Information Technology: Research and Development (1983), *2* (73–88)

A PROPOSAL FOR AN ASSOCIATIVE FILE STORE WITH RUN-TIME INDEXING. PART I: SYSTEM DESCRIPTION

E. J. SCHUEGRAF

Department of Mathematics and Computing Sciences, St. Francis Xavier University, Antigonish, Nova Scotia, Canada B2G 1C0

AND

R. M. LEA

Department of Electrical Engineering and Electronics, Brunel University, Uxbridge, Middlesex, UK

(Received 9 June 1982, revised 2 February 1983)

ABSTRACT

Information retrieval systems are described and their relation to associative processing is established. An overview of present associative hardware is given, followed by a proposal for an Associative File Store (AFS) for online information retrieval systems. Its advantages over conventional file stores in maxi-, mini- and microcomputer systems are outlined. The AFS incorporates a microprogrammed Associative Processor. It is dedicated to high-speed fragment processing, to support file compression, run-time indexing and calculations of storage addresses for records, and file searching. Performance estimates indicate that compared with conventional file stores the AFS could achieve a speed advantage of over two orders of magnitude. Use of fragments for compression approximately doubles the storage capacity; the use of fragments for run-time indexing reduces the volume of data to be searched, but it retains the full flexibility of free text retrieval on variable-length records.

1. INTRODUCTION

1.1 Overview

The last decade has seen a phenomenal growth in the development of computer based retrieval systems for a wide spectrum of applications. Hand in hand with this development came the creation of databases of enormous size, covering various fields ranging from fairly general topics such as ERIC or INDEX MEDICUS to very specialized ones such as TOXLINE.

Some databases are in the public domain and access privileges can easily be obtained, while others, generally smaller ones, are private and access may be

0144-817X/83/02 0073-16 $03.00

severely restricted. Most datafiles found in business systems tend to fall in the latter category.

In document retrieval systems applications can be divided into two groups. Retrospective search systems inform the user of past references to journal and conference papers which are relevant to a particular field of specialization. Current awareness systems keep the user up to date by periodic releases of the newest references relevant to his research interests.

Datafiles of information retrieval systems usually consist of a collection of records. A collection may be ordered or unordered. A record may be a description of a single document and a reference to where it may be found, or it may be the document itself. In a document retrieval system a record normally contains author, title, keywords, abstract and citation. Records found in an office automation system may be entire letters, possibly characterized by some content descriptors.

Operations on these datafiles include additions and deletions of records as well as record modifications. Common to these activities is a search of the database, or parts thereof, for a specified record.

A set of search terms connected by logical operators is commonly called a query. A set of queries is called a profile. Considerable variety exists between retrieval systems with regard to construction of queries. Search terms may be keywords or content descriptors, words or word phrases or character strings of variable length. Words may include truncation symbols matching various endings. Many systems also permit search terms to contain don't care symbols, which match any character. The operators connecting the search terms may be regular Boolean operators, AND, OR, NOT, but also proximity operators such as adjacency and others. Additional refinements may include weights to be attached to search terms.

A query is satisfied if the search terms in the query can be found in the record and the logic expression is valid, or if the sum of the weights exceeds a previously established threshold. Sophisticated retrieval systems order the relevant records according to the sum of the weights before giving it to the user.

It is apparent that information retrieval is a form of set processing, in which a subset of records associated with particular search terms is selected by matching the textual content of a query with all records. Thus retrieval is an example of associative processing and a retrieval system simulates an associative processor. Ideally, therefore, any information retrieval system should incorporate an associative store, which should have a basic structure like the one shown in Figure 1.

1.2 Associative store

The centre of an associative store would be a Content Addressable Memory (CAM) with flexible access to its word rows and bit columns. In the ideal case a word row would contain a complete record and the CAM would be large enough to accommodate all records. A field mask would allocate bit fields of the CAM to key fields of the stored records. In operation, search keys held in the comparand register would be simultaneously compared with the contents of the corresponding key fields of all records in the CAM. Each bit of the CAM would contain logic to compare its content with the corresponding bit in the comparand register and generate the appropriate match or mismatch signal; masked bits would generate a match signal. If all bits in a particular word row generate match signals, then that word row would be tagged in the Tag Register. A Match Reply (MR), which indicates the presence of one or more tags in the tag register, provides feedback on the success of the

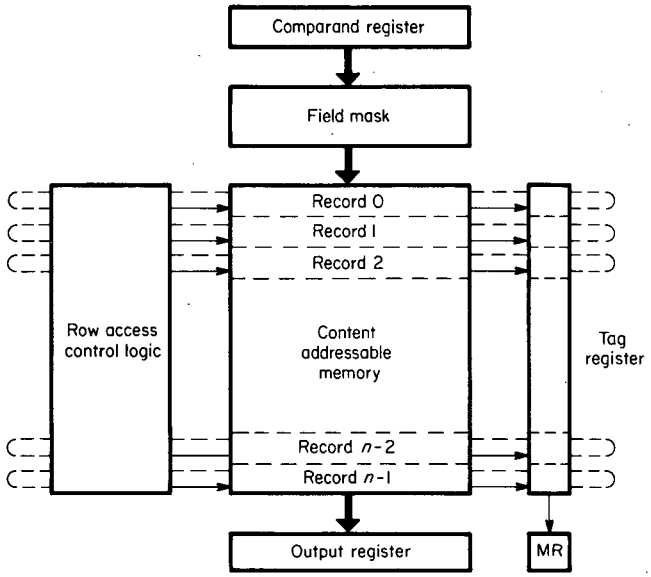

FIG. 1. Ideal associative store

associative search. Thus in a single operation the subset of matching records would be indicated by the content of the tag register. Tagged word rows can be activated for Read or Write operations by the Row Access Control Logic. Retrieval involves reading the content of an activated word row into the output register and updates involve writing the content of the unmasked fields of the comparand register into activated word rows. In summary the ideal associative store is a Single-Instruction Multiple-Data stream machine (Flynn, 1972). Overall control of the information retrieval system would be entrusted to a conventional Single-Instruction Single-Data stream (SISD) processor.

Implementation of associative stores has been of interest for the last 25 years. Many designs for structures such as that shown in Figure 1 have been proposed and several associative machines have been built for mainly military applications. Details about the historical development, and the many architectural variations of associative processors can be found in the literature (Hanlon, 1966; Minker, 1971; Parhami, 1973; Thurber, 1975; Higbie, 1976; Yau and Fung, 1977). Best known are the Goodyear STARAN (Rudolph, 1972), Bell Laboratories PEPE (Crane *et al.*, 1972) and Honeywell's ECAM (Anderson and Kain, 1976).

The feasibility of applying such machines to information retrieval and database management has been investigated by many researchers. Prominent among these studies has been the work of Savitt *et al.* (1967), De Fiore and Berra (1973, 1974), Lea (1977a) and Moulder (1973) with the STARAN; Linde *et al.* (1973) with the APCS and Anderson and Kain (1976) with the ECAM.

Emerging from this wave of endeavour is the growing realization that the ideal associative store as shown in Figure 1 does not suit present information retrieval applications for two reasons.

1. The sizes of many databases exceed the storage capacity of any realistically sized CAM.

2. Despite considerable efforts to develop cheap building blocks for associative hardware Lea (1977a), it is evident that large-scale associative stores are not cost effective for information retrieval applications.

The lack of cost-effective associative hardware has forced the designers of retrieval systems to make use of inappropriate but available SISD hardware.

1.3 Current retrieval systems

For current awareness schemes a cost-effective retrieval system can be achieved by a sequential scan of a datafile stored on disk or tape. Each search term in an ordered batch of profiles stored in the primary memory of the search processor is compared with the proper fields of every record in the datafile. The use of sophisticated term-matching algorithms, such as those described by Salton (1980), permit very rapid processing of records. Such SDI services are provided for example by CAN/SDI and the SDI services of INSPEC and UKCIS.

In the ideal case, the speed of the search processor and the data transmission rate from the device would be synchronized, such that all comparisons could be executed by a single scan of the datafile. The search processor acts as a filter for the subset of matching records from the data stream 'flying' by the heads of the disk or tape drive. Such systems are well suited to services in which datafiles are small, the number of queries predetermined and static and immediate response is not required.

In contrast to SDI schemes the basic assumptions are quite different for retrospective retrieval. Databases are large, queries are one-shot affairs, but are often refined and resubmitted. Response time should not exceed users' patience. Such systems are usually implemented as remote access timesharing systems, commonly based on a large SISD computer providing access to a range of magnetic disk units. Examples include CAN/OLE, DIALOG and BLAISE.

'On the fly' retrieval is not feasible in this case for two reasons.

1. Datafiles are too large to be scanned sequentially within the limits of reasonable response time.
2. Batching of queries is possible but generally not feasible as they are likely to be submitted at different times and for only one search.

Instead the datafile is usually structured and an auxiliary file is created to provide direct access to the elements of the datafile. The index sequential, the inverted, and tree-structured file organizations are examples of this approach and have been shown to provide fast access.

Most auxiliary files define a mapping from a set of search terms to a set of lists containing record identifiers. Each search term is mapped into a list of identifiers of those records which contain the search term. Subsequent processing of these identifier lists, by taking intersections and unions produces a list of records satisfying the query. Thus, only relevant records in the datafile are being accessed. For online retrieval associative file searching is implemented by processing of the auxiliary file. This solves the problem of providing satisfactory response time, albeit at the expense of extra complexity. Creation and maintenance of auxiliary files incurs overhead in

1. Storage space. Extra storage is required for address pointers; in fact it is common for auxiliary files to become larger than the original data files (Cardenas, 1975).

2. Processing time. Auxiliary files necessitate additional accesses to secondary storage. This can become such a large problem, that special methods such as hashing are required to access them effectively.

1.4 Back-end processors and associative file stores

An indication of the difficulties encountered in retrieval and database management systems is given by the 90–10 rule which had been observed in experiments with large databases (Banerjee and Hsiao, 1978). It states that for very large databases, nine times as much irrelevant data as relevant data must be brought into main memory for processing. If a query contains Boolean expressions, the relevant data in main memory must be processed again to select the items relevant to the query. Here the 90–10 rule applies again.

To avoid these problems arising from the use of a SISD processor it has been suggested that specialized additional hardware be used to increase the processing power (Bullen and Mullen, 1972; Banerjee and Hsiao, 1978). In particular the relegation of searching activities to disk controllers would free the central processor from many file operations, avoid transmission delays and protocol processing normally required for datablock transfer. Such an approach would eliminate any application of the 90–10 rule, commonly observed in database management systems (Hsiao, 1980).

The resulting reduction in load for the central processor would free it for other activities. Moreover, if high-speed processing logic is incorporated in disk controllers, on-the-fly searching can be supported. Such special purpose machines are becoming known as back-end processors. They can support on-the-fly file searching by dedicating key-matching logic to the heads of the disk drive. Logic-per-track devices (Slotnick, 1970; Hsiao, 1980) provide very fast scans of entire disks whereas lower-cost devices share comparison logic between data channels.

However, the implementation of back-end processors does not have to be restricted to the application of SISD hardware. Since small and dedicated processors are required, SIMD hardware can be employed cost effectively. A small Associative Parallel Processor (APP) would be an ideal component for a device implementing associative file searching. An APP comprises an associative store, similar to that shown in Figure 1 operating under stored program control. To accomplish on-the-fly searching a small APP would store the set of search keys for comparison with the key fields of the records stored on disk. Such specialized back-end processors are becoming known as Associative File Stores (AFS). Several associative file stores have been reported in the literature (Parker, 1971; Healy *et al.*, 1972; Su *et al.*, 1975; Lin *et al.*, 1976). The first proposal was RAPID (Rotating Associative Processor for Information Dissemination) (Schuegraf and Lea, 1979); RAP (Relational Associative Processor) (Parhami, 1972) was specifically designed for the processing of relational databases; fairly prominent, because of its commercial availability, is the CAFS (Content Addressable File Store) system (Ozkarahan *et al.*, 1975) of ICL.

Other hardware especially designed for efficient text searching are the Associative File Processor (AFP) built by Operating Systems Inc., which is based on a patented hardware comparator unit. The success of this unit has produced an improved system, the High Speed Text Search System (HSTS). A description of the HSTS system may be found in Moore and Michels (1980). The newest addition to text-searching machines is the database engine developed by Textarcana, which claims to be capable of searching up to 8 million characters/s (Product Description, 1982).

It is the intention of this paper to add another proposal for an Associative File Store to those already mentioned. It differs substantially from previous proposals by not relying on hardware alone, but by combining it with software techniques.

2. ASSOCIATIVE FILE STORE—ORGANIZATION

2.1 Overview

The proposed associative file store is a back-end processor consisting of a magnetic disk unit with an APP incorporated within the disk controller. The overall structure is shown in Figure 2. As the design philosophy of the AFS can be applied to all computer systems ranging from maxis to micros, an AFS would include the necessary logic to present a standard interface to the host's hardware and operating system. The file store itself is partitioned into a number of equally sized units, called 'scan blocks', each of which can be individually accessed and independently scanned. The records of a datafile, compressed or uncompressed, are distributed over a number of scan blocks and the APP is used as a search processor for on-the-fly retrieval.

2.2 Scan blocks

The smallest addressable segment in the AFS is a scan block. Thus, each access to the AFS involves scanning the entire content of the addressed scan block. To

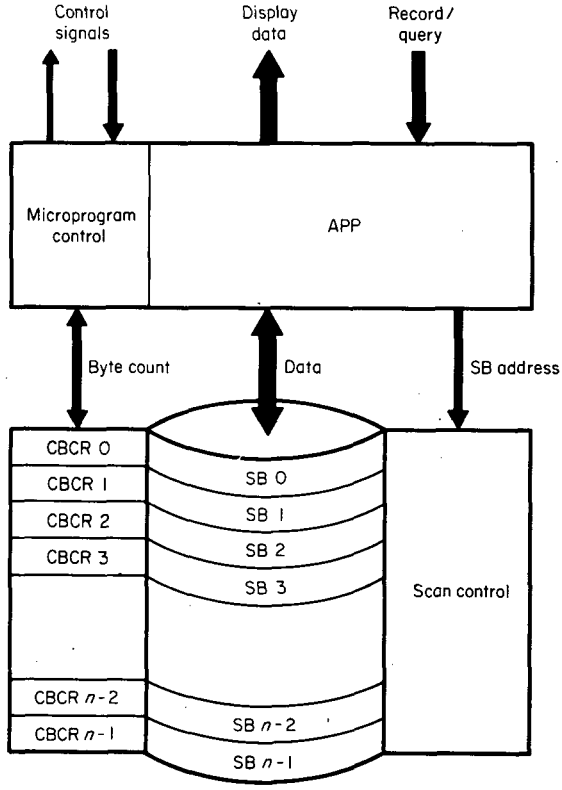

FIG. 2. Proposed associative file store

facilitate the scanning each scan block has an associated register CBCR (Current Byte Count Register) which records the total number of allocated bytes in that scan block. This is shown in Figure 2.

To provide an idea of the volume of scanned data it is helpful to compare it with the data segments of existing direct-access storage units. Table 1 compares the amount of data accessed within the time for a single direct-access file probe, for three commonly available moving head disk drives. These are:

1. Maxicomputer system: a 12 disk pack.
2. Minicomputer system: a single disk.
3. Microcomputer system: an 8 in. single surface floppy.

It has been assumed that a direct-access file probe involves two disk accesses; the first to consult the directory of track and sector addresses and the second to read the addressed sector. Table 1 shows that, in terms of the volume of accessed data, the advantages of scanning over direct access are 58:1 for the maxi-, 42:1 for the mini-, and 112:1 for the microcomputer system. It should be noted that these are conservative figures; indexing restrictions are likely to necessitate several direct accesses per file probe. The availability of higher performance disk drives is increasing the advantage of scanning. A further point in favour of scanning is that direct-access disk sectors usually include 'address headers' which reduce the storage available for useful data. This redundancy increases with decreasing sector size, such that the utilization factor of floppy disks may be less than 70 per cent. Scanning reduces the need for address headers, thereby achieving disk utilization factors much closer to 100 per cent.

Table 1. Comparison of scanning and direct-access file probes for commonly available computer systems

Computer system	*Time for one direct-access file probe**	*Sector size (in bytes)*	*Volume of data scanned in one direct-access file probe**
Maxi	79 ms	1024	58 KB
Mini	186 ms	512	21 KB
Micro	641 ms	128	15 KB

* Assuming 2 disk accesses with average seek times and latency delays.

Scan-block size is determined by compromising the

1. Scan time.
2. Number of scan blocks per disk surface.
3. Total number of scan blocks within the AFS.

For convenient scan-block addressing the number of scan blocks per surface and the total number of scan blocks should be powers of two. However, the major factor in limiting scan-block size is the requirement that the response to a user's query should be displayed within an acceptable period. Table 2 compares possible scan-block sizes

with the three commonly available moving-head disk drives for a response time of one second. The table also shows the total time required to scan all the scan blocks with the given disk drive.

Table 2. Comparison of possible scan blocks for a response time of 1 s with commonly available computer systems

Computer system	*Scan block size (in kbytes)*	*Number of tracks per scan block*	*Number of scan blocks per surface*	*Total number of scan blocks*	*Total scan time for all scan blocks**
Maxi	652	50	8	128	7 s
Mini	74	12	16	32	11 s
Micro	16	4	16	16	11 s

* Assuming all heads are sensed in parallel.

2.3 Data format

In keeping with current practice the scan blocks would be partitioned in sectors, each comprising bit synchronization, data and error-check fields; scanning eliminates the need for sector address fields. Successive datafields would be allocated for the storage of datafile records. As usual, error control considerations would govern the choice of sector size. In the event of error detection the scan would be restarted.

Scanning allows variable-length records, separated by a special 'EOR' (End-of-Record) character to be filed in a sequence which would be terminated by a special 'EOS' (End-of-Sequence) character. A new record would be added at the end of the sequence. Such random order within a block is permissible since scanning the entire block eliminates the need for sorting. The data records themselves may be in their original form or compressed. Data compression offers two advantages, namely, a considerable saving in storage space and a reduction in search time.

2.4 Record signatures

The user of an online retrieval system sitting at a terminal expects a reasonable response time, especially during query refinement. The final query is expected to provide high recall as well as high precision (Salton, 1968). Only in a few cases, such as the final query, will a user tolerate a long response time. If the records matching the query were uniformly distributed over all the scan blocks, then it would be necessary to scan all blocks to find the relevant records. Response time would be the worst scan time as indicated in Table 2.

Record signatures are an attempt to distribute records among scan blocks in such a way that the distribution of records matching a query is non-uniform over all scan blocks. Signatures are to indicate those scan blocks for which the hit rate could be expected to be high, as well as those scan blocks which will not contain any hits. Any reduction in the number of scan blocks which must be scanned to answer a query will improve response time.

In analogy to a human signature which is characteristic of the writer, record signatures characterize the content of the record. This can be achieved by choosing a set of content indicators, enumerating its elements, and creating a bit map for each record. This binary vector, where a 0 or 1 in a particular position indicates the

absence or presence of that particular element, may be called its signature (Harrison, 1971). A record is assigned to a scan block on the basis of its signature. Records with 'similar' signatures would be assigned to the same scan block.

If the content indicators are the keywords, then the signature is called a document vector, and the assignment of records with similar signatures to the same scan block is a form of clustering (Salton, 1968). The procedure which assigns a record with a given signature to a specific scan block may be called a signature-mapping function. It is obvious that many signature-mapping functions exist, but that the restriction of mapping 'similar' signatures to the same scan block severely limits the choice of functions.

3. ASSOCIATIVE FILE STORE—OPERATION

3.1 Basic assumptions

A selection of what elements of a record could be used as content indicators must be made before the operations of the AFS can be described in more detail. Choosing keywords as content indicators generates a document vector with many components, most of which are zero as a consequence of Zipf's law. The number of components in the vector is the same as the number of keywords in the keyword set, a number that may be unlimited. This effectively eliminates keywords as signature elements in the AFS, because of the difficulties encountered when manipulating large binary vectors.

Restricting the size of the content indicator set excludes the use of natural linguistic units such as words, word stems or phrases. Instead, artificial elements, namely fragments, are chosen as signature elements. Fragments or *n*-grams are character strings of arbitrary length completely contained in a record. A set of fragments, commonly called a dictionary, can be limited to any convenient size. Algorithms for choosing a dictionary under varying constraints have been studied extensively (Clare *et al.*, 1972; Schuegraf and Heaps, 1973; Choueka *et al.*, 1981). It has been shown that dictionaries are fairly stable with regard to vocabulary changes (Lynch *et al.*, 1973) and that considerable overlap exists between dictionaries of different languages (Doucette *et al.*, 1977).

It has been shown that fragments can serve in two roles, as language elements for compression (Schuegraf and Heaps, 1974) and as indexing elements for retrieval (Schuegraf and Heaps, 1976). If the character set is included in the dictionary, then every record can be represented as a concatenation of fragments. Considerable savings in storage space, up to 50 per cent, can be realized if a code is substituted for a fragment. It has been shown (Lea, 1978) that a small associative processor using a fragment dictionary for compression and decompression is cost effective and can carry out compression and decompression on the fly. Thus, the use of fragments as signature elements in the AFS provides the option of employing the built-in APP for data compression.

When inspecting a fragment dictionary generated for compression purposes, it is noted immediately that the dictionary not only contains all single characters, but such strings as 'AND-THE', 'OF-A-' and the like. These elements are obviously not content indicators and, therefore, it is necessary to select a subset from the elements of the dictionary. This subset of content fragments will be used to generate the record signature, as shown in Figure 3. The signature is a bit vector in which each content fragment is allocated a fixed position. The length of the bit vector is determined by the number of content fragments found in the dictionary. The

presence or absence of a specific content fragment in the record is indicated by a one or zero in the corresponding position. In the following two sections it is described how a file is created on the AFS and how the data are retrieved. For simplicity of description it is assumed that only one datafile is to be supported by the AFS.

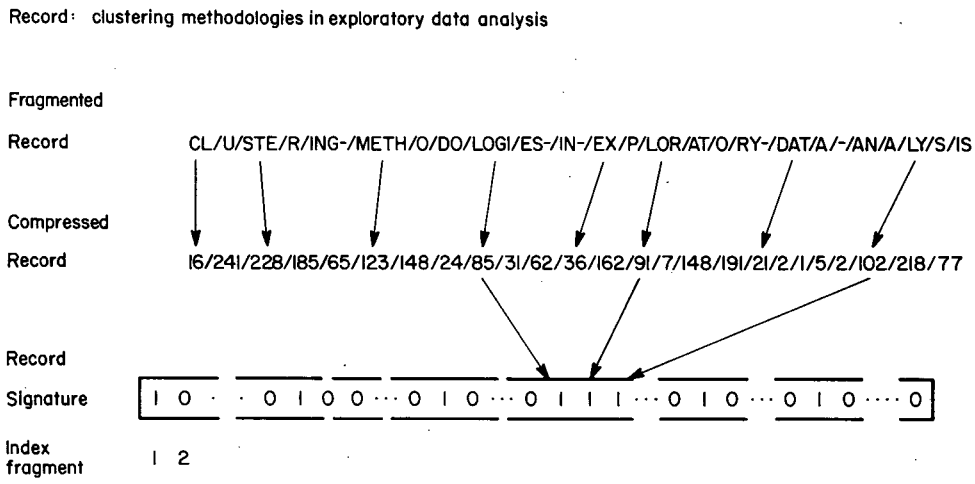

FIG. 3. Sample record and signature

3.2 Datafile creation

The following sequence of steps describes the creation of a datafile with AFS. It is assumed that the datafile will be stored in compressed form, since this can be accomplished without extra expense.

SETUP:

1. Reset the AFS by writing EOS terminator to the first locations of all scan blocks and clear the contents of their CBCRs. Clear record signature.

LOAD:

2. Compress a record to generate a contiguous list of fragment codes and simultaneously count the number of bytes in the compressed record.
3. Generate the record signature by setting the appropriate bit when encountering a content fragment during compression.
4. Apply the signature-mapping function to compute the address of the most suitable scan block for its storage. Set a flag if the mapping was successful, otherwise clear it.
5. If the flag was not set
   then all scan blocks are full, abort datafile creation and inform the user
   else read the content of the CBCR of the addressed scan block
   If the value of the CBCR and the byte count of the compressed record exceeds the size of the scan block
   then return to Step 4
   else write the EOR separator followed by the compressed record and the EOS terminator to the end of the addressed scan block and simultaneously update the corresponding CBCR. Return to Step 2.

The loading process is repeated until there are no more records to be added to the datafile or the AFS is full. During datafile creation, scan blocks are not scanned from their start points for every write operation, but the track currently supporting the EOS terminator is determined (from the content of the CBCR) and this track is addressed directly. It should be pointed out that the repeated application of Step 4 on the same signature will generate a sequence of suitable scan-block addresses for that signature. If the mapping function cannot generate any more suitable scan-block addresses, the scan blocks are full and the mapping-successful-flag will not be set. The mapping function applied to a given signature must generate the address of a suitable scan block only once.

3.3 Retrieval

The following sequence outlines the process of retrieval with the AFS. It is assumed that one datafile on the AFS exists loaded by the procedure described in 3.2.

SETUP:
1. Compress the search query and generate a query signature.

SEARCH:
2. Apply the signature mapping function to the signature to compute the address of a scan block containing potentially matching records. Set a flag if the mapping was successful, otherwise clear the flag.
3. If the flag was set
   then scan the addressed scan block, comparing the fragments of the compressed records with the fragments of the query, storing the count of EOR separators for each matching record and simultaneously counting the number of hits until the EOS terminator is sensed. Display the hit count.
   If the hit count exceeds user-defined limit
   then terminate search
   else return to Step 2
   else the sequence of scan blocks is exhausted
   all scan blocks worth scanning have been searched,
   the search is finished.
   Display the hit count.

It is worth mentioning that the count of EOR separators for a matching record is a convenient way of identifying the position of a record within a scan block. If the user requests a display of the results of the search the above procedure is repeated with the following modifications.

DISPLAY:
2. Same as in Step 2 of SEARCH.
3. If the flag was set
   then scan the addressed block extracting and decompressing those records corresponding to stored EOR counts.
   Display user selected fields of matching records
   If list of EOR counts is exhausted
   then terminate DISPLAY
   else return to Step 2
   else DISPLAY finished.

Since the display of information to the user is controlled by the main processor independently of the AFS, any display during retrieval can be carried out in parallel with scan-block searching. A user can search for only a few relevant records by choosing a small value for the hit count. In that case only those scan blocks are searched which have the highest probability of containing a record. By choosing a large number for the hit count, all relevant records may be retrieved. In this case the AFS will search a subset of the database, consisting of all scan blocks which have a non-zero probability of containing a relevant record. This subset can be easily determined and even in the case of a single term query, will not be more than half the database.

4. PRACTICAL CONSIDERATIONS

4.1 Role and status of the APP

As seen from Figure 2 the Associative Parallel Processor (APP) forms a major component of the AFS distinguishing it from ordinary file stores. Operating under microprogrammed control the APP has three roles, namely, to act as

1. Compressor/Decompressor Unit.
2. Signature Processor.
3. Search Processor.

In each role the associative memory of the APP stores the fragment dictionary. During processing the currently active fragments are marked by 'tag images'. Tag images are one bit memory fields representing snapshots of the tag register. Hence, the APP is mainly occupied with the generation and manipulation of tag images, an application to which it is particularly well suited. Algorithms for text compression/decompression and text searching with the APP are well known and may be found in Lea (1977b, 1978).

Ideally the AFS would be implemented as a single unit, interfacing to a standard communication bus. However, for mini- and microcomputer-based retrieval systems the microprogrammed APP and byte-count registers could be implemented independently of the disk unit, in a separate, bus-compatible module. In this case the module, operating in parallel with the host computer, would control direct-memory access data transfers between the APP and the disk unit. An advantage of this approach is that one or more APP modules could be shared between a number of disk units and other text-processing applications.

The associative processing group at Brunel University has been engaged in research into the architecture software and hardware of APP structures since 1972. Logic and language specifications have been established and machine implementations are being studied. In particular the feasibility of microelectronic APP building blocks has been investigated resulting in the production of 'micro associative processor chips' by Plessey Research Ltd (Lea, 1979). Current results suggest that high-speed APPs could be implemented at low cost.

4.2 Potential benefits of the AFS

Text compression, automatic indexing, clustering, and associative file searching have been the subjects of numerous research investigations making specific contributions to the science of information storage and retrieval. The unique

contribution of the proposed associative file store is that it combines the virtues of those independent areas of interest within a simple machine.

Many studies have demonstrated that associative file searching can improve retrieval performance by up to two orders of magnitude. Indeed, a recent study at Brunel University indicated that a small APP would out-perform a microprocessor-based search processor by a factor of 200 in flexible substring searching applications. Significantly, the APP could operate at typical disk speeds whereas the microprocessor system was too slow and expensive buffering would be required for it to support on-the-fly file searching. In fact, the APP-based search processor turned out to be cheaper and much easier to program than its conventional counterpart.

Text compression has been of interest for many years (Schuegraf, 1976). Several studies have shown that, with the use of 8 bit bytes to represent the 50 or so characters and approximately 200 of the most frequent fragments supporting typical databases, compression ratios of around 50 per cent can be achieved (Clare *et al.*, 1972; Schuegraf and Heaps, 1974; Lea, 1978). In terms of the AFS this means that text compression can double the number of records stored in a scan block and double the number of records scanned in a given time period. Fewer data exist on the use of fragments for indexing.

A few studies have indicated the benefits of this approach and the special problems arising (Clare *et al.*, 1972; Lynch *et al.*, 1973; Schuegraf and Heaps, 1976; Schuegraf and Lea, 1979). It was shown that in a system indexed by fragments it is necessary to scan records retrieved by a fragment index file (Schuegraf and Heaps, 1976). This is necessitated by the unfortunate, but unavoidable feature that a fragment may be contained in more than one word. This implies that in such a system, negated query terms must be ignored, because more than the negated term would be excluded. In addition, it is possible that the fragments comprising a search term may be found in a record, but as parts of other terms. In order to eliminate the potentially large number of false hits, a scan of the 'candidate' records for the full query is necessary. In a system with a fragment index file, this requires additional effort, but in the AFS, scanning is automatically carried out by the APP when a scan block is accessed. In addition, this feature does not put restrictions on the query format, since scanning can easily accommodate different query forms. A special index file is not needed at all; the signature provides a vector of index fragments generated at run time. The signature-mapping function eliminates the auxiliary files and includes a form of clustering of 'similar' signatures. It has been shown that record signatures can work satisfactorily in non-numeric applications (Harrison, 1971; MacLaury, 1979), even though the data are scarce.

In certain applications the potential benefits of the AFS would have to be analysed carefully with regard to its cost effectiveness. However, the simplicity of the design suggests that an AFS could be built at low cost. To determine the cost effectiveness of the AFS is a long-term project, since research vehicles must be developed which support comparative measurements of performance parameters for the various sized computer systems.

5. CONCLUSION

The proposed associative file store as described in the previous sections offers a solution to current problems encountered by online information retrieval systems. By supporting on-the-fly file searching the associative parallel processor avoids the complexities and restrictions normally associated with structured datafiles. The

indexing problem of directly accessing single records has been reduced to indicating the most suitable scan block for a high recall of matching records. Use of fragments not only provides the storage and transmission benefits of file compression, but also provides a means of run-time indexing which eliminates the need for an auxiliary file. Run-time indexing and mapping of signatures to scan block addresses make this proposal significantly different from other ideas found in the literature. It had been suggested that for fast searching it would be beneficial to set up a special file of 'keys' (Hickey, 1977; Roberts, 1979). The 'keys' are to be content indicators for the record, but considerably smaller. To carry out a search, this file is first to be scanned to determine possible candidate records which could satisfy the query. All candidate records are then retrieved in full and subjected to a proper scan. It is interesting to note that Roberts (1979) produces a bit vector which is reduced in size by the method of superimposed codes (Moers, 1951). To speed up the search of the content-indicator file it is suggested that specialized hardware—a new type of associative memory—would be beneficial.

It must be mentioned that considerable work has yet to be carried out before the design for the AFS is shown to be feasible. Some preliminary results on this have been presented (Lea and Schuegraf, 1981). Even though the associative parallel processor has been developed there is still a need to design suitable hardware for the AFS. In addition, the software component of the AFS must be studied in detail, since there is much scope for innovation in dictionary selection, signature generation, and particularly signature-mapping functions. Work in this area is currently underway, leading up to the simulation of an associative file store (AFS). Results of completed software experiments dealing with signature generation and mapping are encouraging, and will be reported in a forthcoming paper (Schuegraf and Lea, in preparation).

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the Natural Science and Engineering Research Council of Canada, and the Science Research Council and British Library. The contacts and discussions with Professors M. F. Lynch, D. Cooper and P. Willett of Sheffield University are also warmly appreciated.

REFERENCES

Anderson, G. A. and Kain, R. Y. (1976) A content addressed memory design for data base applications. *IEEE conference on parallel processing*, pp. 191–195.

Banerjee, J. and Hsiao, D. K. (1978) Performance evaluation of a data base computer in supporting relational data bases. *Proceedings international conference on very large data bases.*

Bullen, R. H. and Mullen, J. L. (1972) Microtext: the design of a microprogrammed finite state search machine for full text retrieval. *Proceedings FJCC 41*, 479–488.

Canaday, R. H., Harrison, R. D., Ivie, E. L., Ryder, J. L. and Wehr, L. A. (1974) A backend computer for data base management. *Communications of the ACM 17*, 53–56.

Cardenas, A. F. (1975) Analysis and performance of inverted data base structures. *Communications of the ACM 18*, 253–263.

Choueka, Y., Fraenkel, A. S. and Perl, Y. (1981) *Polynomial construction of optimal prefix tables for text compression.* Rehovot, Israel: Weizmann Institute of Science (Report CS81-06).

Clare, A. C., Cook, E. M. and Lynch, M. F. (1972) The identification of variable length, equifrequent, character strings in a natural language data base. *Computer Journal 15*, 259–262.

Crane, B. A., Gilmartin, M. J., Huttenhoff, T. H., Rux, P. T. and Shively, R. R. (1972) PEPE computer architecture. *Proceedings IEEE CompCon*, pp. 57–60.

De Fiore, C. R. and Berra, P. B. (1973) A data management system utilising an associative memory. *Proceedings national computer conference 42*, 181–185.

De Fiore, C. R. and Berra, P. B. (1974) A quantitative analysis of the utilization of associative memories in data management. *IEEE Transactions. Computers C–23 2*, pp. 121–133.

De Fiore, C. R., Stillman, N. J. and Berra, P. B. (1971) Associative techniques in the solution of data management problems. *Proceedings of the ACM annual conference*, pp. 28–36.

Doucette, V. L., Harrison, K. M. and Schuegraf, E. J. (1977) A comparative evaluation of fragment dictionaries for the compression of French, English and German bibliographic data bases. *Proceedings 3rd international conference on computing in the humanities*, pp. 297–305.

Flynn, M. T. (1972) Some computer organizations and their effectiveness. *IEEE Transactions. Computers C–21*, 948–960.

Hanlon, A. E. (1966) Content addressable and associative memory systems—A survey. *IEEE Transactions. EC–15*, 509–521.

Harrison, M. C. (1971) Implementation of the substring test by hashing. *Communications of the ACM 14*, 777–779.

Healy, L. D., Lipovski, G. T. and Doty, K. L. (1972) The architecture of a content addressed segment-sequential storage. *Proceedings FJCC 41*, 691–701.

Hickey, T. (1977) Searching linear files on-line. *On-Line Review 1*, 53–58.

Higbie, L. C. (1976) Associative processors: A panacea or specific? *Computer Design July*, 75–86.

Hsiao, D. K. (1980) Data base computers. *Advances in Computers.* (M. C. Yovits, ed.) Vol. 19, pp. 1–64.

Lea, R. M. (1977a) Micro-APP—Building blocks for low-cost high-speed associative parallel processing. *Radio and Electrical Engineering 47*, 91–99.

Lea, R. M. (1977b) Associative processing of non-numerical information. *Proceedings NATO ASI C–32*, 171–215.

Lea, R. M. (1978) Text compression with an associative parallel processor. *Computer Journal 21*, 45–46.

Lea, R. M. (1979) I^2L micro-associative-processors. *ESSIRC 79, Technical Digest*, 104–106.

Lea, R. M. and Schuegraf, E. J. (1981) An associative file store using fragments for run-time indexing and compression. *Proceedings symposium on future directions in information retrieval.* pp. 321–335. London: Butterworths.

Lin, S. C., Smith, D. C. and Smith, J. M. (1976) The design of a rotating associative memory for relational data-base applications. *ACM Transactions on Data Base Systems 1*, 1–8.

Linde, R. R., Gates, R. and Peng, T. (1973) Associative processor applications to real-time data management. *Proceedings national computer conference 42*, 187–195.

Lynch, M. F., Petrie, T. H. and Snell, M. J. (1973) The microstructure of titles in the INSPEC database. *Information Storage and Retrieval 9*, 331–337.

MacLaury, K. D. (1979) Automatic merging of bibliographic data bases—Use of fixed length keys derived from the strings. *Journal of Library Automation 12*, 143–156.

Maller, V. A. (1979) The content addressable file store—CAFS. *ICL Technical Journal.* pp. 265–279.

Minker, J. (1971) An overview of associative or content addressable memory systems and a KWIC index to the literature 1956–1970. *Computing Reviews 12*, 453–504.

Moers, C. N. (1951) Zato-coding applied to mechanical organization of knowledge. *American Documentation 2*, 20–32.

Moore, G. B. and Michels, L. S. (1980) OSI's high speed text search system: A hardware approach to full text searching. *Proceedings ASIS annual meeting 17*, 335–337.

Moulder, R. (1973) An implementation of a data management system on an associative processor. *Proceedings national computer conference 42*, 171–176.

Ozkarahan, E., Schuster, S. A. and Smith, K. C. (1975) RAP: An associative processor for data base management. *Proceedings national computer conference 44*, 379–387.

Parhami, B. (1972) A highly parallel computer system for information retrieval. *Proceedings FJCC 41*, 681–690.

Parhami, B. (1973) Associative memories and processors—An overview and selected bibliography. *Proceedings IEEE*, 722–730.

Parker, I. L. (1971) A logic-per-track retrieval system. *Proceedings IFIPS congress*, 146–150.

Product Description (1982) *Electronics April 7*, 176.

Roberts, C. S. (1979) Partial-match retrieval via the method of superimposed codes. *Proceedings IEEE 67*, 1624–1642.

Rudolph, J. A. (1972) A production implementation of an associative array processor—STARAN. *Proceedings FJCC*, 229–241.

Salton, G. A. (1968) *Automatic Information Organization and Retrieval.* New York: McGraw-Hill.

Salton, G. A. (1980) Automatic information retrieval. *Computer September*, 41–54.

Savitt, B. A., Love, H. H. and Troop, R. E. (1967) ASP—A new concept in language and machine organization. *Proceedings SJCC 30*, 87–102.

Schuegraf, E. J. (1976) A survey of data compression methods for non-numerical records. *Canadian Journal of Information Science 2*, 95–103.

Schuegraf, E. J. and Heaps, H. S. (1973) Selection of equifrequent fragments for information retrieval. *Information Storage and Retrieval 9*, 697–711.

Schuegraf, E. J. and Heaps, H. S. (1974) A comparison of algorithms for data base compression by use of fragments as language elements. *Information Storage and Retrieval 10*, 309–319.

Schuegraf, E. J. and Heaps, H. S. (1976) Query processing in a retrospective document retrieval system that uses word fragments as language elements. *Information Processing and Management 12*, 283–292.

Schuegraf, E. J. and Lea, R. M. (1979) An integrated approach to on-line retrieval from non-numeric data bases: An associative parallel processor using fragment indexing. *Proceedings conference on information sciences and systems, Baltimore, MD*, 395–401.

Schuegraf, E. J. and Lea, R. M. (in preparation) A proposal for an associative file store with run-time indexing. Part II: Experimental evaluation of software algorithms.

Slotnick, D. L. (1970) Logic—per-track devices. *Advances in Computers 10*, 291–296.

Su, S. Y., Copeland, G. P. and Lipovski, G. J. (1975) Retrieval operations and data representations in a content-addressed disk-system. *SIG PLAN Notices 10*, 144–153.

Thurber, K. J. (1975) Associative and parallel processors. *Computing Surveys 7*, 215–255.

Yau, S. S. and Fung, H. S. (1977) Associative processor architecture—A survey. *Computing Surveys 9*, 3–27.