ROTATING MEMORY PROCESSORS FOR THE MATCHING OF COMPLEX TEXTUAL PATTERNS

Lee A. Hollaar
Department of Computer Science
University of Illinois at Urbana-Champaign

Many people have suggested adding scanning logic to a rotating memory system, such as disk or shift registers, to allow faster execution of database operations. Most of these have been concerned with producing a form of associative memory which is then used to implement one or more of the models for information storage and retrieval, such as relational or hierarchical. While these are capable of searching for simple character strings, they are incapable of handling the complex patterns sometimes necessary for textual information retrieval. In addition, textual information retrieval does not lend itself to highly formatted databases, encoding of information, or arbitrary ordering of data, concepts common to the other structures.

A description of the operations desirable in textual information retrieval is given, and contrasted to those operations allowed in non-textual systems. The general structure for a scanning processor is presented, and a number of different trade-offs in its design and operation are discussed.

Introduction

As the cost of direct access storage has rapidly decreased due to advances in disk memory and semiconductor technologies, there has been a change in the utilization of computers. While early systems performed batch operations on sorted sequential files, contemporary ones operate interactively and directly retrieve and process the desired data. Data from a number of different applications can be collected and stored in a single database, with an accompanying increase in utility and decrease in storage requirements. Large collections of reference material can be updated rapidly and searched to find citations relevant to the user's needs.

Most of the work in the non-numeric processing of large amounts of data has been in the area of database management systems, with a number of abstract models, such as the relational and the hierarchical, available to describe the structure of the database. The operations of these systems include searching the database to retrieve all items that match a specified pattern, performing simple computations based on the data contained in the retrieved entries, and updating, adding, and deleting items from the database. The search criteria is generally quite simple. For example, in an inventory management application, it may consist of finding all parts manufactured by a given supplier between two dates.

Textual Information Processing

Another class of non-numeric processing of large collections of data is the handling of textual information. Two subclasses exist -- the transformation of character strings to another representation, such as done by an assembler, compiler, or typeset formatter; and the storage and retrieval of documents based on user requests. This retrieval can either be done on a rapidly changing database based on previously formulated queries (such as in a sellective dissemination of information system or a retrieval system where the user asks the same general questions), or on a static or increasing database using queries formulated interactively to best locate desired information. The latter case is an automated reference library system, and is useful, for instance, in legal research where previous court decisions are searched for precedents for a current case. It is on this last form of non-numeric processing that this paper will concentrate.

Because of the differences between a system used to retrieve and print relevant portions of a collection of test and a system used to maintain current data regarding applications such as inventory control, the operations available to the user differ greatly. For an automated reference library, the primary operation consists of searching the material for documents that contain specific words or phrases, and optionally printing those that satisfy the query. The user is not permitted to alter, add, or delete items from the database, much as he is not allowed to deface or remove books from a conventional reference library. (Of course, privileged instructions are available to allow the system management to change documents within the database, but these operations are far less common than the user operations.) Auxillary operations allow him to combine sets of documents found by searches using Boolean expressions, browse through documents instead of printing it in its entirety, and maintain a private file of annotations or comments regarding the various documents and sets of query results.

Unlike the database management structure, these textual databases are not heavily formatted. Delimiters exist that mark the beginning and end of documents, that are blocks of text that share a common idea, such as chapters of a book, articles in a magazine, or individual court decisions. Within the documents, it is convenient to replace typographical items with flags to allow the easier identification of various contexts. For example, the period at the end of a sentence can be replaced with a special end of sentence mark, to eliminate the ambiguity of the same character to end a sentence, indicate an abbreviation, or as a decimal point. Paragraphs can be marked instead of using a number of leading blanks on the first line to signal their start. Important subdivisions such as the title, author, abstract, and keywords can be flagged to allow faster searching.

In general, because the search is for items within a large block of text rather than in a fixed field, and because authors of different documents within the database do not use exactly the same terms to describe a subject, the search expressions are more complex than for database for textual retrieval, and is based on a proposed modification [1] of the query language of the EUREKA information retrieval system [2].

The various forms can be combined to form a complex search expression, such as:

BASSET HOUND# and (BRITAIN or ENGLAND) in sentence
and <(GRIMS or FOCHNO), KENNEL>5

that finds all documents that contain the phrase BASSET HOUND, or any suffix on the word HOUND, and BRITAIN or ENGLAND in the same sentence, and either GRIMS or FOCHNO within five words of the word KENNEL.

Useful database sizes range from under one billion characters for a rapidly changing system of current newspaper and magazine articles, to well over 25 billion for an automated legal research system that contains all the reported court decisions, along with auxillary material such as the state statutes and various regulations. This prevents the searching of the entire database by a conventional digital computer to find the documents that satisfy an interactive query, since even if it could process one character from the database every microsecond (unlikely even on the fastest available processors if the query consists of many search terms and operators), it would take approximately 2.8 hours to search a ten billion character file. This time can be reduced to a more reasonable value by using a partially inverted file structure [3] to eliminate those documents that obviously cannot satisfy the user's request. This is especially useful for the general queries initially entered in an interactive search. If a search is made only of the set of documents that satisfy an inverted file operation or matched in a previous query, it may involve searching less than one million characters. While this can take only a few seconds on a conventional processor, if the system services a large number of simultaneous users even this may result in unacceptable response times.

Specialized Processors

Due to the sequential nature of searching the documents for a given pattern, it is not necessary to store them in a totally direct access memory system, such as the primary memory of a conventional processor. Instead, a form of block accessible sequential memory can be used. This generally consists of some form of rotating memory, such as a disk file or a shift register, with a mechanism for rapidly positioning to a block within the memory. With a disk file, this positioning is provided by electrically selecting one of many heads or mechanically moving the heads to a different area on the disk. Currently available disk memory systems allow the storage of 300 million characters on a single drive, with the average time to access a block of less than 50 milliseconds. Shift register memories, implemented using CCD or bubble technology, can access a given block either by selecting one of many shift registers or by using a major/minor loop technique.

A	Finds any document which contains the word A.
A or B	Finds any document which contains either the word A or the word B.
A and B	Finds any document which contains both the word A and the word B anywhere in the document.
A and B in sent	Finds any document which contains both word A and word B in the same sentence. [specified context]
A B	Finds any document which contains the word A immediately followed by the word B. [finds a phrase]
A ... B	Finds any document which contains the word A followed (either immediately or after an arbitrary number of words) by the word B.
A .n. B	Finds any document which contains the word A followed by the word B within n words.
<A,B>n	Finds any document which contains the words A and B within n words of each other.
(A,B,C,D)%n	Finds any document which contains at least n of the different words A, B, C, or D. Note that if n=1, this operation is an OR, while if n equals the number of different words specified, it is an AND. [threshold OR]
a!b	Matches the character string a, followed by any single character, followed by the string b. [wildcard character]
a#b	Matches the character string a, followed by an arbitrary number of characters (possible none), followed by the string b. Note that #a matches a with any prefix, while a# matches a with any suffix.

Figure 1 - Typical Text Retrieval Operations

Many disk memory systems, particularly those for the IBM System/360, have facilities in their controller not only to position the access mechanism and transfer data to and from the central processor, but to perform a limited search on the data without intervention from the central processor. While this search capability is very primitive, and not extensively used, it provides a possible solution to the need for high speed scanning of data on a disk. A specialized processor can replace the existing disk controller to provide for the more complex operations common to textual information retrieval. This tehn forms a backend processor to the general purpose digital computer that is the information system's host [4].

Figure 2 illustrates the basic structure of such a backend system. Data is transferred from the disk memory system either to the channel of the host system, in a mode that makes the backend system compatible with other disk systems, or is examined by scanning logic. This logic compares characters as they are read from the disk against the specified search terms and checks for special delimiters within the text, such as end of sentence marks or flags indicating special contexts. Whenever a match is found, an indication is passed to logic that resolves whether a match has occured. If so, a pointer to the document is stored for later use in other queries or to retrieve the document for printing. To improve system response time, the scanning logic can be replicated and many searches performed in parallel.

This structure is similar to the logic per track device suggested by Slotnick [5], which forms a basis for database computers such as RAP [6], CASSM [7], RARES, [8], or

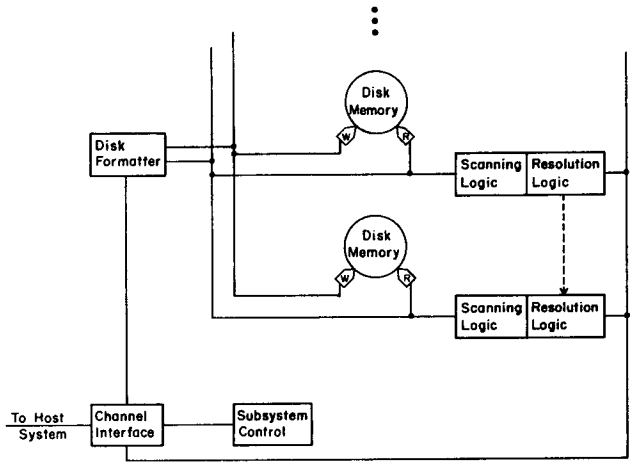

Figure 2 - Basic Structure

DBC [9]. However, each of these are designed to efficiently implement database management schemes instead of textual processing. They operate best on formatted data searched with simple expressions, and devote much of their logic to update and calculation capabilities unnecessary and undesirable for general textual retrieval.

Stellhorn [10] proposes a unit more suitable for text processing, but with a limited search capability. As originally proposed, it is capable of only determining whether one or more terms occur within a given context. It did not support more than one context in a search, or the matching of arbitrary characters or words. Bird [11] has described a backend processor for a PDP-11 computer that examines data from a 3330-type moving head disk memory and signals the host processor if one of a number of specified keys are matched. The resolution of whether a document matches the specified expression is then performed by the minicomputer. It too does not include support for arbitrary characters, although a later unit includes this feature along with the ability to search specified documents rather than the whole database for a query.

Problems and Trade-Offs

There are a number of problems which must be considered when a specialized text searching processor is designed, along with a number of implementation trade-offs. Many of the solutions depend on the types of queries expected on the host information retrieval system, and have no correct answer. Others depend on the technology used to implement the backend system and its memory, and involve balancing cost and performance. These include the size of queries and individual terms that it can handle and whether it can process the requests of more than one user at a time.

Whether to handle only one user at a time, or to allow the simultaneous processing of multiple requests while the file is being scanned depends on the nature of queries in the system and the desired response times. If the queries involve the searching of only a small subset of the database, as would be the case using inverted files or results from previous searches, it may be possible to offer satisfactory response times serving users one at a time. However, if the entire database must be searched for each query, such as in the system described by Bird, it is better to have additional hardware to allow more than one user's request to be processed. In this case, the scanning logic must be able to handle considerably more terms and the resolution logic able to determine for which user the document matches.

The question is further complicated when some mechanism exists to rapidly position from one document to another, since documents that may be skipped by one user's query may be necessary for another's. The positioning logic must determine which document to select next, and indicate which user's queries it should be checked against. Another complication comes when a user enters a query during the time a search is already in progress. If the new query is immediately added to the search in progress, additional bookkeeping is necessary to determine when a query has been completed and to reorder the results that started in the middle of the file. If, on the other hand, the new query search is not started until the previous search completes, the system may function as if it could handle only a single user, depending on the arrival time of the queries and the time necessary to process them.

The size of queries and search terms is also highly dependent on the way the host retrieval system is used. If the words are all approximately the same length (no unreasonably long words or users willing to accept a search based on only the first n characters of a word), then simple fixed length comparators can be used (as in Stellhorn's proposed unit). If, on the other hand, these constraints are not acceptable, or it is not desirable to waste term storage within the scanning logic, an approach with a bulky memory holding variable length terms can be used, similar to the technique used by Bird. For the queries, the problem is not as difficult. A reasonable upper bound on the complexity of the query can be set after examining statistics on system utilization. If a query is larger than this upper bound, the system can split it into more than one query and merge the results of the separate searches, using a technique similar to that used in merging inverted file directory lists.

Parallelism and Spanned Contexts

For higher speed operation, it is desirable to either operate on larger data items (bytes or words instead of bits) or search more than one track of data at the same time. This is especially productive if the cost of the search logic is small when compared to the cost of storage device, as would be the case if the scanner were implemented using LSI techniques. The simplest form of parallel execution for a large database stored on more than one drive is to have separate search logic for each disk drive or group of drives. This is the approach suggested by Bird, and requires only the introduction of a priority arbitration connection between the various searchers and the host system. However, if only one user is being served at a time and only a limited subset of the documents are being searched, many of the searchers may be idle at any given time. This underutilization can be reduced by providing a switching network between a number of processors and a pool of memory devices.

Rather than use standard disk memory systems, where only one of the heads can be selected at a given time, the drives can be modified to transfer data from more than one head during an operation [12]. With this modification, either bytes can be processed instead of bits, or more than one track can be searched at a time. Processing byte streams from the disk rather than bit streams requires a more complex comparator (parallel rather than serial) and much faster resolution of whether a document matches, since the time allowed to compare a word from disk is reduced. If this parallelism is increased to a multiple of the byte size, additional difficulties arise in aligning the bytes within the word for comparison.

The other approach, searching more than one track at a time, presents problems not present in non-textual database processors of similar design. These database computers require that an entire item, such as an n-tuple, fit within a track in memory. If they are larger, schemes to break them into two or more items exist. With textual information, this is not generally possible. A great variation in the size of a context, such as a sentence, exists. Some are short. However, because (unlike non-textual database systems) the specific contents are not controlled by the system manager, but by the authors of the included documents, it is possible that very long and complex sentences may also exist, some possible taking up an entire paragraph, making

it difficult to insure that each sentence is entirely contained on a single track without wasting a great deal of disk storage capacity. The difficulty is more pronounced for higher level contexts such as a paragraph or whole document, for which storage on a single track may be impossible.

While this causes no difficulties when each track is examined sequentially, as in the unit suggested by Bird, when adjacent tracks are processed at the same time problems exist. For example, consider the case where a sentence starts on track A and ends on track B, and the query asks whether words X and Y both occur in a sentence. Suppose further that word X only appears on track A and word Y on track B. Then the logic searching tracks A and B each cannot determine if the documents match the query.

The solution to this problem is to introduce a communications path between neighboring resolution units, which functions if a desired context spans more than one track (indicated by the absence of either a start or end of context flag) and a search term has been located but the logic cannot determine if the document matches. In this case, the partial information is passed to the resolution logic of the track that contains the start of context flag for final processing. Since the context must extend to the end of this track (since it starts on the track and continues through the next), when the scanning of the context is completed by this unit, it will be completed by all other units. Final resolution can then be made while the system is positioning to the next group of tracks.

If the context starts on the current group of tracks, but ends on the group that will be searched next, the partial information is sent to the processor for the first of the next set of tracks, and its resolution unit acts as if it found the start of context flag. This special case functions similarly to the non-multitrack search when a context spans the current track.

Backtracking

Another problem not present in non-textual database processors and only to a limited extent in the other proposed textual systems is the necessity to backtrack during the search. This problem is introduced by using operators in the query that can match an arbitrary number of words or characters (the ... or operators in EUREKA), and the sequential nature of the memory system and the comparisons. For example, consider searching for the substring ISSIP in MISSISSIPPI. First, the M is compared against the I, and since it doesn't match, is skipped. The following ISSI is then matched, and the logic next looks for a P. However, it finds the current input character is an S, and the match fails at that point. However, if it starts a new search at this point, it will not find the desired substring, since it will be comparing an I from the search term against the current input character, an S. To function correctly, the logic must be able to back the input up before starting the search again. This is similar to the pulling back of the needle in a SNOBOL bead diagram [13].

Since the sequential memory cannot be backed up, some other scheme must be used. The simplest is to abort the search at the point where the problem occurs and, remembering the location, search the track again, this time ignoring the first, ill-fated match. This can substantially degrade system performance if it occurs many times. A second approach is to have a buffer that contains the last few characters. If a problem occurs, the search can be restarted with the data within the buffer. Finally, the scanning logic can be modified so that it is always looking for the start of a desired substring. In this case, for the example of MISSISSIPPI, a substring comparison will start every time an I is encountered in the input, with the second comparison succeeding and the other three failing. This approach requires many more comparators, but may be easier to implement if the bulk memory, variable length term comparison is being used. In that case, multiplexing of the memory and the comparators may reduce the number of gates in the comparison hardware, at an increase in control complexity.

Summary

The operations necessary for textual information retrieval in an automated reference library system have been presented. Although on the surface the character string matching operations appear similar to those on proposed database processors, the complexity of the queries and the lack of extensive formatting of the stored data require the development of different processors. The basic structure and operation of these processors was described and a number of problems and trade-offs in their design were presented.

References

1. Hollaar, L A, and P A Emrath. An Improved Query Language for the EUREKA Textual Information Retrieval System. EUREKA Project Memorandum, Department of Computer Science, University of Illinois, in preparation.

2. Hollaar, L A, et al. The Design of System Architecture for Information Retrieval. Proc ACM National Conf. 1976.

3. Hollaar, L A. Streaming Processor Networks for Combining Sorted Lists. Submitted to ACM Trans on Database Systems, September 1977.

4. Hollaar, L A, and W H Stellhorn. A Specialized Architecture for Textual Information Retrieval. Proc AFIPS NCC, June 1977.

5. Slotnick, D L. Logic per Track Devices. Advances in Computers 12, Academic Press, 1972.

6. Ozkarahan, E A, S A Schuster, and K C Smith. RAP -- An Associative Processor for Data Base Management. Proc AFIPS NCC, 1975.

7. Healy, L D, K L Doty, and G J Lipovski. The Architecture of a Content Addressed Segment Sequential Storage. Proc AFIPS FJCC, 1972.

8. Lin, C S, D C P Smith, and J M Smith. The Design of a Rotating Associative Memory for Relational Database Applications. ACM Trans on Database Systems, March 1976.

9. Hsiao, D K, and K Kannan. The Architecture of a Data Base Computer -- A Summary. Proc Third Non-Numeric Workshop, May 1977.

10. Stellhorn W H. A Processor for Direct Scanning of Text. presented at First Non-Numeric Workshop, October 1974.

11. Bird, R M, J C Tu, and R M Worthy. Associative/ Parallel Processors for Searching Very Large Textual Data Bases. Proc Third Non-Numeric Workshop, May 1977.

12. D R Johnson. A Two Channel Movable Head Parallel Access Disk Memory System. EUREKA Project Memorandum, Department of Computer Science, University of Illinois, 1976.

13. Griswold, R E, J F Poage, and I P Polonsky. The SNOBOL4 Programming Language. Bell Telephone Laboratories, 1968.