TR-IT-0114

Massively Parallel Text Retrieval

Ali M. AlHaj Eiichiro Sumita Hitoshi Iida

1995.5

Abstract

Text retrieval systems are computer-based systems the function of which is locate user-requested documents in text databases. The requested documents which are commonly stored in electronic form may include news articles, technical abstracts, office memos, electronic mail messages, among many others. In fact, the recent widespread of such electronic documents has greatly enhanced the importance of text retrieval systems and made them vital components in modern information systems. A wide variety of operational text retrieval systems already exit, however, these systems are mostly implemented on sequential computers which lack both the storage capacity and the retrieval speed that are needed for large on-line information systems. The research reported here is based on the argument that parallel computers represent a potential tool to cope with the increasing computational and storage demands of modern text retrieval systems. In this report we describe an efficient implementation of an experimental parallel text retrieval system using an MIMD parallel computer. The system is intended to be the first step towards achieving the ultimate goal of developing an effective and efficient retrieval system to support the on-going massively parallel example-based spoken language translation project at ATR Interpreting Telecommunications Research Laboratories.

ATR Interpreting Telecommunications Research Laboratories

1 Introduction

Text retrieval systems are computer-based systems the function of which is locate user-requested documents in text databases. The requested documents which are commonly stored in electronic form may include news articles, technical abstracts, office memos, electronic mail messages, among many others. In fact, the recent widespread of such electronic documents has greatly enhanced the importance of text retrieval systems and made them vital components in modern information systems. A wide variety of operational text retrieval systems already exit, however, their performance vary significantly with respect to retrieval effectiveness and retrieval speed.

Most commercial 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 pointers to the documents in which they occur, and the Boolean query supplied by the user consists of search words interrelated by the Boolean operators (and, or, not). The retrieval operation returns to the user pointers to the documents which have matched the query. As far as performance is concerned, Boolean systems are ineffective as they exhibit poor retrieval quality, and inefficient with respect to retrieval speed as they are usually implemented using sequential computers.

With the advances in computer technology and text retrieval algorithms, document-ranking retrieval systems are emerging as alternatives to the conventional Boolean systems. These systems are based on the vector processing retrieval algorithm in which documents and user queries are modeled as weighted vectors. The retrieval operation consists of scoring documents vectors as to how well they match the query vector, and then returning the top ranked documents to the user. The retrieval effectiveness of these systems is usually high by virtue of using weighted vectors and ranking. Furthermore, these systems can be implemented to yield fast retrieval speed as they are amenable to parallel implementations using large-scale computers, where a large number of vectors can be accommodated and processed in parallel.

A parallel implementation of the vector processing text retrieval algorithm has been carried out on the Connection Machine [2]. The implementation was based on a bit-string representation of the documents, where a binary data structure, called signature, was prepared for each document by superimposing the hashed bit representation of all its words. Obviously, this implementation avoided the weighted word vector representation of the documents since the memory units attached to the Connection Machine's small processing units are too small to accommodate word weights. Although retrieval speed gains have been obtained[3], the level of retrieval effectiveness was not satisfactory due to the fact that the binary vector representation did not support document word weighting [4].

In this paper, we describe a parallel vector processing text retrieval system. Unlike the SIMD type Connection Machine system, however, this system is implemented on the KSR parallel computer; a MIMD multiprocessor machine which has large memory units capable of accommodating sophisticated document and query word weights. An overview of the system is given in section 2, and a detailed description of its implementation is given in sections 3 and 4. Per-

formance of the system with respect to retrieval effectiveness and retrieval speed is evaluated in section 5, and concluding remarks and future work direction are given in section 6. Finally, a brief description of future research projects is given in section 7.

2 System Overview

An overview of the system is shown in the block diagram of Figure 1. It consists of four components; text database, weighted vectors generator, weighted vectors database, and parallel query processor. The vectors generator and the text database are implemented on the system's host (a SUN Sparc workstation), and the weighted vectors database and the parallel query processor are implemented on the KSR parallel machine.

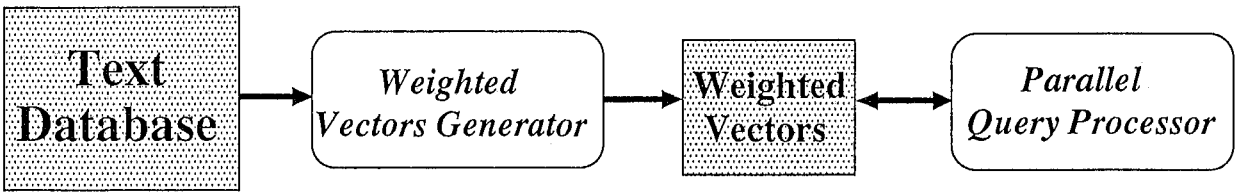

Figure 1: Block diagram of the system.

The database contains the texts of the documents or articles to be searched. It also contains a set of example queries and a file contationg their corresponding relevant documents which are needed for the operation of the relevance feedback method. The generator operates directly on the text database to produce a much smaller weighted vectors database. The parallel query processor operates directly on the weighted vectors database by performing a parallel match operation between a given query vector and all document vectors. It also runs the relevance feedback operation by formulating a new relevance feedback query depending on the query's relevance information.

The KSR parallel machine is an MIMD highly scalable parallel computing system [5]. It combines the shared-memory architecture of traditional supercomputers and mainframe systems with the scalability of highly parallel systems. Unlike the typical shared-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. Figure 2 shows the architecture of a KSR model (KSR-2) which consists of 25 processing elements connected to a distributed shared memory of 800 MB (0.8 GB).

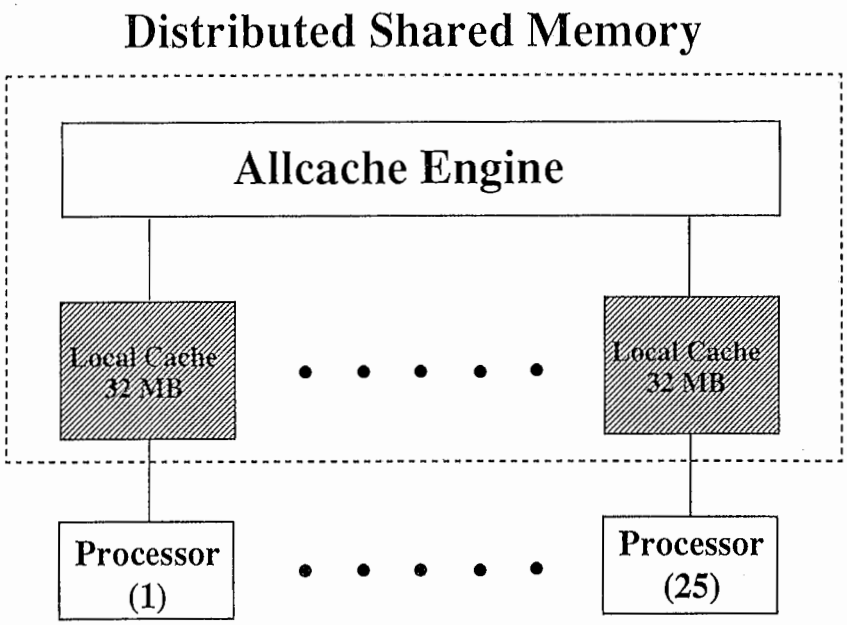

Figure 2: Architecture of the KSR Computer.

3 Weighted Vectors Generator

The weighted vectors generator consists of three main components; stop list words filter, suffix stripping stemmer, and word weights assignment function [6]. A block diagram of the generator is shown in Figure 3, and a brief description of each of its components is given below:

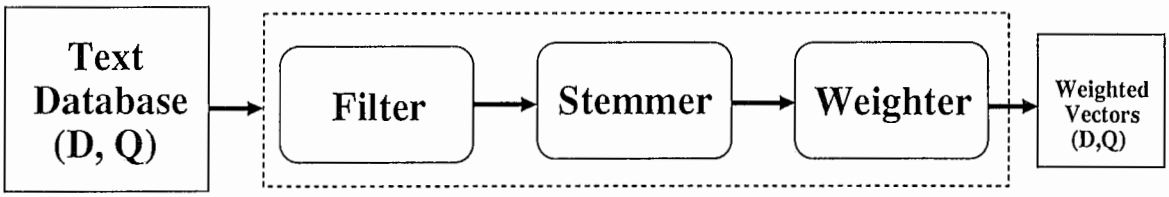

Figure 3: Components of the Weighted Vectors Generator.

- The stop list filter removes from the text of each document or query the most frequently occuring words in English such as $(and, of, or, but, the, etc, ...)$. These words are poor discriminatories, and their removal would have no effect on the retrieval effectiveness. Moreover, the filtering process reduces storage requirements and increases query processing speed. The filter was applied using a stop list consisting of 425 words derived from the Brown Corpus [7].

- The suffix stripping stemmer replaces the words preserved by the stop list filter to their stem forms. For example, the stemmer replaces a variety of different forms such as *analysis, analyzing, analyzes,* and *analyzed* by a common word stem *analy.* The stemming operation reduces storage requirements since many words are replaced by a single

stem word. Furthermore, it might increase the retrieval effectiveness since the stem word has a higher frequency of occurrence than that of the words replaced. In this system we used the well-known Porter stemming algorithm [8].

- The weight assignment function assigns a real-number weight to each word stem produced by the stemmer. The weight distinguishes the degree of importance of the word in the document (query), and thus leads to improved retrieval effectiveness. Moreover, it adds user-friendliness to the system as it facilitates ranking of the retrieved documents. In this system, we used the following weight assignment function for both the documents and the queries [9].

$$w_i = \frac{(0.5 + 0.5\frac{f_i}{\max f}).log\frac{N}{n_i}}{\sqrt{\sum_{j=1}^{j=W}((0.5 + 0.5\frac{f_j}{\max f})^2.(log\frac{N}{n_j})^2)}}$$

where,
w_i: weight of word i in the document (query).
f_i: frequency of occurrence of word i in the document (query).
n_i: number of documents(queries) to which word i is attached.
N: number of documents (queries) in the database.
W: total number of words in the document (query).

The denominator of the function above is a weight normalization component which ensures that the lengths of document (query) vectors are equal. The function assigns weights varying between 0 and 1, where 0 represents a word that is absent from the vector, and 1 represents a fully weighted word.

Given below is an example of the weighted vector generation process. The weighted vector of the short document " Photographic and Computer Systems in Biomedical Information.is generated as shown in Tabel 1. The weights are assigned arbitrary since only one document has been considered.

Document's Words	Filter	stemmer	Weighter
Photographic	Photographic	Photograph	.365
and			
Computer	Computer	Comput	.126
Systems	systems	System	.100
in			
Biomedical	Biomedical	Biomed	.409
Information	Information	Inform	.051

Table 1: weighted Vector Generation Example.

4 Parallel Query Processor

The parallel query processor scores documents vectors as to how well they match a given query vector, and then returns a ranked list of pointers to the top scored documents. It also executes the computation-intensive relevance feedback method which is applied to improve retrieval effectiveness. The operation of the query processor is shown in Figure 4, and is described below:

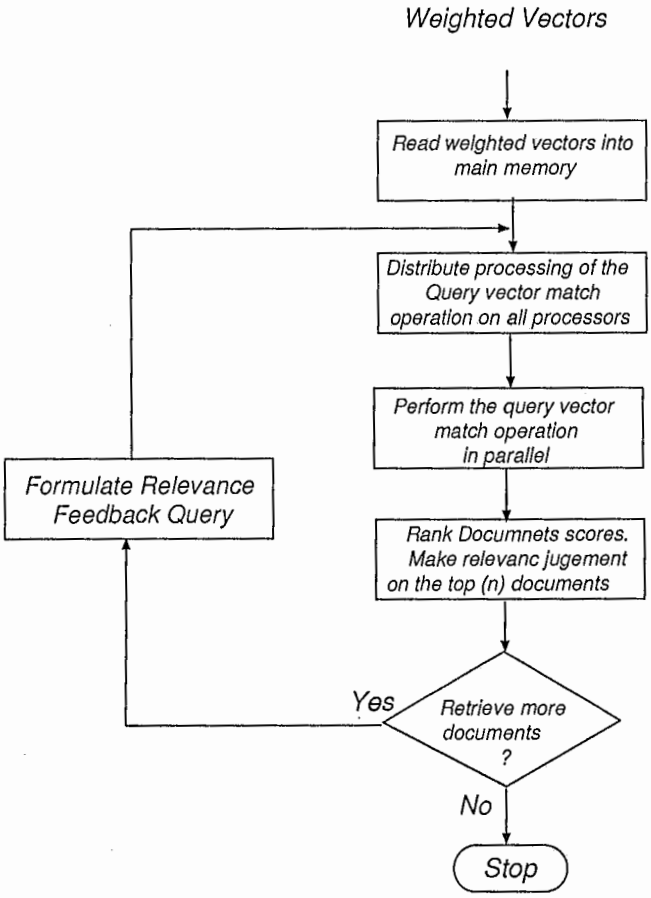

Figure 4: Parallel Query Processing Algorithm.

i. Read into the main memory of KSR the weighted vectors of all documents and the weighted vector of one example query. Each weighted vector is read and stored as a three-member structure; a variable of type integer to hold the size of the vector, an array of character strings to hold the words of the vector, and a corresponding array of type float to hold the weights of the words.

ii. Distribute the processing of the query vector matching operation on all KSR processing elements by assigning different document vectors to different processing elements. The document vectors assignment is made in such a way that each processing element P_x is assigned to process the query vector with D_x document vectors, where Dx is determined according to the following assignment function:

$$D_x = \begin{cases} floor(D/X) & \text{if } x > D \bmod X \\ ceil(D/X) & \text{otherwise} \end{cases}$$

D corresponds to the total number of document vectors and X to the total number of processing elements in KSR. This assignment attempts to achieve evenly-balanced processing so as to assure high utilization of the parallel machine resources.

iii. Perform in parallel the query vector matching operation. This corresponds to having each processing element P_x execute an inner product operation between the query vector and each document vector in D_x. Every inner product operation produces a real-number score which corresponds to the similarity between the document vector and the query vector.

iv. Rank the documents in decreasing order of their similarity scores, and retrieve the top n documents to judge for their relevance to the example query. This experimental system runs in a batch mode operation, and thus the relevance judgment is made automatically be referring to a relevance information file which contains names of the example query's relevant documents. In an interactive mode of operation, however, the relevance judgment is made by the user, see Figure 5. Upon making the judgment, if all (or sufficient) relevant documents have been retrieved, exit the retrieval operation and refer to the relevant documents full text in the disk file of the host machine. Otherwise, move on to the next step to retrieve more (or the remaining) relevant documents by applying the relevance feedback method.

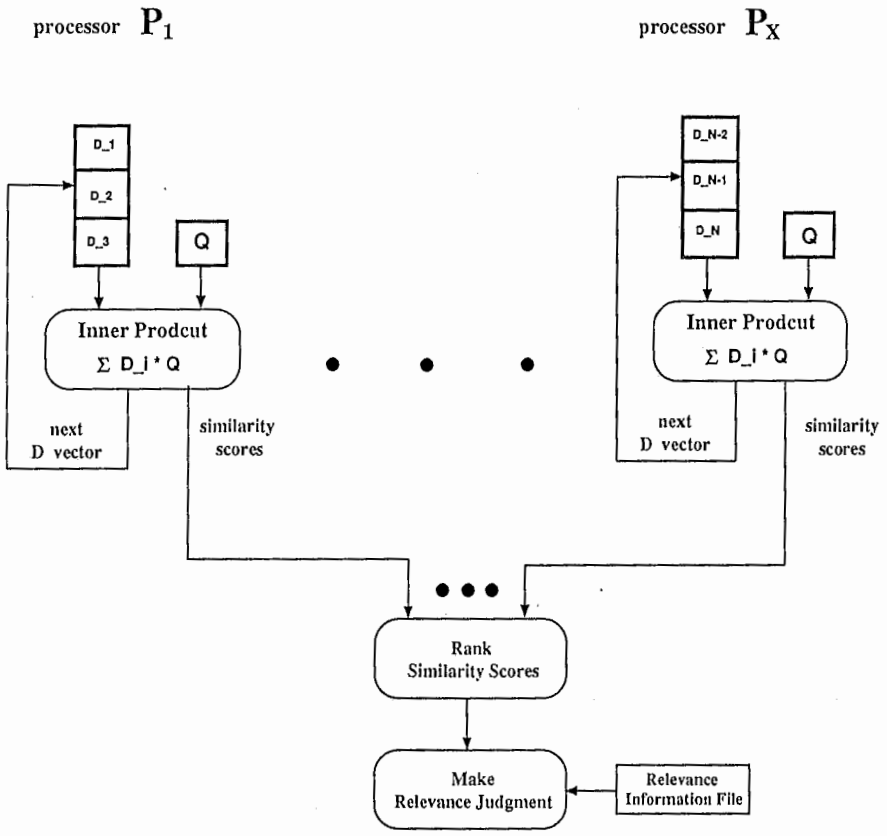

Figure 5: Parallel Similarity Computation and Ranking.

v. Reformulate the query vector by expanding and re-weighting its elements according to the following Ide dec-hi relevance feedback method [10]:

$$Q_{i+1} = Q_i + \sum All \ \ rel.docs - Top \ \ nonrel.doc$$

Q_{i+1} is the new query vector which is obtained by (1) adding to the previous query vector Q_i all words and corresponding weights of all relevant documents vectors, and (2) subtracting from the new query vector all words and corresponding weights found in the top ranked non-relevant document vector. Query reformulation using the Ide dec-hi method has proved to be superior to many other relevance feedback methods [11]. The above operation is illustrated in the following simple example:

- Q_1 = ⟨ Text 1.00, Retrieval 1.00 ⟩
- D1.rel = ⟨ Text 0.81, Retrieval 0.65 ⟩
- D3.rel = ⟨ Text 0.25, Processing 0.7 ⟩
- D2.nonrel = ⟨ Parallel 0.90, Processing 0.3 ⟩
- Q_2 ⟨ Text 2.06, Retrieval 1.65, Processing 0.4 ⟩

vi. Repeat steps $(iii - v)$ until all (or sufficient) relevant documents have been retrieved.

5 System Performance

We evaluated the performance of the system using six experimental document collections covering various subject areas [12]. Due to space limitation, however, the performance data reported in this section are those obtained using two collections only; the LISA library science collection which consists of 6004 documents and 35 example queries, and the ADI information science collection, which consists of 82 documents and 35 example queries, see Table 2. We conducted a series of retrieval experiments, in each of which the weighted vector generator produced the documents weighted vectors and the weighted vector of the query which has the largest number of relevant documents. The weighted vectors were then read and processed in parallel by the KSR computer. We evaluated performance of the system with respect to its retrieval effectiveness and retrieval time.

Collection Name	Subject Area	Number of Documents	Number of Queries
ADI	Information Science	82	35
MED	Medicine	1033	30
CRAN	Aeronautics	1400	225
CISI	Library Science	1460	112
CACM	Computer Science	3204	64
LISA	Library Science	6004	35

Table 2: The Virginia Disc One Test Collections.

5.1 Retrieval Effectiveness

Retrieval effectiveness of text retrieval systems is normally evaluated using the recall and precision measures. Recall is defined as the proportion of relevant documents that are retrieved from the document collection, and precision is defined as the proportion of retrieved documents that are relevant. We conducted several retrieval experiments as described above using the LISA collection, and tabulated the recall and precision ratios in Table 3. All ratios given in the table were computed under the assumption that the top 20 documents retrieved in each search iteration were judged for relevance, and the weighted words contained in all the relevant documents and the top non-relevant document were used to reformulate the query vector.

Iteration Number	Query Words	Recall Ratio	Precision Ratio
0	18	0.264	0.700
1	477	0.358	0.475
2	609	0.509	0.450
3	823	0.584	0.387
4	969	0.716	0.380
5	1228	0.735	0.325
6	1268	0.811	0.307
7	1396	0.849	0.281

Table 3: Retrieval effectiveness performance of the System.

Referring to the figures in Table 3, the 0 iteration corresponds to the processing of the initial query, and the successive iterations correspond to the processing of the queries formulated by relevance feedback. Although the number of feedback iterations can be set before hand, in our experiments it was determined dynamically by allowing a new feedback iteration only when the previous iteration produced at least one relevant document. That is, the relevance feedback operation was suspended automatically when the top retrieved documents were judged to include no relevant documents. Naturally, increasing or decreasing the number of documents retrieved in each search iteration would change the number of feedback iterations, i.e, more retrieved documents leads to less feedback iterations.

The recall ratios in Table 3 show that the retrieval quality improved as more feedback iterations were executed. However, to evaluate the true effectiveness of the relevance feedback process, it was also necessary to compare the performance of the feedback iterations search with the results of the initial search performed with the original query vector, i.e, iteration 0. To perform such measurement, the frequently used residual collection method [11] was applied. In this method all documents previously retrieved were simply removed from the collection, and the subsequent searches were evaluated using the reduced collection only. The improvement in the recall measure achieved by applying the relevance feedback method is demonstrated in Figure 6.

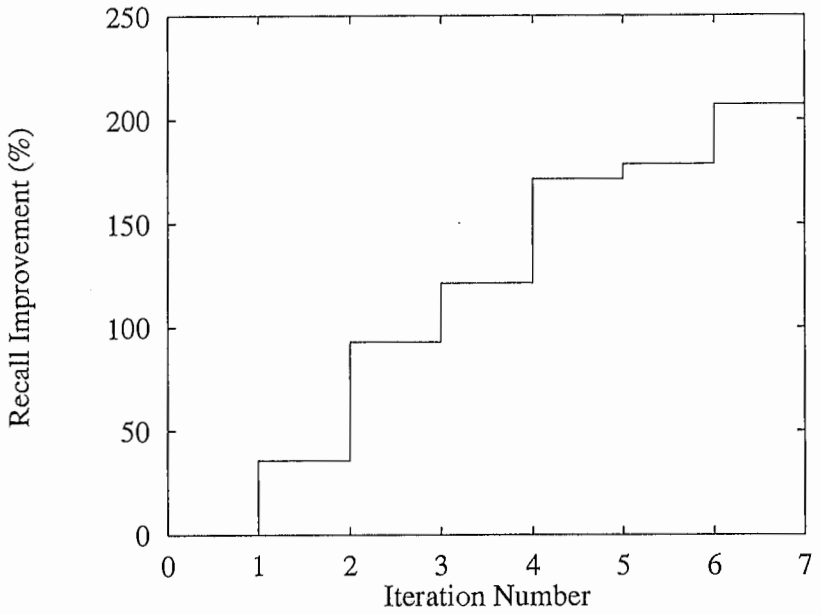

Figure 6: Improvement in the recall measure as a function of relevance feedback iterations.

5.2 Retrieval Time

We studied retrieval time performance of the system as a function of the number of words in the queries formulated by the relevance feedback, and as a function of the number of documents in the collection. In both cases, the measured retrieval time corresponded to the time spent by KSR to match the query vector against all the document vectors plus the time spent to rank the similarity scores in decreasing order.

It is clearly evident from the query words entry of Table 3 that the number of words in the newly formulated queries increased as more relevance feedback iterations were executed. This increase in query size would naturally lead to a proportional increase in the retrieval time. Based on our measurement, it took about 0.23 m-secs to process the initial query vector which consisted of 18 words, and 13.93 m-secs to process the last query vector which consisted of 1396 words. We plotted the retrieval time as a function of the number of words in the query vector in Figure 7 along with the retrieval times obtained when the same retrieval experiment was conducted using a single processor. On the average, a speedup ratio (S) of 22 was achieved, where (S) was computed as Time(1PE)/Time(25PE). Dividing (S) by the total number of processors, we get an approximate system utilization ratio of 0.88.

To measure the retrieval time of the system as a function of the number of documents, we compared the speedup performance of the LISA collection with the speedup performance of the much smaller ADI collection. The average speedup and utilization ratios obtained for the two collections are given in Table 4. As shown in the table, the LISA collection achieved a better utilization of KSR resources, and thus a higher speedup ratio. This suggests that the performance of the system is particularly efficient when the size of text database is large. It is therefore quite possible to assume that by virtue of the high scalability of the KSR system, large and practical text databases can be efficiently searched using KSR by simply adding more processing elements to the system.

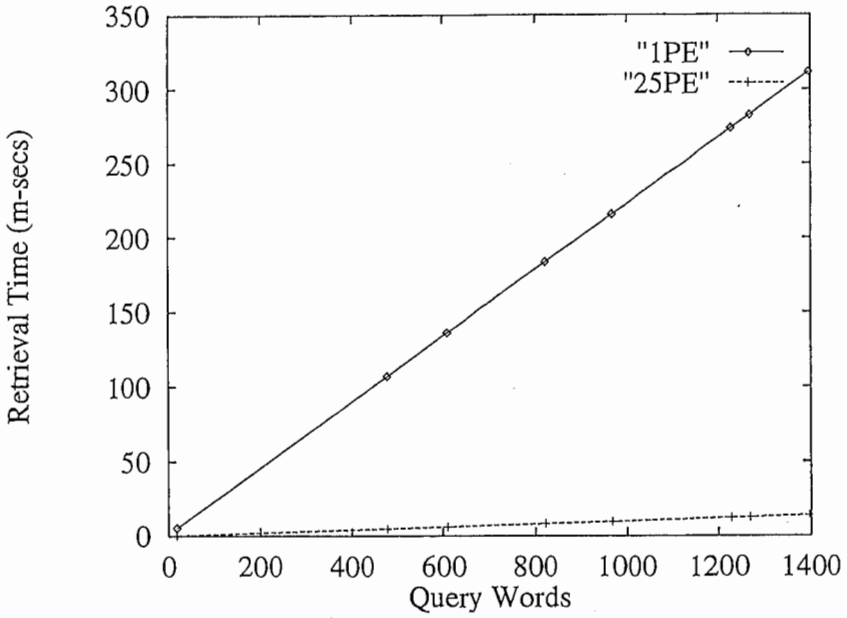

Figure 7: Relevance feedback query size vs. parallel and sequential retrieval times.

Collection Name	Documents Number	Maximum Query Size	Speedup Ratio	Utilization Ratio
ADI	82	860	14.03	0.60
LISA	6004	1396	22.68	0.88

Table 4: Performance comparison between the LISA and ADI documents collections.

6 Conclusions

Efficient implementations of the vector processing text retrieval algorithm could produce effective and efficient text retrieval systems. In this paper, we described an efficient implementation of a parallel vector processing text retrieval system, and confirmed its effectiveness and efficiency in the context of the computational demands of the relevance feedback method and the size of the text database.

7 Future Work

Future work consists of investigating the performance of parallel implementations of several text retrieval systems. These systems include:

- A Parallel Interactive Text Retrieval System.
- An SIMD Parallel Text Retrieval System.
- A Parallel Boolean Text Retrieval System.

7.1 A Parallel Interactive Text Retrieval System

The system which has been described in this report is a parallel batch text retrieval system. An interactive system can based on this system provided that interesting document collections are selected and a friendly interactive environment is developed. Pre-processing and query processing are basically the same as the already developed for the batch system. Implementation flowchart of a KSR-based parallel interactive system is shown and described below.

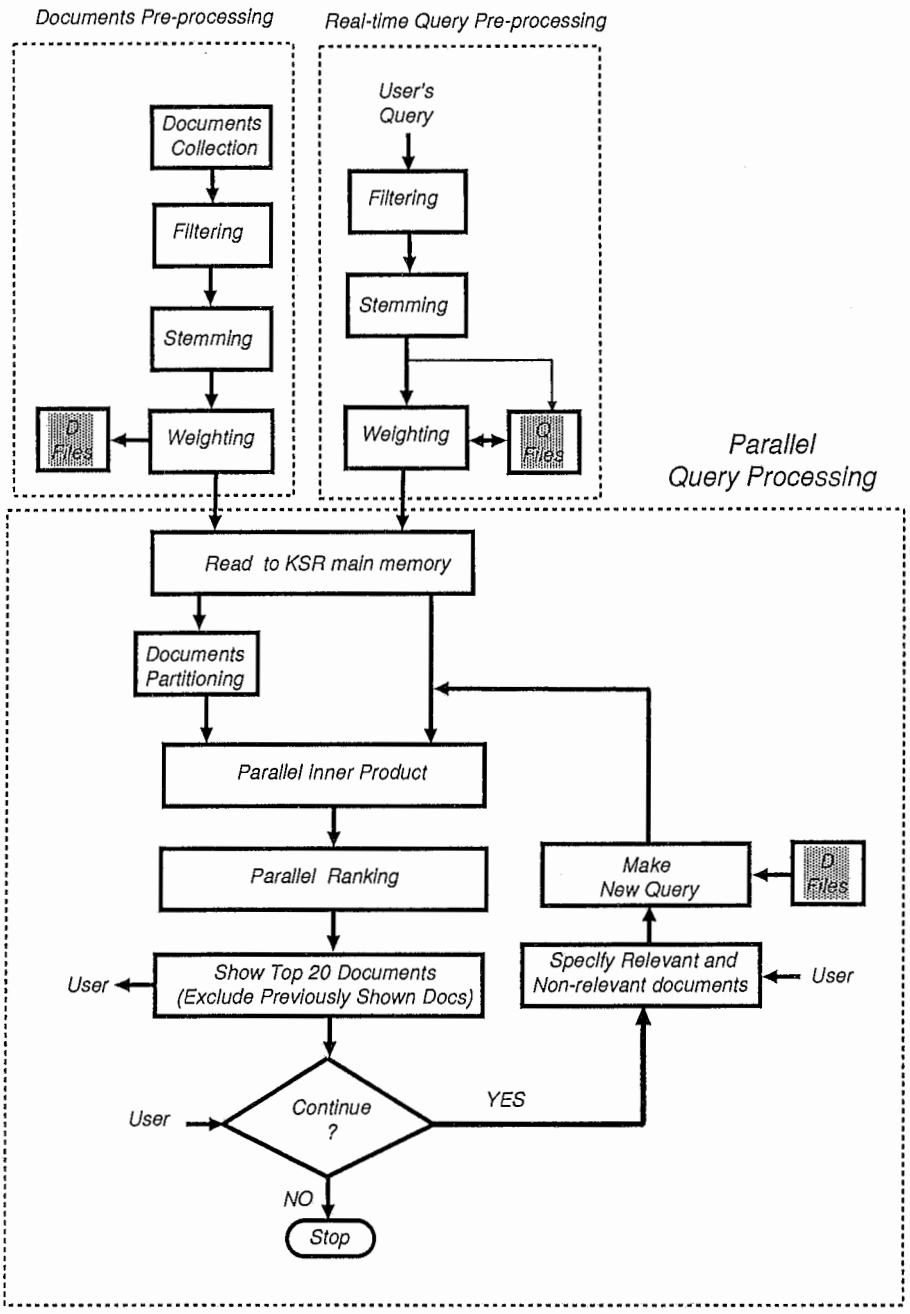

Figure 8: Implementation flowchart of a parallel interactive text retrieval system.

7.1.1 Documents Pre-Processing

The documents can be selected from the collections described below, and then filtered, stemmed and weighted as described in section 3.

- Virginia Disc One: These collections can be selected as a starting point. However, the relevance feedback should be implemented by applying the relevance information from the command line and not from the relevance information file as was done in the batch system.

- Clarinet News: Several interesting groups can be selected as documents collections.

- The TIPSTER documents collections: these collections contain news form different sources and are found under the following directories.

7.1.2 Real-Time Query Pre-Processing

Each query supplied by the system user is filtered and stemmed, and weight of 1.00 is given to each term. If there is a set of queries prepared in advance by analyzing the documents collection or gathered from previous users, then the user-supplied query is filtered and stemmed by itself, and the its weights are determined using the documents weighting function the weights of all queries in the queries set.

7.1.3 Parallel Query Processing

- Read into the main memory of KSR the weighted vectors of all documents and the weighted vector of the user's query. Each weighted vector is read and stored as a three-member structure; a variable of type integer to hold the size of the vector, an array of character strings to hold the words of the vector, and a corresponding array of type float to hold the weights of the words.
- Distribute the processing of the query vector matching operation on all KSR processing elements by assigning different document vectors to different processing elements. The document vectors assignment is determined according to the assignment function described in section 4.
- Perform in parallel the query vector matching operation. This corresponds to having each processing element P_x execute an inner product operation between the query vector and each document vector in D_x. Every inner product operation produces a real-number score which corresponds to the similarity between the document vector and the query vector.
- The resultant scores are returned ranked with their corresponding documents numbers(names).
- The user is asked to see if the top (10-20) retrieved documents satisfies his needs or not. If he is satisfied, then the operation is stopped, otherwise he is asked to check the documents and mark those relevant and non-relevant to his query.
- The relevance feedback is activated to add all terms from the user-marked relevant documents to the original query, and subtract the top non-relevant document in order to make a new query. The score and rank process is repeated until the user is satisfied.

7.2 An SIMD Parallel Text Retrieval System

A MasPar-based parallel text retrieval system does not have to be designed from scratch. Pre-processing and relevance feedback query formulation are the same as described in sections 3, and 4. Implementation flowchart of a MasPar-based parallel text retrieval system is shown and described below.

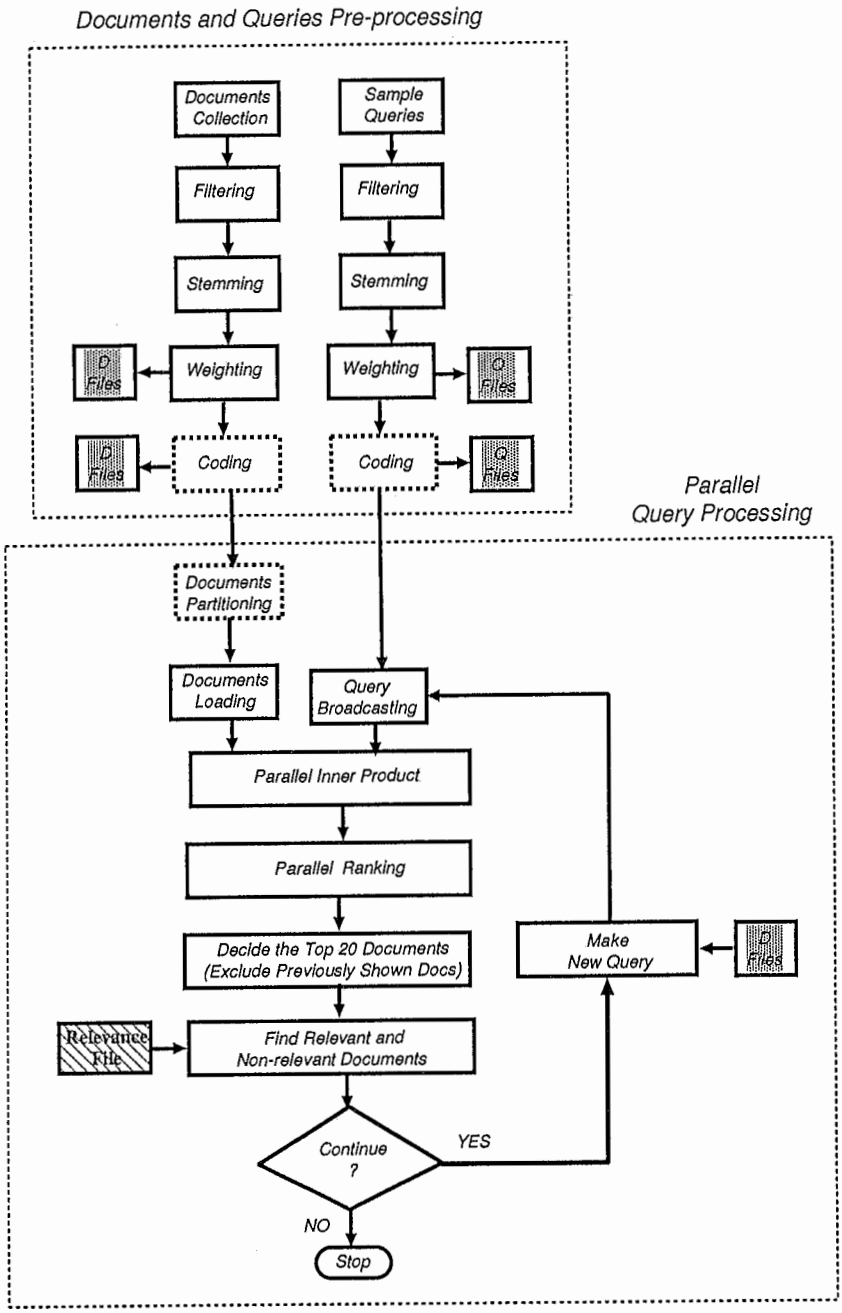

Figure 9: Implementation flowchart of an SIMD parallel text retrieval system.

7.2.1 Pre-Processing

- Same as with the KSR system implementation described in section 3. The weighted vectors for both the documents and queries can be used as they are with no modification if they do not require a storage capacity larger than the maximum storage capacity of the processing elements.

- There are cases when the weighted vectors are too large to be stored in the local memories of the processing elements. In such cases some sort of coding must be employed.

7.2.2 Query Processing Algorithm

- All documents vectors are loaded into the Parallel Array. Each PE stores the weighted vector of one document weighted vector. However, if the number of documents is larger than the number of processing elements, then some sort of virtual mapping is required. In such a case, each processing element would be assigned more than one document. The partitioning scheme described in section 4 can be applied, however, a more efficient scheme might be necessary.

- After deciding on how many documents a processor element can store, the documents vectors are loaded sequentially from the SUN file system into the processors; one processor at a time. However there is a possibility to load the documents vectors into maspar parallel disc and keep them there. Later when the program is executed, all vectors can be read into the processing elements in parallel. All is required by the program is the name of the collection to be loaded from the parallel disc.

- The initial query weighted vector is read form the SUN file system into the main memory of the front end machine. It is then broadcasted to all processing elements simultaneously.

- An inner product operation is performed in each processor using query and documents vectors stored in its local memory. Every inner product operation produces a real-number score which corresponds to the similarity between the document vector and the query vector.

- A parallel rank operation is performed on all scores computed in the previous step.

- The scores and their corresponding ranks are returned to the front end machine in order to evaluate the performance.

- At the front end machine, generate the new relevance feedback query and repeat the above steps. The new query generation and quality computation performed by the front machine are exactly the same as used in the KSR system implementation.

7.3 A Parallel Boolean Text Retrieval System

The implementation process of the system consists of preprocessing the documents collection to generate the the inverted index, and Parallel search and Boolean manipulation of the documents. Implementation flowchart of a parallel Boolean text retrieval system is shown and described below.

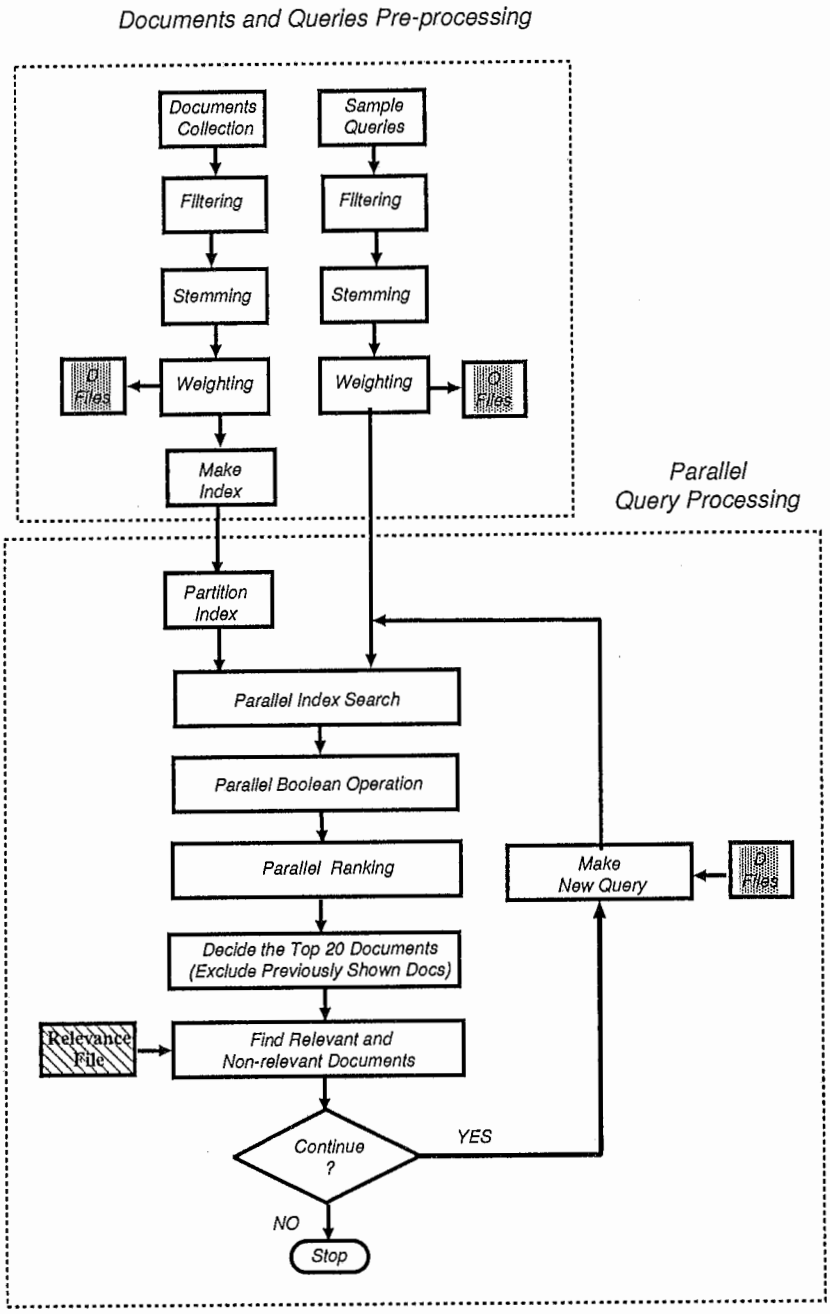

Figure 10: Implementation flowchart of a parallel boolean text retrieval system.

7.3.1 Pre-Processing and Inverted Index construction

- pre-process all documents in the collection by applying the operations; filtering, stemming and weighting. All Virginia Disc document collections have been pre-processed already, and thus they can be readily used as they are.
- Construct a weighted inverted index from all the weighted words of all documents. A simple example of a weighted inverted index is shown below, where w associated with each document is the weight of the index entry in the that particular document.

Term	Documents(weights)
Text	D1(w), D2(w), D3(w)
Retrieval	D1(w), D3(w), D9(w)
Parallel	D4(w), D7(w), D8(w)
Computer	D1(w), D4(w), D5(w)

Table 5: weighted Inverted Index Example.

7.3.2 Boolean Query processing

- Processing a boolean query usually takes two steps:
  - find from the index the documents and corresponding weights for each term in the example boolean query.(a set of boolean queries is provided with each collection in the Virginia disc)
  - Perform Boolean operations (and, or , not) on the results. As an example, if the above inverted index is to given the boolean query: Text $\cap$ Retrieval, then the corresponding processing is as follows:
    (D1, D2, D3) $\cap$ (D1, D3, D9) = D1, D3.
- The above query processing should be implemented sequentially on the SUN machine as a starting point, and on later on the KSR machine. The parallel implementation should consider two things:
  - developing a parallel search algorithm which can distributes the inverted index entries equally on all processing elements so that the search for a given query terms can be done in parallel.
  - developing a parallel algorithm to execute the boolean operations (and, or, not).
- Applying the Relevance Feedback: The relevance feedback is usually used with the vector processing text retrieval algorithm. Some literature is available on using the relevance feedback method with the Boolean text retrieval algorithm[11].

Acknowledgment

The authors would like to express their gratitude and appreciation to Mr. Hitoshi Nishimura and Mr. Masayoshi Sato for the their continuous technical support.

References

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

[2] Stanfill, C. and Kahle, B., "Parallel Free-Text Search on the Connection Machine," in *Communications of the ACM,* 29(12), pp. 1229-1239, December 1988.

[3] Waltz, D., "Applications of the Connection Machine," in *Computer,* 20(1), pp. 58-79, January 1988.

[4] Salton, G. and Buckley, C., "Parallel Text Search Methods," in *Communications of the ACM,* 31(2), pp. 202-215, February 1988.

[5] Kendall Square Research Corporation., Technical Manuals, MA, USA, 1994.

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

[7] Fracis, W. and Kucera, H., *Frequency Analysis of English Usage,* Houghton Mifflin, New York, 1982.

[8] Porter, C., "An Algorithm for Suffix Stripping," in *Program,* 24(3), pp. 56-61, 1980.

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

[10] Ide, E., "New Experiments in Relevance Feedback," in *The SMART Retrieval System,* ed. Salton, G., pp. 337-354, Prentice Hall, New Jersey, 1971.

[11] Salton, G. and Buckley, C., "Improving Retrieval performance by Relevance Feedback," in *Journal of the American Society for Information Science,* 24, pp. 288-297, 1990.

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

[13] Oi, K., Sumita, E., Furuse, O., Iida, H., and Kitano, H., "Toward Massively Parallel Spoken Language Translation," in *Proc. of the* 2^{nd} *Workshop on Parallel Processing for Artificial Intelligence, IJICAI-93,* pp. 36-39, France, August 1993.