Text Retrieval Using Parallel Computers

2G−4

Ali M. AlHaj Eiichiro Sumita Hitoshi Iida
ATR Interpreting Telecommunications Research Laboratories

1 Introduction

With the recent wide-spread of electronic information systems, demands are mounting for efficient implementation of effective text retrieval algorithms. As such the case, efforts are being constantly made to respond to such demands. However, such efforts have been mostly directed towards implementing text retrieval algorithms on sequential computers which lack both the storage capacity and the retrieval speed that are needed for large on-line information systems. In this paper, we propose implementing text retrieval algorithms using parallel computers. First, an effective text retrieval algorithm is selected in section 2, and a highly parallel computer is briefly described in section 3. Next, a parallel implementation of the selected algorithm is described in sections 4 and 5. Finally, experimental evaluation and conclusions are given in sections 6 and 7.

2 Text Retrieval Algorithms

Most existing text retrieval systems are based on the inverted index & Boolean query text retrieval algorithm [1]. The inverted file consists of a list of keywords and identifiers of the documents in which they occur, and the user-supplied Boolean query consists of search terms interrelated by the Boolean operators (and, or, not). The retrieval operation returns to the user the names of the documents which have matched the query. As far as performance is concerned, text retrieval systems based on this algorithm are not easy to use and they exhibit poor retrieval quality. An alternative and more efficient approach to text retrieval has been the implementation of the the vector-model document ranking algorithm [1]. In this algorithm, both documents and queries are modeled as vectors. The retrieval operation consists of scoring documents vectors as to how well they match a given query vector, then returning the top ranked documents to the user. The score-and-rank method of retrieval generally yields superior quality of search while, at the same time, being easier to use because the user does not need to decide which Boolean connectives to use. Moreover, it is not necessary to learn a complex query language, as the query is simply a vector consisting of a list of words. Due to these advantages, we have selected this algorithm for parallel implementation.

3 The KSR Computer

The KSR computer is an MIMD type, distributed shared memory, highly scalable parallel computing system [2]. It combines the shared-memory architecture of traditional supercomputers and mainframe systems with the scalability of highly parallel systems. Unlike the typical memory architecture which has large pools of main memory and small caches, all KSR main memory consists of large, communicating local caches each of which physically adjacent to a processor. Communication between the caches is implemented using a slotted, pipelined, rotating ring. A parallel application can be easily implemented on the KSR computer by breaking it down into several pieces of work, and assigning each one to a pthread. A pthread is a sequential flow of control within a process that cooperates with other pthreads to solve the application problem. Pthread parallel programming is done using an extended version of the standard C language.

4 Database Representation

The first step in the implementation of a vector-model text retrieval system is to represent each document in the given full-text database by a weighted vector. The set of documents weighted vectors are considered as a compact representation of the original database. The generation of a weighted vector for each document is carried out on the host machine by running the following three text pre-processing operations:

- Filtering: use a stop list of common function words $(and, of, or, but, the, etc, ...)$ to delete from the document the high frequency function words that are insufficiently specific for document representation [3].
- Stemming: use suffix stripping routine to reduce the remaining words to word stem form [3].
- Weighting: for each remaining word stem i occuring in document j, compute a term weighting factor, which is the product of the term frequency of term i in document j multiplied by the inverse document frequency of term j in the documents database as a whole [4].

5 Query Processing

Processing a user-supplied query corresponds to determining which documents are relevant to that query. The processing steps are outlined below:

- Generate the query's weighted vector by following the same procedure described in the previous section. This step is performed sequentially on the host computer.
- Load the documents weighted vectors and the query weighted vector into the main memory of KSR computer.
- Assign the processing of different document vectors to different processors. The assignment should be made evenly-balanced in order to achieve high utilization of the computer.
- Compute in parallel the similarity scores between the query vector and each document vector. This corresponds to performing an inner product operation between the two vectors.
- Rank the documents according to their similarity scores with the query vector.
- Present the user with the names of the n top ranked documents. The full text of each document is available in the system disk of the host computer.

6 Experimental Evaluation

The parallel text retrieval system we are reporting here is an experimental system. Therefore, we have carried out our retrieval experiments using several document collections that have been developed exclusively for the purpose of performance evaluation. The collections cover topics in the fields of medicine, computer science, library science and engineering. The collections are rather small in size, ranging from 82 documents to 6004 documents [5]. The KSR computer used in our evaluation experiments consisted of 25 processors. A 32 MB of memory is connected to each processor, thus making the total available shared memory 800 MB (0.8 GB). For each collection, the host computer generated the documents weighted vectors and the weighted vector of an arbitrary query. The weighted vectors were then loaded into the KSR main memory. Next we carried out two retrieval experiments for each collection; sequential query processing and parallel query processing. In the sequential processing experiment, a single processor was used to process the query vector against all documents vectors, and in the parallel processing experiment, all the available 25 processors were used.

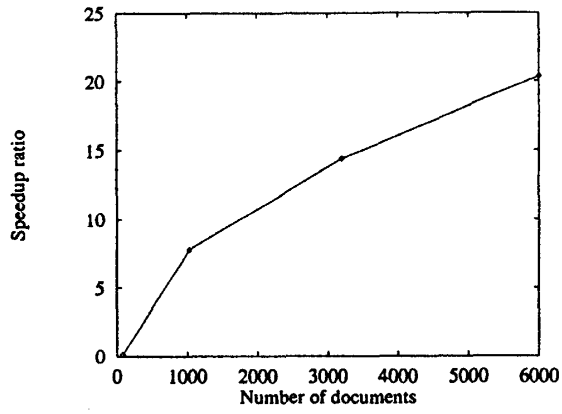

Figure 1: Speedup ratio vs. number of documents.

We recorded the retrieval time for each experiment, and compared them using the speedup ratio (speedup = retrieval time(1) / retrieval time(25)). The results are plotted in Figure 1. As the the speedup curve indicates, the speedup ratio increases as the number of documents increases. This suggests that using parallel computers to search text databases is particularly efficient when the size of text databases is large.

7 Conclusion

As the size of text databases becomes larger, the retrieval algorithms needed to locate required text may become computationally intensive. We have shown that the use of parallel computers represents the best means of coping with this explosion in the computational requirements for text retrieval.

References

[1] Salton, G. and McGill, M., *Introduction to Modern Information Retrieval,* McGraw Hill, New York, 1983.

[2] Kernel Square Research Corporation., Technical Manuals, MA, USA, 1994.

[3] Frakes, W. and Baeza-Yates, R., *Information Retrieval Data Structures & Algoirthms,* Prentice Hall, New Jersey, 1992.

[4] Salton, G. and Buckley, C., "Term weighting approaches in automatic text retrieval," in *Information Processing and Management,* 24, pp. 513-523, 1988.

[5] Fox, E., ed., Virginia Disk One, Virginia Polytechnic and State University, Blacksburg, 1990.