Parallel Text Searching In Serial Files Using A Processor Farm

Janey K. Cringean[1], Roger England[3], Gordon A. Manson[2] and Peter Willett[1,4]

Departments of Information Studies[1] and Computer Science[2], University of Sheffield, Western Bank, Sheffield S10 2TN, U.K. and National Transputer Support Centre[3], Sheffield Science Park, Arundel Street, Sheffield S1 2NS, U.K.

[4] to whom all correspondence should be addressed.

Abstract. This paper discusses the implementation of a parallel text retrieval system using a microprocessor network. The system is designed to allow fast searching in document databases organised using the serial file structure, with a very rapid initial text signature search being followed by a more detailed, but more time-consuming, pattern matching search. The network is built from transputers, high performance microprocessors developed specifically for the construction of highly parallel computing systems, which are linked together in a processor farm. The paper discusses the design and implementation of processor farms, and then reports our initial studies of the efficiency of searching that can be achieved using this approach to text retrieval from serial files.

1 Introduction

The last few years have seen a proliferation of interest in the use of *parallel processing* techniques, where some or many processors operate together so as to reduce the elapsed time required for a computational task. The use of multiple processors can bring about substantial increases in performance, as well as providing some degree of fault tolerance (and hence graceful degradation in the case of a system malfunction) and a facile upgrade path (since increased computational demands can be met simply by acquiring additional processors).

There have been several attempts to classify the many ways in which parallelism may be implemented in a computer system (Flynn, 1972; Handler, 1977; Hockney & Jesshope, 1988; Shore, 1973; Skillicorn, 1988). Of these, the most common is that developed by Flynn (1972), which classifies computer systems in terms of the multiplicity of the instruction stream and of the data stream. Four possible architectures are recognized:

Permission to copy without fee all part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

(C) 1990 ACM 0-89791-408-2 90 0009 429 $1.50

- SISD, or single instruction stream, single data stream.
- SIMD, or single instruction stream, multiple data stream.
- MISD, or multiple instruction stream, single data stream.
- MIMD, or multiple instruction stream, multiple data stream.

Of the four classes, MISD is difficult to conceptualize and is usually regarded as a null class, while SISD represents the serial architecture that has formed the basis for nearly all computers since their initial development over forty years ago. SIMD machines are computers in which the same instruction is executed in parallel across some, or many, processors simultaneously. Examples include pipelined vector processors and array processors (Hockney & Jesshope, 1988; Sharp, 1987). The latter type of machine, which contains thousands of simple, bit-serial processing elements, is particularly well suited to the processing of large databases and there have been many reports of the use of two types of array processor, the Distributed Array Processor and the Connection Machine, for database processing applications (see, e.g., Carroll *et al.*, 1988; Waltz *et al.*, 1987): a review of this work is presented by Willett and Rasmussen (1990).

MIMD computers are characterised by multiple processors capable of independent operation; this allows a greater architectural variation than is the case with SIMD machines, and requires a greater range of organisational problems to be overcome if efficient computation is to be achieved (Dubois *et al.*, 1988; Gajski & Peir, 1985; Patton, 1985). The most widely applied sub-categorisation of MIMD computers depends on the degree to which the processor memories are coupled. In tightly coupled systems, all of the processors share a global memory through some central switching network or a high-speed bus that links the processors together. In loosely coupled systems, conversely, each of the processors in the network has a local memory and communicates with other processors by passing messages; thus facilities for highly efficient inter-processor communication are required. There are several ways in which message passing MIMD computers can be built: we are interested here in what are often referred to as *multicomputers*, *multicomputer networks* or *microprocessor-based multiprocessors*. A multicomputer consists of a large number of low-cost, high-performance microprocessors, each of which has some local memory and which can communicate with other such processors over some type of network (Athas & Seitz, 1988; Fox *et al.*, 1988; Reed & Fujimoto, 1987).

There have been a few reports of the use of multicomputers for text retrieval (Cringean *et al.*, 1988; Cringean *et al.*, 1989a; Sharma, 1989; Stewart & Willett, 1987; Walden & Sere, 1989). Cringean *et al.* (1989b, 1990) review the work to date and give a detailed specification of a multicomputer system that is designed for text scanning in serial files and that is intended to be used in conjunction with conventional PC-based equipment. The system uses an initial text signature search

to reduce the number of documents that need to undergo the detailed and time-consuming pattern matching search. It is in the pattern matching stage that the parallel processing capabilities of the multicomputer come most obviously into play, using the *processor farm* approach to distributed processing (as described in Section 3 below). The present paper reports the results of initial experiments using this multicomputer system for serial text scanning.

2 The Transputer

The multicomputer networks considered here are based on the INMOS Transputer. A transputer is a high-performance processor developed especially for use in multiprocessor systems (Hey, 1988; Pritchard *et al.*, 1987). Each transputer consists of a processor with its own memory and links to connect it to other transputers, all on a chip less than a centimetre square. In fact, the transputer is a generic name for a family of devices. Each device contains some selection of the following architectural features capable of operating concurrently with other features:

- a high performance (up to 20 MIPS) 16-bit or 32-bit RISC central processor, including hardware support for simulated concurrency on a single processor; this special hardware facilitates fast process-switching which is orders of magnitude faster than time-slicing facilities on conventional non-parallel machines.
- very fast RAM, typically of size 4 Kbytes.
- external memory interface, with either multiplexed address and data buses for economy of device pins and price, or non-multiplexed for performance.
- some number (typically two or four) of serial communication link processors, for bi-directional communication between pairs of devices within the family. The links operate at speeds of 5, 10 or 20 Mbits/sec., and can achieve data transfer rates up to 1.7 Mbytes/sec. unidirectionally, and 2.3 Mbytes/sec. bidirectionally.
- further special function co-processors, currently including a floating point processor and disk controller.

The most important of these features are the links, which enable transputer devices to be used as building blocks in the construction of low-cost, high performance multiprocessing systems. Communication takes place only between pairs of devices and is distributed throughout a multiprocessing system, thus overcoming the classic von Neumann bottleneck of limited bandwidth which is often encountered in bus-based multiprocessing systems. It should be noted that even simple, single processor applications can make use of the concurrent operation of the CPU and link processors;

at any one instant, the CPU might be processing one item of data, one link transferring from disk to memory the next item of data, and a further link transferring from memory to disk the previous calculated result.

There are several different classes of transputer device. The two of most importance in the context of this paper are:

- T414 – this is a 32-bit processor, containing 2 Kbytes of RAM and four links operating at up to the maximum data rates previously quoted.
- T800 – this is a T414 processor with the addition of a concurrent 64-bit floating point processor that has been formally verified to conform to the ANSI IEE 785 standard and can sustain up to 2.3 Mflops/sec.

The particular transputer equipment that we have been working with is a model M40 Meiko Computing Surface. This is a modular transputer system that allows the network topology to be determined by the user. A Computing Surface may be configured as a general purpose computing resource or be optimised to a specific function using any mixture of subsystem boards to populate the expansion slots of the M40. Our work has used the following subsystem boards:

- MK014: Local Host board, incorporating a T414 transputer with 3 MBytes of external RAM, IEEE 488 and dual RS232 I/O interfaces.
- MK021: Mass Store, incorporating a T800 transputer with 8 MBytes of fast parity-checked RAM.
- 4×MK009: An MK009 Quad Computing Element system incorporates 4 T800 transputers, each with 256 Kbytes of external RAM; thus 4 boards provide 16 T800 transputers.

Transputer systems are designed in terms of an interconnected set of *processes*, i.e., subcomponents of the overall computational task. These processes are executed using one or more *processors*, i.e., individual transputers in the present context. Each process can be regarded as an independent unit of design, which communicates with other processes along point-to-point links called *channels*. The internal structure of each process can be in terms of a set of communicating processes. In fact, a complete program can be considered as a process made up of other communicating processes. This is reflected in the design of the language occam [1], which has been developed in conjunction with the transputer (May & Taylor, 1984) and which has been used for all of the work described in this paper. Programs can be developed in occam to run on an individual transputer or a network of transputers. When occam is used to program an individual transputer, all program code is placed on one transputer and the

[1] occam is a trademark of the INMOS Group of Companies.

processor shares its time between the concurrent processes; channel communication is implemented by moving data within memory. When occam is used to program a network of transputers, one or several processes may be allocated to each processor; communication between processes on different transputers is implemented directly by transputer links.

Having introduced the transputer and the process model of computation, the next section describes the processor farm approach to distributed computing that we have adopted for our studies of serial text retrieval.

3 Implementation Of Processor Farms

Database searching involves accepting an input stream of data records, D, and then carrying out some sort of matching operation, $M(D, R)$, which outputs a results stream, R, containing the records from D which match the query. Cringean *et al.* (1988) discuss several different ways of distributing thse matching operations across a multicomputer network and conclude that the *processor farm*, or *processor pool*, approach to distributed computation (Hey, 1988; Pritchard *et al.*, 1987) is the most appropriate for this type of application.

The farm approach involves decomposing the input datastream, D, into n sub-streams,

$$d_1, d_2, \ldots d_i, \ldots d_n \ (1 \leq i \leq n),$$

where n is the number of processors. The matching operation, M, is then replicated to give a series of identical processes, i.e., sub-components of the overall computational task,

$$M_1(d_1, r_1), M_2(d_2, r_2), \ldots M_i(d_i, r_i), \ldots M_n(d_n, r_n),$$

each of which outputs a result stream, r_i. Two main processes are required: one to decompose the datastream and to replicate M and the other to combine the n individual sets of results; in what follows, we shall refer to these as the *distributor* and the *collector* processes, respectively.

We can represent a processor farm logically by the flow diagram of Figure 1. Each worker process requests a data packet from the distributor process, performs the matching operation on that data, and sends the results to the collector process. The distributor process sends a data packet to an individual worker process whenever that process requests work. However, the network shown in Figure 1 cannot be

implemented directly using transputers because there are only four links per transputer at present. Networks must therefore be designed to allow this logical model to be mapped onto a feasible physical network. One of the simplest physical networks that can be implemented with transputers is the *single linked chain*. In a single linked chain, the transputers are linked to each other in a simple, linear array, so that some transputer, T_j, in the body of the chain of n transputers is connected to two other transputers, T_{j-1} and T_{j+1} $(1 < j < n)$, as depicted in Figure 2.

If we wish to implement a processor farm based on a single chain, then data must be passed down the chain in one direction and results must be passed up the chain in the opposite direction along the transputer's bi-directional links. The distributor and collector processes are often executed on a single controlling processor, as in Figure 2. This processor is connected *via* one of its four links to the transputers which make up the farm and by another link to a host machine (typically a PC or a workstation) that provides terminal and file system support. Because it is attached to the host machine, this processor is called the *root* processor.

This physical network is not as simple as the logical model of the farm, where the distributor and collector have direct access to every worker process. In the case of a single linked chain the distributor and collector have direct access only to the worker process at the top of the chain, as is shown in Figure 2. Therefore, additional processes are required to allow a message to be forwarded from one transputer to another *via* other transputers in the network. The workers must therefore be able to do more than request some work, input a work packet, perform some work and output a result. They have to be able to input a work packet, decide whether it is intended for them, then either perform some work and output a result or pass the message further down the chain. Because all the functions of a single transputer can work independently of the others, it is possible to allow communication and decision making to be performed on one work packet at the same time as work is being performed on another.

A slightly more complicated network is the *triple chain*, as shown in Figure 3. In this case, instead of having just a single chain attached to the root processor there are three chains, thus using all available links on the root processor. This network has the advantage that messages have a smaller average distance to travel without the network being too complicated.

4 System Design

4.1 Introduction

Our work in Sheffield seeks to evaluate the efficiency of nearest neighbour searching in transputer networks, using *text signature* representations to increase the efficiency of scanning in files of documents that are organised using the serial file structure, rather than the inverted file structure that provides the basis for the great majority of current text retrieval systems. A text signature is a fixed-length bit string in which bits are set to describe the contents of a document or query. These bit strings are created by applying some hashing-like operation to each of the terms describing a document or query, each such operation causing one or more bits in the string to be set (Faloutsos, 1985). Signature matching operations are probabilistic in character since a match at the bit string level is a necessary, but not sufficient, condition for a match at the word level. Thus, *false drops*, i.e., matches at the bit level that do not correspond to matches at the word level, can always occur. The elimination of such mis-matches requires the use of a second-level, pattern matching search which is carried out for just those documents that match at the signature level (so that the computationally demanding text scanning is applied only to small numbers of documents). There is a further problem with text signatures, this being the fact that the setting of a particular bit in a signature indicates merely the possible presence of a term in a document or query, this giving no information about the location of the word in the document or query or of how frequently the word occurs (Salton & Buckley, 1988). Thus, the specification of proximity or adjacency information in a query or the use of some types of frequency-based weighting schemes will again require the use of a second-stage search. For these reasons, we believe that it is imperative that retrieval systems based on text signatures should include the second-stage pattern matching search after the signature search; if this is not the case, then the undoubted efficiency of text signatures will be achieved only at the cost of reductions in effectiveness. Since pattern matching is very demanding of computational resources, it is thus an obvious candidate for the application of parallel processing techniques, as discussed in the remainder of this paper.

4.2 Implementation of nearest neighbour searching

In brief, the problem tackled is that of searching in a serial file for a query using a nearest neighbour retrieval algorithm. In common with most such retrieval systems, we will allow the query to be input at a keyboard as a natural language statement of need. The words in the query are compared with a stopword list and the remaining, content bearing words are then stemmed to conflate morphological variants. The stems are hashed into a bit string to form the query signature and this is then matched with a set of analogous signatures that represent each of the documents in

the database. The documents are ranked in order of decreasing similarity with the query and the most similar documents then passed on for the second-stage pattern matching search, which is carried out using Horspool's version of the Boyer-Moore algorithm (Horspool, 1980). This search is used to produce the final ranking that is presented to the user. The similarity is calculated by means of the vector dot product, with the query terms being weighted using inverse document frequency (IDF) weighting. A basic data flow diagram for this process is shown in Figure 4.

The IDF-based similarity calculation that forms the basis for the final ranking is based on the matching of the actual stems that are present in the document and query texts. An analogous calculation can also be carried out during the signature search, although the calculated similarity value cannot be as accurate because less information is available (Mohan & Willett, 1985). If the bit or bits that are set in the query signature for some stem are also set in the document signature then the similarity is incremented by the IDF weight for that query stem; note that this procedure assumes the availability of a dictionary that contains the frequencies of occurrence of all of the words in the database (Croft & Savino, 1988). However, the many-to-one nature of the hashing procedures that are used in signature generation means that a match at the bit level indicates only the *possible* presence of the query stem in the document. Hence, the similarity which is calculated corresponds to the maximum possible similarity between the query and the document; the actual value will be somewhere between zero and this upperbound value.

We can then compare this calculated upperbound value with the actual similarity values that have already been calculated for earlier documents in the text scanning part of the search process. If the user wishes to see, e.g., twenty documents and the similarity value of the twentieth document in the ranking at that moment is higher than the upperbound calculated from the bit string of the document currently being considered, then there is no possibility that the current document can have a similarity value which would bring it into the ranking. There is therefore no need to send it forward for the pattern matching search. In order to implement this analysis of the bit strings, the threshhold value of similarity for the number of documents the user wishes to see must be passed back to the bit string search process, as shown in Figure 4, at intervals during the searching process.

4.3 Parallel implementation

As has already been stated, although the text signature search is very rapid, the second-level text searching stage is very time-consuming. It is this stage, therefore, for which we require a parallel solution. We have already stated that the processor farm model of distributing the computation is the most appropriate for this type of application. Therefore, the parallel implementation of text searching in a serial file can be represented by simply replacing the process called 'Compare texts' in

Figure 4 by the processor farm shown in Figure 1. Here, the two inputs to the farm are the required documents and the query stems, the work packet is the query and a document with which it is to be compared, and the required result is the text of a document for which the calculated similarity value is sufficiently large for it to be considered for inclusion in the final list of documents that is presented to the user at the end of the search. A physical implementation using a single chain processor farm is shown in Figure 5. The process 'Read required docs.' of Figure 4 is done before any processing takes place and the documents are stored in memory on a separate transputer board, which acts as the distributor and collector for the farm. All other processes are performed on the root transputer, with the text signatures also being read and stored in memory before the processing starts. The reader is referred to the papers by Cringean *et al.* (1989b, 1990) for a more detailed description of the logical processes required for serial text scanning and of the way in which these processes are implemented using a physical processor farm.

5 Efficiency Of Searching

The previous sections have described the implementation of serial text scanning on a transputer-based multicomputer system. This system is now operational in our laboratory and has been demonstrated at the Thirteenth International Online Information Meeting in London in December, 1989. In this section, we present the initial results from an ongoing study into the efficiency of retrieval that can be obtained from this system.

5.1 Measurement of performance

The measurement of computer performance is a controversial subject (even on conventional, serial processors) since the observed performance for an application is strongly affected by both the algorithm and the architecture (Hockney & Jesshope, 1988). Probably the most widely used measure of performance for parallel processors, and the one which is used in the following results, is an application-dependent one, the *speed-up*. The speed-up for P processors, S_P, is defined as

$$S_P = \frac{T_1}{T_P}, \tag{1}$$

where T_1 and T_P are the times to carry out an algorithm on one and P processors respectively. In the ideal case, $S_P = P$; all of the processors are fully utilized and the system is said to exhibit a *linear speed-up*, i.e., the speed-up varies directly with the number of processors that are available.

It must be emphasised that linear speed-up represents an upperbound to the performance of a parallel system: the actual degree of speed-up that can be obtained

is controlled, at least in part, by Amdahl's law (Amdahl, 1967; Gustafson, 1988; Quinn, 1987). This states that if a given problem has a fraction f of sequential operations, then the maximum speed-up which can be achieved with a parallel computer of P processors is

$$S_P \leq \frac{1}{(f + \frac{(1-f)}{P})}. \quad (2)$$

This suggests that, unless very little serial processing is required, the running time for a system with both serial and parallel operations will be dominated by the serial processing.

In the context of the transputer-based document retrieval system considered here, the requisite computation can be sub-divided as follows:

- Serial processing. The main serial component is the initial preprocessing of the query. Two other serial components are the first-stage scan of the text signatures, which is carried out on the root transputer, and the communication of the texts of the documents passing this scan over the network to the transputers in the farm. However, the fact that they are performed at the same time as the main parallel processing element makes it difficult to specify whether these are important as serial components.
- Parallel processing. This is the second-stage pattern matching search, where many document texts can be processed in parallel by the transputers in the farm.

The observed level of performance will thus be determined in large part by the relative proportions of these two types of processing.

The speed-up is a system-oriented measure of performance. Of more interest to the actual user of a retrieval system is the *response time*, i.e., the time that elapses between the submission of a query and the display of (hopefully) relevant documents at the terminal. It is thus of importance to note not only the speed-up relative to a single transputer but also the absolute time for a search.

5.2 Experimental results to date

The dataset used in our experiments contains the titles and abstracts of the 6,004 documents that formed the input to the 1982 issues of *Library and Information Science Abstracts* (LISA) database. Associated with these documents is a set of 35 natural language queries, which were collected from students and staff in the Department of Information Studies, University of Sheffield by Ms. A. Davies as part of her 1982 MSc dissertation project.

Timing figures were obtained for searches in the LISA dataset with the 35 queries. Four sets of runs were carried out with the single chain processor farm as shown in parts (a) to (d) of Table 1. The first of these used text signatures in which all of the bits in the document bit strings had been set to '1', so that none of the documents were eliminated in the first stage of the search and so that the maximum possible amount of work was done by the network. The other runs used 128-bit, 256-bit and 512-bit text signatures, so that progressively less pattern matching would need to be carried out by the farm, as shown by the figures at the bottom of the tables, giving the number of documents that matched the query at the signature level and were passed on for text searching. The text signatures were created by hashing the stem of each non-stopword in a document or query onto a single bit position in the bit string. The number of documents shown to the user at the end of the search also determines how much pattern matching needs to be done; this number was set at 1, 5, 10 or 20. When all of the documents undergo the text search, i.e., when all of the bits are set to '1', the run times are little affected by the number of documents that are displayed and thus just a single figure has been listed in Table 1(a), that for 10 documents.

It is clear from an inspection of the figures in Table 1 that the search time reduces as the number of transputers is increased. More importantly, the response times for the best results obtained here are consistent with the basic assumption underlying our work, *viz.* that parallel processing can provide a mechanism for interactive access to serial files of documents. A comparison of the figures in parts (b), (c) and (d) with those in part (a) demonstrates the great decrease in times when text signatures are used. This is not only because documents are eliminated in the first stage of the search but also because, even when documents are sent for text searching, some processing can be eliminated. This is possible because it is not necessary to search in the text of a particular document for query terms whose bit position is set to '0' in the text signature of that document. This effect can be observed in Table 2, which shows the reductions in the number of documents searched and in the search times compared with the searches in which all the bits were set to '1'. The greater reductions in search times compared to the reductions in the number of documents searched can be attributed to the fact that the second-stage text searching has been optimised to make as much use of the information in the text signatures as possible. This is a particularly effective strategy when large numbers of documents are required by the user; the reduction in search time is nearly twice the reduction in documents searched when 20 documents are required by the user.

A rather less satisfactory state of affairs is evident if we consider the speed-ups, rather than the response times. When no signatures are used, i.e., when the maximum amount of text searching takes place, then near-linear speed-up behaviour is observed for networks containing up to around eight or twelve transputers; thereafter, there is very little further increase in speed-up. This is somewhat disappointing since our previous simulation studies had suggested that near-linear speed-ups

should be obtainable with much larger farms than those used here (Cringean *et al.*, 1988). The speed-up behaviour becomes still more disappointing as longer and longer text signatures are used; indeed, there is very little speed-up at all when the 512-bit text signatures are used (although the actual response times here are very fast indeed).

It was thought that the problem could be a bottleneck in distributing the documents to the farm and that this might be alleviated by using the triple chain farm to reduce the average distance travelled by the messages, i.e., having three chains attached to the document store transputer instead of just the one shown in Figure 5. Four sets of runs were therefore carried out with a triple chain farm as shown in parts (a) to (d) of Table 3. The same parameters were used as for the previous runs except that the number of worker transputers was chosen so that all three chains were of equal length.

When all of the documents undergo the text search, i.e., when all of the bits are set to '1', the run times are longer with small numbers of transputers than in the case of the single chain because the use of the triple chain introduces overheads associated with the distribution of the documents. However, the response times when the farm is large are a considerable improvement on the single chain results. In this case, the speed-up is much more satisfactory than before. With a network of 15 worker transputers it is possible to get a speed-up of nearly 13, which is fairly close to the ideal linear speed-up. This verifies our assumption that the previous disappointing results had been due to a distribution bottleneck.

However, the speed-up behaviour is still disappointing when long text signatures are used; when the 512-bit text signatures are used there is again very little speed-up at all and little improvement in the response times over the single chain farm.

One point that should be made about these results is that they have been obtained in nearest neighbour searches where the natural language queries have been converted to a set of right-hand truncated word stems. The signature generation method used has been optimised for efficient searching using this particular type of query representative, and the signature search is thus very successful in eliminating documents from the pattern matching search. However, these signatures could not be used to eliminate documents if other types of pattern matching needed to be carried out, e.g., searches for left-hand truncated terms, embedded don't care characters and arbitrary substrings. Accordingly, very many more documents would have to be passed on for pattern matching, with a consequent substantial improvement in the speed-up behaviour. The results in Table 1 thus represent a lower bound for the speed-up which can be obtained when such text signatures are used.

We would also expect to obtain greater speed-up than those obtained to date if a more appropriate type of signature for these sorts of pattern matching operations were to be used, e.g., one based on the presence of trigrams. Text signatures based

on the trigrams of non-stopwords were created by hashing each trigram to a single bit position in the bit string. A bit was also set for each word stem, as before. The length of the bit string was chosen so that approximately half of the bits were set to '1', since this is theoretically the most efficient arrangement; thus, the bit string length was 352. Searches were then performed using the original queries and modified queries in which wildcard characters were inserted at appropriate places, usually at the end of query terms. For words which did not contain wildcard characters, the appropriate IDF weighting was calculated as before: for words containing wildcard characters, an estimated IDF weight was calculated by assuming the term frequency was the average frequency of the terms in the document collection. This means that there is a graceful degradation in retrieval effectiveness from the case when no terms contain wildcards to the case when all terms contain them.

Table 4 contains the results of searches using these trigram text signatures. It is evident from comparison of Tables 4(a) and 4(b) that, as expected, both the response times and the speed-ups increase when wildcards are used in queries. It can also be observed that the response times are much greater than with the previous text signatures and the speed-ups are much lower. This was, at first, surprising because we had expected at least an improvement in speed-up because more text searching is being carried out (as is shown by a comparison of the numbers of documents searched here and in Table 3(c), where 256-bit signatures are used). However, it has now been realised that, by using text signatures, we have created another bottleneck. The original bottleneck was caused by a problem in distributing the documents from the document store transputer. Now we have a bottleneck in getting the document numbers sent out from the root transputer to the document store transputer quickly enough. With the trigram signatures, many more bit positions have to be checked for each term than previously. Thus, there is a greater delay between sending out the document numbers of those documents which need to be searched by the processor farm. Accordingly, the farm cannot be used to full effect. This could only be counteracted by having several processors searching the text signatures simultaneously and feeding document numbers into the document store transputer. A method of distributing text signature searching over several processors has been described by Walden and Sere (Walden & Sere, 1989).

A further unexpected, and possibly related, result is that there is no consistent trend in the variation of speed-up as the number of documents shown to the user is varied. This was again rather surprising, since, as more documents need to be displayed, so more documents need to be sent for text searching, and previous experience would have suggested to us that the speed-up should increase. However, with all the above results, as expected, the search time reduces as the number of documents required by the user decreases; this is because there is a higher threshhold similarity value when fewer documents are required.

6 Conclusions

The main conclusions of our experiments are as follows:

- The use of a transputer-based system allows very rapid searches based on serial files to be carried out, with response times that are comparable to those obtainable from PC-based inverted file systems.
- At least some speed-up has been obtained using parallel hardware under all the conditions investigated here.
- If no documents are eliminated at the first stage of the search, near-linear speed-up can be achieved with up to fifteen transputers.
- Large text signatures significantly reduce the search time, but there is also a marked reduction in the speed-up that can be achieved.
- The use of a single chain processor farm resulted in a bottleneck in the distribution of documents to the farm; this was alleviated by use of a triple chain network.
- The use of trigram signatures increases the response time and reduces the speed-up because it introduces a delay and thus creates a new bottleneck in the system.

Our results would thus suggest that transputer networks have at least some potential for allowing PCs to be used for interactive nearest neighbour searching in serial document files. That said, much further work is required to identify how such networks should be used to maximise the efficiency of searching. At present, we are investigating alternative ways of distributing the processes over the transputers at the top of the farm.

Acknowledgements. We thank the British Library for funding this work under grant number SI/G/814, the Library Association for the provision of the LISA data, and Mr. Andy Jackson, Mr. Glenn Miller, Mr. Geraint Jones and Dr. Jon Kerridge of the National Transputer Support Centre for technical support.

References

Amdahl, G. (1967). The validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings, 30*, 483-485.

Athas, W.C. & Seitz, C.L. (1988). Multicomputers: message-passing concurrent computers. *Computer, 21*(8), 9-24.

Carroll, D.M., Pogue, C.A. & Willett, P. (1988). Bibliographic pattern matching using the ICL Distributed Array Processor. *Journal of the American Society for Information Science, 39*, 390-399.

Cringean, J.K., Manson, G.A., Willett, P. & Wilson, G.A. (1988). Efficiency of text scanning in bibliographic databases using microprocessor-based multiprocessor networks. *Journal of Information Science, 14*, 335-345.

Cringean, J.K., Lynch, M.F., Manson, G.A., Willett, P. & Wilson, G.A. (1989a). Parallel processing techniques for information retrieval. Searching of textual and chemical databases using transputer networks. *Proceedings of the Thirteenth International Online Information Meeting*, 447-462.

Cringean, J.K., England, R., Manson, G.A. & Willett, P. (1989b). *Best Match Searching in Document Retrieval Systems Using Transputer Networks*. London: British Library Research and Development Department.

Cringean, J.K., England, R., Manson, G.A. & Willett, P. (1990). Implementation of text scanning using a multicomputer network. Part I. System design. Submitted for publication.

Croft, W.B. & Savino, P. (1988). Implementing ranking strategies using text signatures. *ACM Transactions on Office Information Systems, 6*, 42-62.

Dubois, M., Scheurich, C. & Briggs, F.A. (1988). Synchronisation, coherence and event ordering in multiprocessors. *Computer, 21*(2), 9-21.

Faloutsos, C. (1985). Access methods for text. *ACM Computing Surveys, 17*, 49-74.

Flynn, M.J. (1972). Some computer organisations and their effectiveness. *IEEE Transactions on Computers, C-21*, 948-960.

Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K. & Walker, D.W. (1988). *Solving Problems on Concurrent processors. Volume I. General Techniques and Regular Problems* Englewood Cliffs: Prentice Hall.

Gajski, D.D. & Peir, J.K. (1985). Essential issues in multiprocessor systems. *Computer, 18*(6), 9-27.

Gustafson, J.L. (1988). Reevaluating Amdahl's Law. *Communications of the ACM, 31*, 532-533.

Handler, W. (1977). The impact of classification schemes on computer architectures. In *Proceedings of the 1977 International Conference on Parallel Processing* (pp. 7-15). New

York: IEEE.

Hey, A.J.G. (1988). Reconfigurable transputer networks: practical concurrent computation. *Proceedings of the Royal Society, A326*, 395-410.

Hockney, R.W. & Jesshope, C.R. (1988). *Parallel Computers 2. Architecture, Programming and Algorithms.* Bristol: Adam Hilger.

Horspool, R.N. (1980). Practical fast matching in strings. *Software - Practice and Experience, 10*, 501-506.

May, D. & Taylor, R. (1984). occam - an overview. *Microprocessors and Microsystems, 8*, 73-79.

Mohan, K.C. & Willett, P. (1985). Nearest neighbour searching in serial files using text signatures. *Journal of Information Science, 11*, 31-39.

Patton, P.C. (1985). Multiprocessors: architecture and applications. *Computer, 18*(6), 29-40.

Pritchard, D.J., Askew, C.R., Carpenter, D.B., Glendinning, I., Hey, A.J.G. & Nicole, D.A. (1987). Practical parallelism using transputer networks. *Lecture Notes in Computer Science, 258*, 278-294.

Quinn, M.J. (1987). *Designing Efficient Algorithms for Parallel Computers.* New York: McGraw-Hill.

Reed, D.A. & Fujimoto, R.M. (1987). *Multicomputer Networks: Message-Based Parallel Processing.* Cambridge, MA: MIT Press.

Salton, G. (1989). *Automatic Text Processing: the Transformation, Analysis and Retrieval of Information by Computer.* Reading, MA: Addison-Wesley.

Salton, G. & Buckley, C. (1988). Parallel text search methods. *Communications of the ACM, 31*, 202-215.

Sharma, R. (1989). A generic machine for parallel information retrieval. *Information Processing and Management, 25*, 223-235.

Shore, J.E. (1973). Second thoughts on parallel processing. *Computers and Electrical Engineering, 1*, 95-109..

Skillicorn, D.B. (1988). A taxonomy for computer architectures. *Computer, 21*(11), 46-57.

Stewart, M. & Willett, P. (1987). Nearest neighbour searching in binary search trees: simulation of a multiprocessor system. *Journal of Documentation, 43*, 93-111.

Walden, M. & Sere, K. (1989). Free text retrieval on transputer networks. *Microprocessors and Microsystems, 13*, 179-187.

Waltz, D., Stanfill, C., Smith, S. & Thau, R. (1987). Very large database applications of the Connection Machine system. *AFIPS Conference Proceedings, 56*, 159-165.

Willett, P. & Rasmussen, E.M. (1990). *Parallel Database Processing. Text Retrieval and Cluster Analysis using the Distributed Array Processor.* London: Pitman.

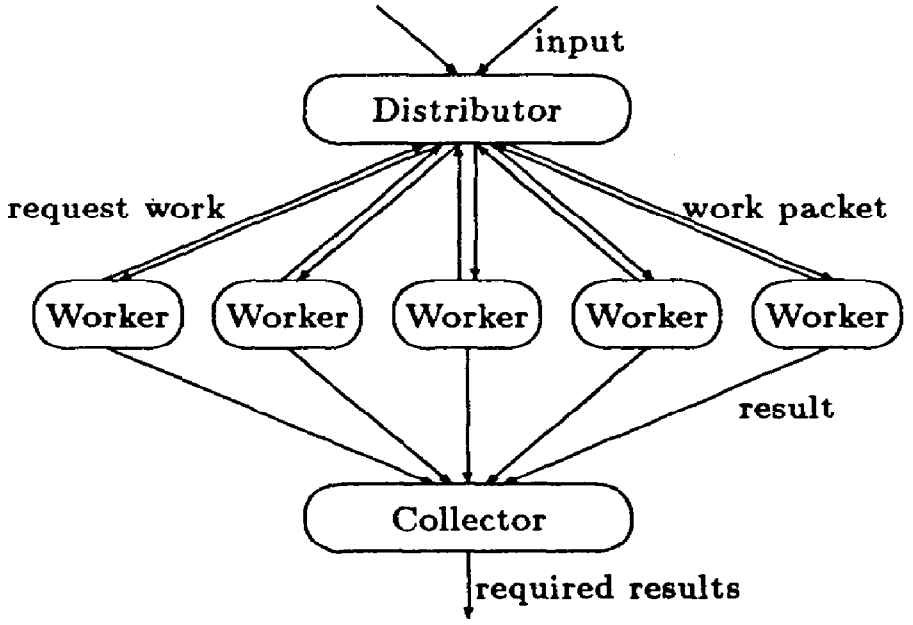

Figure 1: Flow diagram of processor farm with five worker processors. Rounded boxes indicate processes.

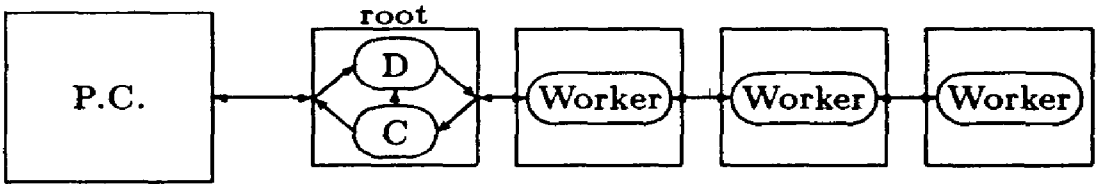

Figure 2: Single linked chain network with both the distributor process (D) and the collector process (C) on the root processor. Rounded boxes indicate processes and squares indicate physical processors.

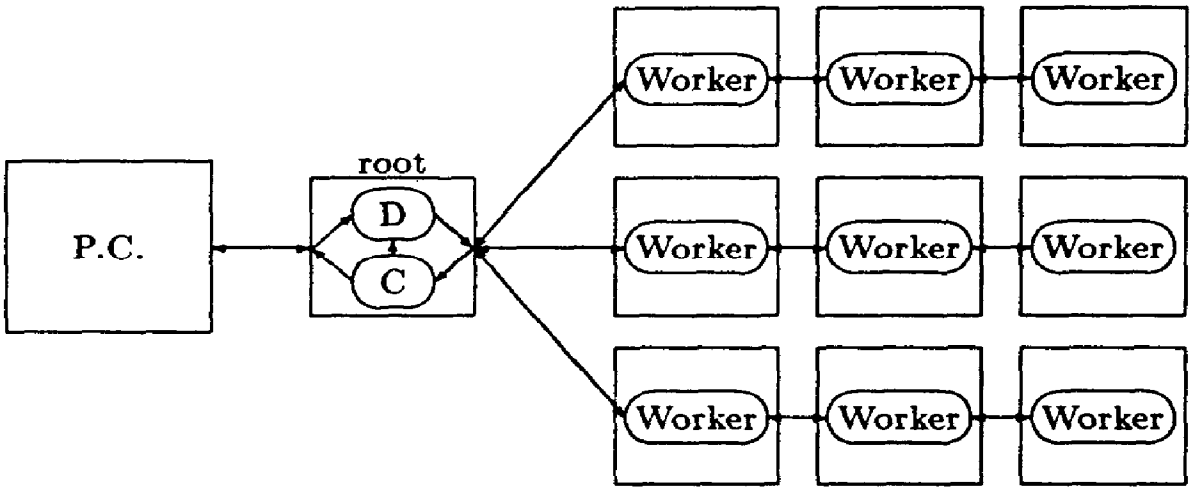

Figure 3: Triple chain network with both the distributor process (D) and the collector process (C) on the root processor. Rounded boxes indicate processes and squares indicate physical processors.

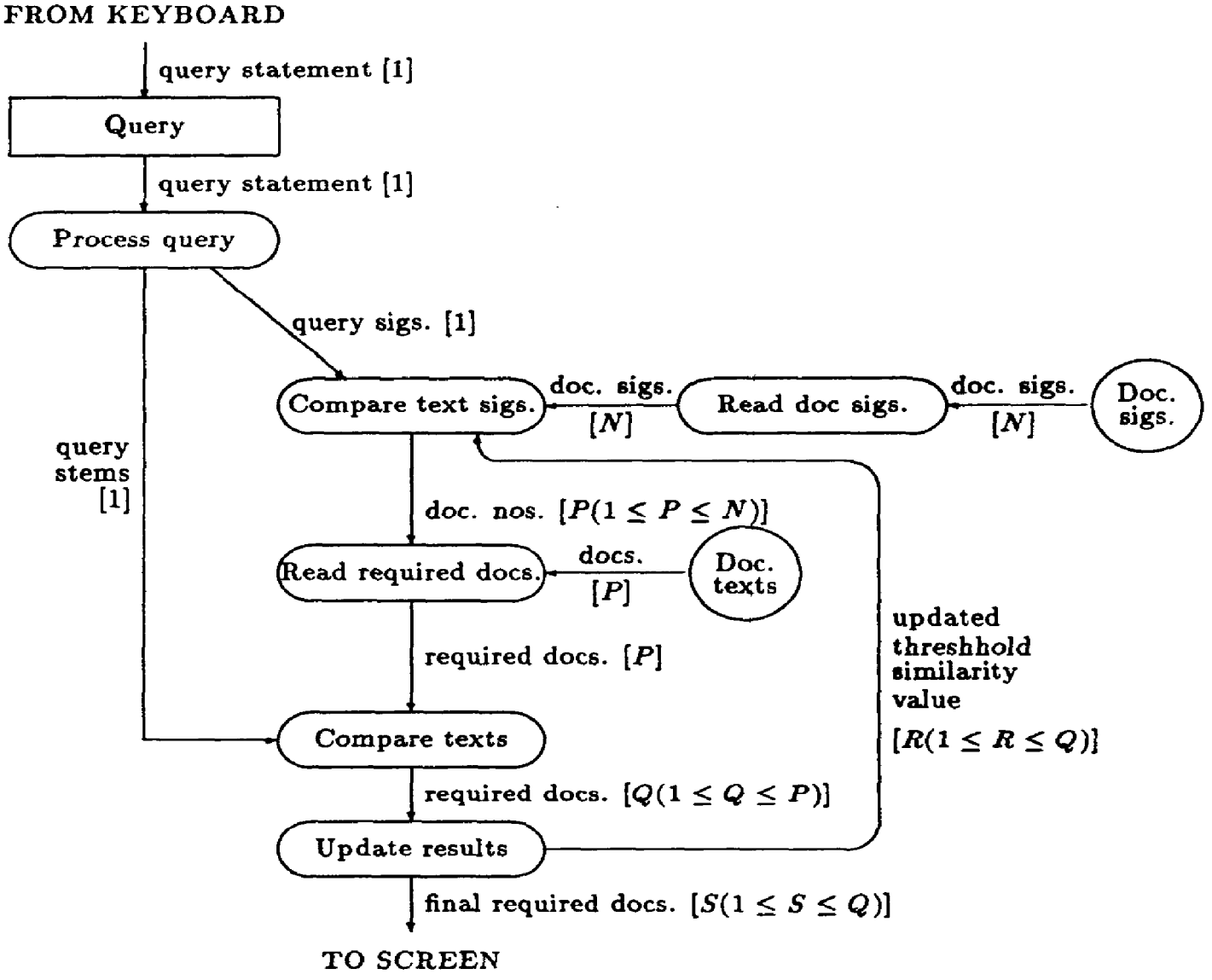

Figure 4: Basic flow diagram of two-level text searching in a serial file. Circles indicate disk storage media, rounded boxes indicate processes, and the rectangle indicates data input to the system from external sources; numbers in square brackets represent the number of times data units are transferred from one process to another.

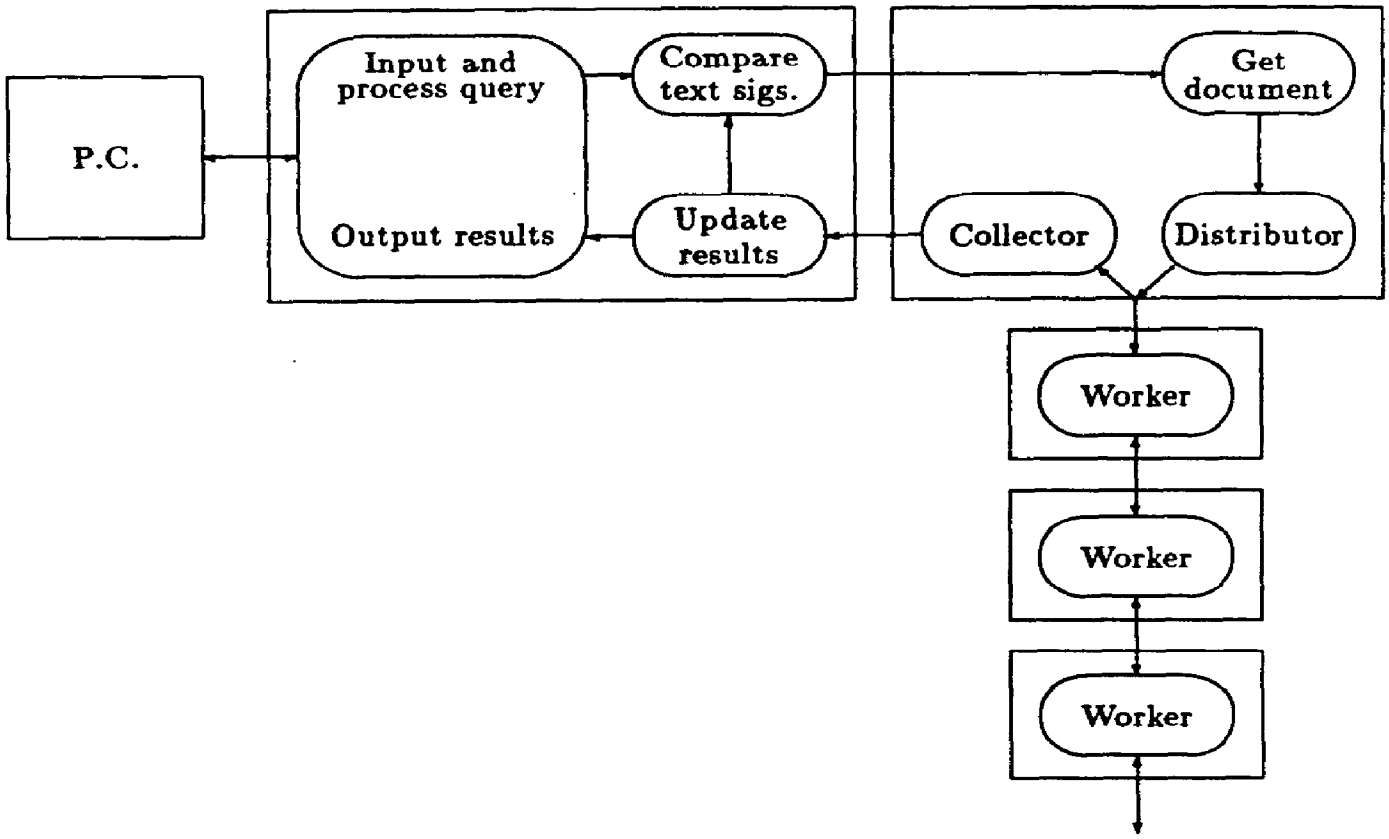

Figure 5: Physical implementation of parallel text searching using a single chain processor farm with processes placed on processors in a transputer network. Rounded boxes indicate processes and rectangles indicate processors.

Number of workers (P)	T_P	S_P
1	125.6	–
2	63.5	2.0
4	32.8	3.8
8	18.3	6.9
12	14.9	8.4
16	14.2	8.8

(a) No text signatures used (100% of 6004 documents searched)

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	24.3	–	32.4	–	35.5	–	39.3	–
2	12.5	1.9	16.9	1.9	18.9	1.9	21.9	1.8
4	6.8	3.6	9.5	3.4	11.1	3.2	13.8	2.8
8	5.2	4.7	7.6	4.2	9.2	3.9	12.1	3.2
12	5.3	4.6	7.6	4.2	9.2	3.9	12.1	3.2
16	5.3	4.6	7.7	4.2	9.3	3.8	12.2	3.2
Docs. searched	1976 (33%)		2934 (49%)		3307 (55%)		3680 (61%)	

(b) 128-bit text signatures

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	6.8	–	12.0	–	14.7	–	18.7	–
2	4.0	1.7	6.8	1.8	8.6	1.7	11.7	1.6
4	3.1	2.2	4.7	2.6	6.1	2.4	8.9	2.1
8	3.1	2.2	4.6	2.6	6.0	2.4	8.8	2.1
12	3.1	2.2	4.6	2.6	6.0	2.4	8.8	2.1
16	3.1	2.2	4.7	2.6	6.1	2.4	8.9	2.1
Docs. searched	613 (10%)		1222 (20%)		1521 (25%)		1883 (31%)	

(c) 256-bit text signatures

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	2.8	–	4.5	–	6.0	–	9.0	–
2	2.6	1.1	3.7	1.2	4.9	1.2	7.3	1.2
4	2.6	1.1	3.6	1.2	4.7	1.3	7.1	1.3
8	2.6	1.1	3.6	1.2	4.7	1.3	7.1	1.3
12	2.7	1.0	3.6	1.2	4.7	1.3	7.1	1.3
16	2.7	1.0	3.6	1.2	4.8	1.2	7.2	1.2
Docs. searched	149 (2%)		387 (6%)		540 (9%)		751 (12%)	

(d) 512-bit text signatures

Table 1: Mean search time in seconds, T_P, and speed-up, S_P, for searching the LISA dataset using a single chain processor farm (averaged over 35 queries).

Number of docs. shown	*RD*	*RT*
1	67%	81%
5	51%	74%
10	45%	72%
20	39%	69%

Table 2: Mean percentage reduction in documents searched, RD, and reduction in time, RT, for searching with 1 worker transputer in the LISA dataset (averaged over 35 queries) using 128-bit text signatures (compared to times when all bits set to '1').

Number of workers (P)	T_P	S_P
1	137.0	–
2	68.8	2.0
3	46.1	3.0
6	23.9	5.7
9	16.6	8.3
12	13.0	10.5
15	10.9	12.6

(a) No text signatures used (100% of 6004 documents searched)

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	28.3	–	37.9	–	41.4	–	45.5	–
2	14.3	2.0	19.3	2.0	21.4	1.9	24.3	1.9
3	9.7	2.9	13.1	2.9	14.8	2.8	17.4	2.6
6	5.2	5.4	7.2	5.3	8.5	4.9	11.0	4.1
9	3.9	7.3	5.4	7.0	6.6	6.3	9.1	5.0
12	3.6	7.9	4.8	7.9	6.0	6.9	8.5	5.4
15	3.6	7.9	4.8	7.9	5.9	7.0	8.4	5.4
Docs. searched	2010 (34%)		2985 (50%)		3366 (56%)		3756 (63%)	

(b) 128-bit text signatures

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	7.9	–	14.0	–	17.0	–	21.4	–
2	4.4	1.8	7.5	1.9	9.3	1.8	12.6	1.7
3	3.4	2.3	5.5	2.5	6.9	2.5	9.9	2.2
6	2.8	2.8	4.0	3.5	5.1	3.3	7.8	2.7
9	2.8	2.8	3.8	3.7	4.9	3.5	7.6	2.8
12	2.8	2.8	3.8	3.7	4.9	3.5	7.6	2.8
15	2.8	2.8	3.8	3.7	4.9	3.5	7.6	2.8
Docs. searched	613 (10%)		1231 (20%)		1542 (26%)		1910 (32%)	

(c) 256-bit text signatures

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	2.9	–	4.9	–	6.5	–	9.8	–
2	2.6	1.1	3.7	1.3	4.8	1.4	7.5	1.3
3	2.6	1.1	3.5	1.4	4.6	1.4	7.1	1.4
6	2.6	1.1	3.5	1.4	4.5	1.4	7.0	1.4
9	2.6	1.1	3.5	1.4	4.5	1.4	7.0	1.4
12	2.6	1.1	3.5	1.4	4.5	1.4	7.0	1.4
15	2.6	1.1	3.5	1.4	4.5	1.4	7.0	1.4
Docs. searched	149 (2%)		390 (6%)		545 (9%)		760 (13%)	

(d) 512-bit text signatures

Table 3: Mean search time in seconds, T_P, and speed-up, S_P, for searching the LISA dataset using a triple chain processor farm (averaged over 35 queries).

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	13.5	–	18.8	–	21.5	–	25.5	–
2	9.9	1.4	11.6	1.6	13.1	1.6	15.9	1.6
3	9.6	1.4	10.6	1.8	11.7	1.8	14.2	1.8
6	9.6	1.4	10.5	1.8	11.6	1.9	14.0	1.8
9	9.6	1.4	10.5	1.8	11.6	1.9	14.0	1.8
12	9.6	1.4	10.5	1.8	11.6	1.9	14.0	1.8
15	9.6	1.4	10.5	1.8	11.6	1.9	14.0	1.8
Docs. searched	653 (11%)		1172 (20%)		1421 (24%)		1722 (29%)	

(a) No wildcards in queries

Number of workers (P)	Number of documents shown to user							
	1		5		10		20	
	T_P	S_P	T_P	S_P	T_P	S_P	T_P	S_P
1	15.6	–	21.6	–	24.9	–	29.1	–
2	10.7	1.5	12.7	1.7	14.2	1.8	17.0	1.7
3	10.1	1.5	11.1	1.9	12.2	2.0	14.5	2.0
6	10.1	1.5	10.9	2.0	11.9	2.1	14.1	2.1
9	10.0	1.6	10.9	2.0	11.9	2.1	14.1	2.1
12	10.1	1.5	10.9	2.0	11.9	2.1	14.0	2.1
15	10.0	1.6	10.9	2.0	11.9	2.1	14.1	2.1
Docs. searched	692 (12%)		1182 (20%)		1438 (24%)		1722 (29%)	

(b) Wildcards in queries where appropriate

Table 4: Mean search time in seconds, T_P, and speed-up, S_P, for searching the LISA dataset with trigram text signatures using a triple chain processor farm (averaged over 35 queries).