdot m \ll F$. The false drop probability is:

$$F_d = w^{s_{match}}$$

where w is the average weight of the document signature, i.e. the probability that a signature bit in the signature file is set to 1. The value of w is [Sti60]:

$$w = 1 - (1 - \frac{1}{F})^{mD} \simeq 1 - e^{-mD/F} \simeq \frac{m \cdot D}{F}$$

The last approximation works for $m \cdot D \ll F$.

We will define as *response time* t_{retr} of the retrieval algorithm the time until the first qualifying document appears, and will pessimistically assume that all the false drops are encountered before any qualifying document appears. Therefore we will include all the overhead cost due to the false drops in the response time of the algorithm:

$$t_{retr} = c \cdot m \cdot t_{hash} + V \cdot t_{op} + s_{fetch} \cdot t_{slice} + s_{match} \cdot V \cdot t_{op}$$
$$+ w^{s_{match}} \cdot N \cdot (t_{glob} + t_{doc} + b \cdot t_{scan}) \quad (1)$$

In the total fetch policy case, all slice faults cause the corresponding slices to be fetched in memory, therefore we simply have $s_{match,TF} = s_q$.

Now consider a partial fetch policy. Such a policy in general decides to bring a subset of the slice faults in memory. The policy should minimize the expected t_{retr} with respect to s_{match}, given the values of s_q and s_{res}. Solving $\partial t_{retr} / \partial s_{match} = 0$ in equation (1) above we have:

$$s_{match,PF} = \frac{1}{\ln w} \cdot \ln \frac{t_{slice}}{N \cdot (-\ln w) \cdot (t_{doc} + b \cdot t_{scan})} \quad (2)$$

assuming $V \cdot t_{op} \ll t_{slice}$ and $t_{glob} \ll t_{doc}$ (of course keeping $s_{res} \leq s_{match} \leq s_q$).

5 Arithmetic Examples and Discussion

In the graphs that follow we will present and discuss some analytical results derived from formulae (1)-(2). We have assumed a database that consists of single-block documents ($b = 1$, for a block size of 4Kbytes), where each document contains $D = 100$ terms. The signature file parameters have been tuned for optimal expected response time.

We assumed an architecture with characteristics similar to that of a fully configured CM-2 [Thi89]. We have used the following parameter values: $t_{seek} =$ 25ms, $t_{xfer} =$ 5ns (i.e. transfer rate of the Parallel I/O system 200Mbits/sec), $t_{op} =$ 10ns, $t_{glob} =$ 100ns, $t_{hash} =$ 0.8ms, $t_{blk} =$ 1ms and $t_{scan} =$ 1ms.

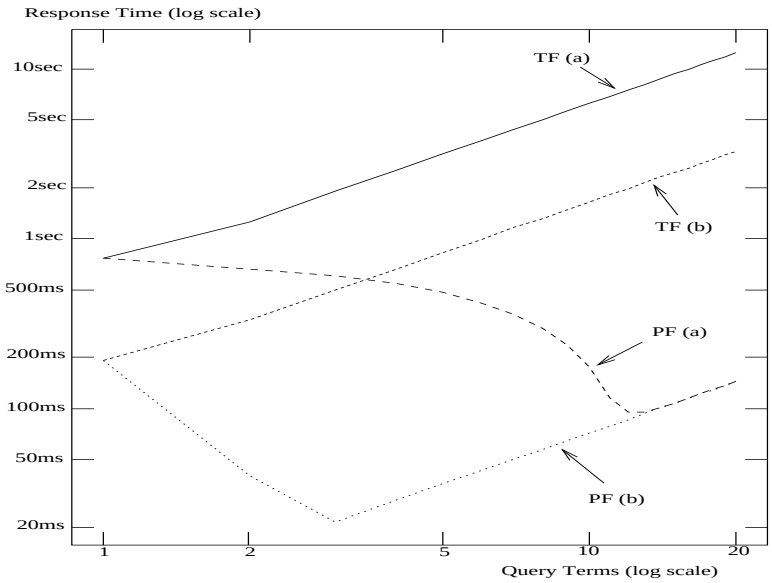

Fig. 3. Comparison of TF and PF policies ($P = 64$K processors, $M = 64$Kbits per processor) in log-log scale: (a) $N = 10$ million documents, 40GB database (b) $N = 2$ million documents, 8GB database

5.1 Comparison of Total and Partial Fetch Policies

Figure 3 compares the total fetch and partial fetch policies for different query sizes and for two database sizes. In case (a) we have assumed a 40GB database consisting of $N = 10$ million documents. The system consists of $P = 64$K processors, each with $M = 64$Kbits of memory available for signature slices. Both policies tended to select large signature lengths which produce small false drop probabilities. We chose to keep the upper limit of 4Kbits for signature lengths in order to keep a reasonable storage overhead (12.5% of the actual text file size) for the signature file. This signature length was chosen as the optimal by both TF and PF algorithms. The optimal value $m = 9$ was calculated for both policies. This gave a document signature weight $w \simeq 20\%$, as opposed to the required 50% in serial methods. The memory capacity is enough to hold only 10% of the signature file.

In the TF algorithm the false drop elimination cost quickly becomes negligible because of very small false drop probability as the query size grows larger. Almost all of the cost is signature slice I/O, which grows at a rate almost linear to the query size. The PF policy corrects this imbalance. Since the signature weight is very low, fewer slices need to be fetched for matching, while the false drop probability remains small enough to contribute very little to the retrieval cost. In fact, for queries with more than about 12 terms, the PF policy decides to fetch almost no slices at all on the average. Therefore, even though the percent-

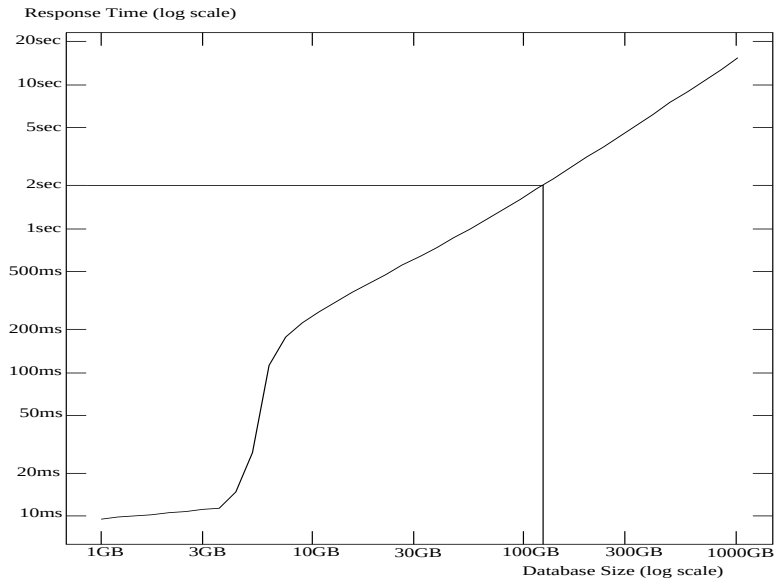

Fig. 4. Response time vs. database size ($P = 64$K processors, $M = 64$Kbits per processor, $c = 1$ terms per query, PF policy) in log-log scale

age of referenced non-resident slices increases with larger queries, the retrieval cost does not increase; the increase in response time accounts for the small (some milliseconds) increase in CPU time due to initial hash computations (construction of the query signature). The advantage of the partial fetch policy is clear and becomes even greater for multiple-word queries.

Case (b) in figure 3 shows that the same behavior can be expected from applications with relatively smaller database sizes. Here we have assumed a 8GB database consisting of $N = 2$ million documents. Both methods optimized the design for a signature length of $F = 4096$ bits and $m = 9$ bits set by each term. Now 50% of the total size of the signature file can be memory resident at any time. Again the PF policy performs much better than the TF policy. In the following graphs we use partial fetch for the PBSSF retrieval algorithm.

5.2 Effect of Database Size

Figure 4 shows the performance of the algorithm on a system with 64K processors, with 64Kbits memory each, for various database sizes and single-word queries. The database size ranges from 1GB (250,000 documents) to 1000GB (250 million documents). The diagram depicts the performance with optimal values of the design parameters F and m for the corresponding database size[3].

[3] It can also be shown (see [Pan92]) that the PBSSF method behaves well in a dynamically growing database environment, even when the database size deviates from the estimated size when the signature parameter were chosen.

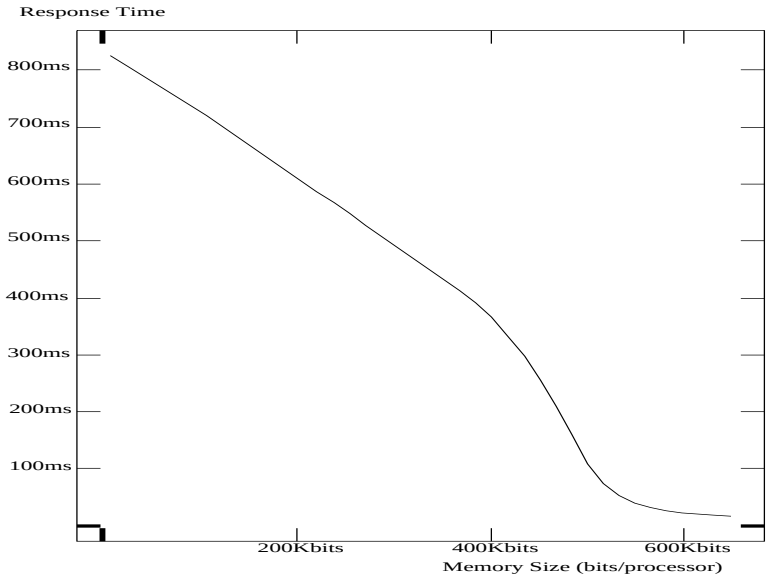

Fig. 5. Response time vs. memory size ($P = 64$K processors, $N = 10^7$ documents, $c = 1$ terms per query, PF policy); the design parameters are optimized for the given memory size

These values in general depend on the database size, but for large databases all methods favor large signature lengths, therefore they set $F = 4$Kbits, with $m = 10$ bits set per term. When the optimal signature length is used, a database of size 1GB enables the entire signature file to reside in main memory, and a database of size 1000GB permits only 0.4% of the signature file to be memory resident. The diagram shows that interactive response times (up to 2sec) are derived in such a system for databases of size up to 128GB (32 million documents). For larger databases, each processor is assigned more documents and fewer signature slices are kept resident. Even as the proportion of resident slices decreases, the algorithm keeps the number of slices to be fetched low. Nevertheless the increased size of each slice makes slice I/O more expensive. Most of the cost observed is slice I/O, and to a lesser extent false drop elimination.

5.3 Effect of Memory Size

Figure 5 shows the effect of the memory size on the response time of the algorithm for single-word queries ($c = 1$). The number of processors was set to 64K and the database to 10^7 documents. The memory size ranges from $M = 8$Kbits per processor, which gives a memory capacity of 64MB (1.25% the size of the signature file) to $M = 640$Kbits per processor, which makes a total memory size that exceeds the size of the signature file. The signature length F was optimized for each given memory size.

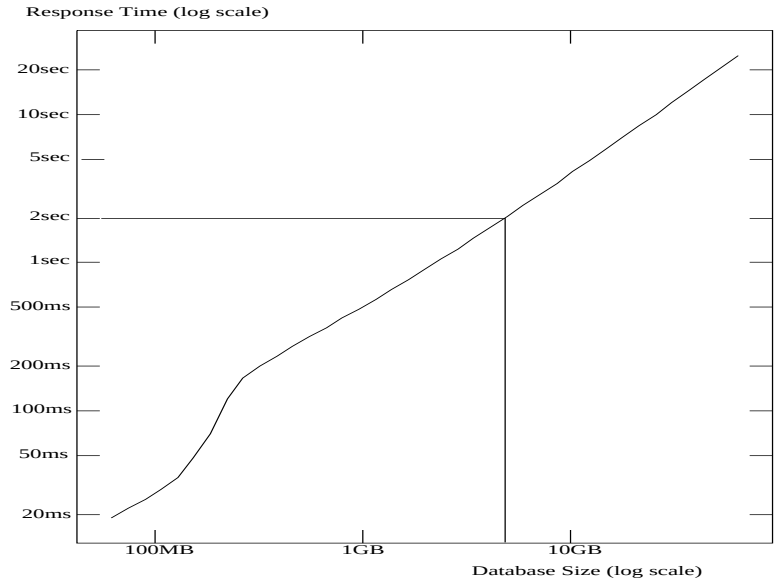

Fig. 6. Response time vs. database size on a uniprocessor machine ($P = 1$ processor, $M = 16$MB main memory, PF policy) in log-log scale

For small capacity memories most of the cost is signature slice I/O. The drop in the retrieval time on the left side of the curve savings are due to the fewer slice faults. When the memory size increases to more than $M = 400$Kbits per processor, the optimal signature length F becomes smaller and therefore the slice fault rate drops dramatically and the total retrieval cost decreases, until no more slices are fetched into memory. For large memories which make the database signature resident, the increase in memory does not offer significant savings in response time and tends to stabilize to the CPU cost.

5.4 Performance of a Uniprocessor Machine

The partial fetch/partial match idea and the analysis can be easily applied to a uniprocessor ($P = 1$) system. Our method can exploit the large main memories that are commercially available to improve performance. Figure 6 shows the time cost of the retrieval algorithm on a serial architecture with 16MB of main memory, for single-word queries ($c = 1$). The transfer rate was adjusted to 10Mbits/sec and the t_{op} was adjusted for a 20 MIPS machine like a Sparcstation. The database size ranges from 64MB (16,000 documents) to 64GB (16 million documents). The signature design parameters were optimized for each database size. The results show that a Sparc-class machine with 16MB of main memory devoted to the application can achieve a 2sec response time for a 5GB database. For such a configuration, the design was optimized for signature length $F =$ 4096 bits and $m = 6$ bits set per term. Such light-weight signatures keep the

false drop probability low. In the serial machine case, the most significant factor in the cost formula is the CPU cost for signature matching.

6 Conclusions

In this paper we have studied the application of a full text retrieval method based on Bit-Sliced Signature Files on a SIMD machine architecture with limited main memory. The contributions of the paper are the following:

- We have proposed a Parallel BSSF (PBSSF) method which uses *partial fetch* slice swapping for full text retrieval on large text databases.
- We have derived a formula (equation (2)) for the optimal number of signature slices to be fetched.
- We have provided an analytic performance evaluation of the proposed retrieval algorithm for various system and database parameters.

The results demonstrate that a SIMD architecture is well suited to a full-text database retrieval application. In particular, we derived the following conclusions:

- The retrieval method described exhibits satisfactory interactive performance (less than 2sec) for databases of size up to 128GB (32 million documents) on a system with 64K processors with 64K bits memory each, assuming no more than 12.5% storage overhead for the signature file and timing characteristics similar to a Connection Machine CM-2 system (figure 4).
- The method behaves well in a growing database environment, causing graceful performance degradation when the database size increases, even if the database grows much beyond the initial expectations of the designer.
- Long and light-weight document signatures (i.e. large F) are in general beneficial for the Parallel Bit-Sliced Signature File method, especially for large databases. This allows the false drop probability to be very low.
- Consequently, the signature slice I/O is normally the dominating factor in the total cost, compared with the false drop elimination and CPU costs. Therefore a partial fetch policy greatly improves the performance of the retrieval method compared with the total fetch policy (figure 3).

Moreover, the partial fetch method is also useful for a uniprocessor system. This policy helps a serial processor benefit from existing large main memories, which is the trend in commercial workstations. A response time of 2sec can be expected from a Sparc-like architecture with a main memory buffer of 16MB, operating on a 5GB database (figure 6).

Future research could examine (a) the effects of skewness in the frequencies of query terms (e.g. Zipf distribution [Zip49]), (b) the parallelization of vertical [LF92] and horizontal [SD83, LL89] partitioning signature methods, and (c) the parallelization of hybrid methods that combine signature retrieval with inverted indices [FJ91].

References

[CF84] Stavros Christodoulakis and Christos Faloutsos. Design Considerations for a Message File Server. *IEEE Transactions on Software Engineering*, 10(2):201–210, March 1984.

[Fal90] Christos Faloutsos. Signature-Based Text Retrieval Methods: A Survey. *IEEE Data Engineering*, pages 25–32, March 1990.

[FC87] Christos Faloutsos and Stavros Christodoulakis. Description and Performance Analysis of Signature File Methods for Office Filing. *ACM Transactions on Office Information Systems*, 5(3):237–257, July 1987.

[FC88] Christos Faloutsos and Raphael Chan. Fast Text Access Methods for Optical Disks: Designs and Performance Comparison. In *Proceedings of the 14th International Conference on Very Large Databases*, pages 280–293, Long Beach, California, August 1988.

[FJ91] Christos Faloutsos and H. V. Jagadish. Hybrid Index Organizations for Text Databases. Technical Report UMIACS-TR-91-33 and CS-TR-2621, Department of Computer Science, University of Maryland, March 1991.

[Has81] R. Haskin. Special-Purpose Processors for Text Retrieval. *Database Engineering*, 4(1):16–29, September 1981.

[LF92] Zheng Lin and Christos Faloutsos. Frame Sliced Signature Files. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):158–180, June 1992. Also available as UMD CS-TR-2146 and UMIACS-TR-88-88.

[Lin92] Zheng Lin. CAT: An Execution Model for Concurrent Full Text Search. In *PDIS*, 1992.

[LL89] D. L. Lee and C. W. Leng. Partitioned Signature File: Designs and Performance Evaluation. *ACM Transactions on Office Information Systems*, 7(2):158–180, April 1989.

[Pan92] George Panagopoulos. Bit-Sliced Signature Files for Very Large Databases on a Parallel Machine Architecture. Technical Report CSC-809, Department of Computer Science, University of Maryland, April 1992.

[SD83] Ron Sacks-Davis. Two Level Superimposed Coding Scheme for Partial Match Retrieval. *Information Systems*, 8(4):273–280, 1983.

[SM83] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw–Hill, 1983.

[Sta88] Craig Stanfill. Parallel Computing for Information Retrieval: Recent Developments. Technical Report DR88-1, Thinking Machines Corporation, Cambridge, Mass., January 1988.

[Sti60] Simon Stiassny. Mathematical Analysis of Various Superimposed Coding Methods. *American Documentation*, 11(2):155–169, February 1960.

[Sto87] Harold S. Stone. Parallel Querying of Large Databases: A Case Study. *IEEE Computer*, 20(10):11–21, October 1987.

[TC83] D. Tsichritzis and S. Christodoulakis. Message Files. *ACM Transactions on Office Information Systems*, 1(1):88–98, January 1983.

[Thi89] Thinking Machines Corporation, Cambridge, Mass. *Parallel Instruction Set, Version 5.2*, October 1989.

[Zip49] G. K. Zipf. *Human Behavior and Principle of Least Effort: An Introduction to Human Ecology.* Addison–Wesley, Cambridge, MA, 1949.