

---

# Nearest-neighbour searching in files of text signatures using transputer networks

JANEY K. CRINGEAN  
*Department of Information Studies  
University of Sheffield  
Western Bank  
Sheffield, S10 2TN, UK*

ROGER ENGLAND  
*National Transputer Support Centre  
Sheffield Science Park  
60 Howard Street  
Sheffield S1 2LX, UK*

GORDON A. MANSON  
*Department of Computer Science  
University of Sheffield*

PETER WILLETT<sup>1</sup>  
*Department of Information Studies  
University of Sheffield*

---

## SUMMARY

**This paper discusses the implementation of nearest-neighbour document retrieval in serial files using transputer networks. The system uses a two-stage retrieval algorithm in which an initial text-signature search is used to exclude large numbers of documents from the detailed and time-consuming pattern-matching search. The latter is implemented using a processor farm, so that documents which match at the signature level can be examined in parallel to determine whether they are, in fact, a good match for the query. The results demonstrate that communication is the critical factor in all of the transputer networks that were investigated. A high degree of speed-up can be obtained when only the pattern-matching search is carried out. When text signatures are used, however, the speed-up is less, decreasing in line with an increase in the size of the text signatures that are used.**

**KEY WORDS** Best-match searching Full-text documents Geometric parallelism Information retrieval Nearest-neighbour searching Parallel processing Processor farm Text signature Transputer network

## 1 SERIAL TEXT RETRIEVAL

Early text retrieval systems used serial files, in which the text of a query was matched with the texts of each of the documents in a database using a pattern-matching algorithm. There are several efficient pattern-matching algorithms available (see, for example, Reference [1]), but serial scanning is very demanding of computational resources if large files of text need to be searched, and nearly all modern text retrieval systems thus use inverted files (see, for example, References [2–4]). These provide a sufficiently fast response for interactive searching, but require a considerable amount of additional storage for the indexes that need to be set up to provide rapid access to the database. Moreover, the generation and maintenance of these indexes incur substantial overheads, since index generation can involve massive sorting operations and since updating involves the modification of the index for each author name, keyword, classification code, etc., in a new document. A further problem is the wide range of pattern-matching operations needed for text retrieval. Some of these are difficult to implement efficiently using conventional inverted-file systems, e.g., embedded substring, left-hand truncation and both fixed-length and variable-length don't care searches, while others are very demanding of additional storage, e.g., the complex

---

<sup>1</sup> To whom all correspondence should be addressed.

---

positional information that needs to be stored for the efficient implementation of proximity and adjacency searches.

The limitations of inverted-file systems have led to a resurgence of interest in techniques that can increase the efficiency of serial text searching [5]; in particular, there have been many studies of a modification of the basic serial-file structure known as a *text signature* file. A text signature is a bit string, usually of fixed length, in which bits are set by applying some hashing-like operation to the keywords or descriptors that characterize a document [6,7]. Once a file of signatures has been created, queries that are presented to the database may be searched by comparing the query signature with the set of document signatures. This is normally done by a serial scan of the document signatures. The bit-parallel, word-serial nature of conventional processors means that matching operations upon strings of bits can generally be implemented very much more efficiently than can operations upon strings of characters, and thus signature searching is extremely rapid, despite the fact that all of the document signatures need to be inspected in the search. Further economies of processing arise from the fact that the signatures are very much smaller than the source documents from which they have been generated, and thus, during the course of a search, less data needs to be transferred into the processor from backing storage than would be the case if the entire file of document texts had to be inspected. The use of a hashing function means, however, that matches may occur at the bit level that do not correspond to matches at the word level. The elimination of these mismatches requires the use of a second-level, pattern-matching search, but this is carried out only for those documents that match at the signature level (so that the time-consuming pattern matching needs to be applied to only a small number of documents).

Even with the development of text signatures and fast pattern-matching algorithms, serial text scanning is very time-consuming when implemented on a conventional processor. There has accordingly been increasing interest in the use of *parallel processing* for text scanning, where a parallel processor is one in which the constituent processors operate together so as to reduce the elapsed time required for a computational task; parallel processing techniques can also, of course, be used to increase the efficiency of inverted-file processing (see, for example, References [8,9]). In this paper, we discuss the efficiency of text scanning achievable with one particular type of parallel processor, a transputer network. Section 2 introduces the transputer, and describes how networks of transputers can be used for database searching applications. The reader should note that this section provides only a summary account of the logical and physical characteristics of transputer-based searching systems: these topics are dealt with in much greater detail in a previous paper [10]. Sections 3 and 4 describe the experiments that were carried out to evaluate the efficiency of transputer networks for serial nearest-neighbour document retrieval. Section 5 considers ways in which the networks' current capabilities could be enhanced, and the paper concludes with a summary of our main findings.

## 2 THE TRANSPUTER

### 2.1 Transputer hardware and software

There have been several attempts to classify the many ways in which parallelism may be implemented in a computer system (see, for example, References [11–13]): in this paper, we

---

are concerned with what are generally referred to as *multiple instruction stream, multiple data stream*, or *MIMD* computers [11], which are characterized by multiple processors capable of independent operation [14–16]. The most widely applied sub-categorization 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; facilities are thus required for highly efficient inter-processor communication. There are several ways in which message-passing MIMD computers can be built: those studied here are examples of what are often referred to as *multicomputers, multicomputer networks* or *microprocessor-based multiprocessors*. These consist of a number of microprocessors, each of which has some local memory and which can communicate with other such processors over a network [17,18].

The multicomputer considered here is based on the INMOS Transputer. The transputer is a generic name for a family of high-performance devices that have been developed specifically for use in multiprocessor systems. At the heart of a transputer is a high-performance (up to 25 MIPS) 16-bit or 32-bit CPU, including hardware support for very fast process switching that allows simulated concurrency on a single processor. The processor has a limited amount of fast RAM, typically of size 4 Kbytes, and interfaces to external memory units. In addition, and most importantly, there is some number (typically two or four) of serial communication link processors, for bidirectional communication between pairs of devices within the transputer family. The link processors operate concurrently with the CPU so that, for example, the CPU might be processing one item of data, one link transferring the next item of data from disk to memory, and a further link transferring previous calculated results from memory to disk. The links can achieve data transfer rates up to 1.7 Mbytes/second unidirectionally, and 2.3 Mbytes/second bidirectionally, thus allowing the implementation of parallel algorithms with a fair degree of granularity. The links 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 such a multiprocessing system, thus overcoming the classic von Neumann bottleneck of limited bandwidth which is often encountered in bus-based multiprocessors.

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`,<sup>2</sup> which has been developed in conjunction with the transputer 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 processor shares its time between the concurrent processes; channel communication is

---

<sup>2</sup> `occam` is a trademark of the INMOS Group of Companies.

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.

There have been several reports of the use of transputer networks for text searching [19–22], as well as for a range of other types of information retrieval application, including image databases [23,24], cartographic databases [25], chemical structure databases [26,27], and protein sequence databases [28].

## 2.2 Database searching using transputer networks

There are many ways in which transputer networks can be used to provide a distributed solution to some computational problem. One of the most common approaches involves the use of a *processor farm*, or *processor pool* [29,30]. A processor farm comprises a *root* processor and a number of *worker*, *slave* or *task* processors. Computational work is distributed to the worker processors by the root processor, which is responsible for collecting intermediate results and for communicating with the host machine for the network. Cringean *et al.* [19] have compared the processor farm with other models of distributed computation and suggest that it is the most appropriate model for database searching applications that use serial files. The farm is well suited to such applications since the root processor can be used to distribute data to the farm, where the individual query–document matching operations can be implemented concurrently by all of the worker processors.

Three main processes are involved in the implementation of a text retrieval system on a processor farm. These are:

- A *Distribution* process, which sends data packets into the farm for processing by the first worker processor that is not already engaged in computational work.
- A *Collection* process, which collects the results of individual query–document matching operations once they have been carried out by worker processors in the farm.
- A *Worker* process, which is located on each worker processor and which compares an individual query–document pair. This process also contains additional processes to route data packets to worker processors elsewhere in the farm and to route results packets back to the root processor.

A group of transputers can be linked together to form a processor farm in many ways, subject only to the constraint that none of them can be connected to more than four other transputers (since current transputers have only four linkage processors). The work reported here used a *triple-chain* network, which consists of three chains that are linked to the root processor. Each of these chains consists of a simple, linear array of linked transputers so that some transputer,  $T_j$ , in the body of a chain of  $n$  transputers is connected to two other transputers,  $T_j - 1$  and  $T_j + 1$  ( $1 < j < n$ ). When the chain is used in a processor farm, data is passed down the chain in one direction and results are passed up the chain in the opposite direction along each transputer's bidirectional links. Our previous studies showed that the triple-chain network gave a higher level of performance than when just a single chain of transputers is used: a triple-chain network not only allows full utilization of the four links that are available on the root transputer but also reduces the average distance

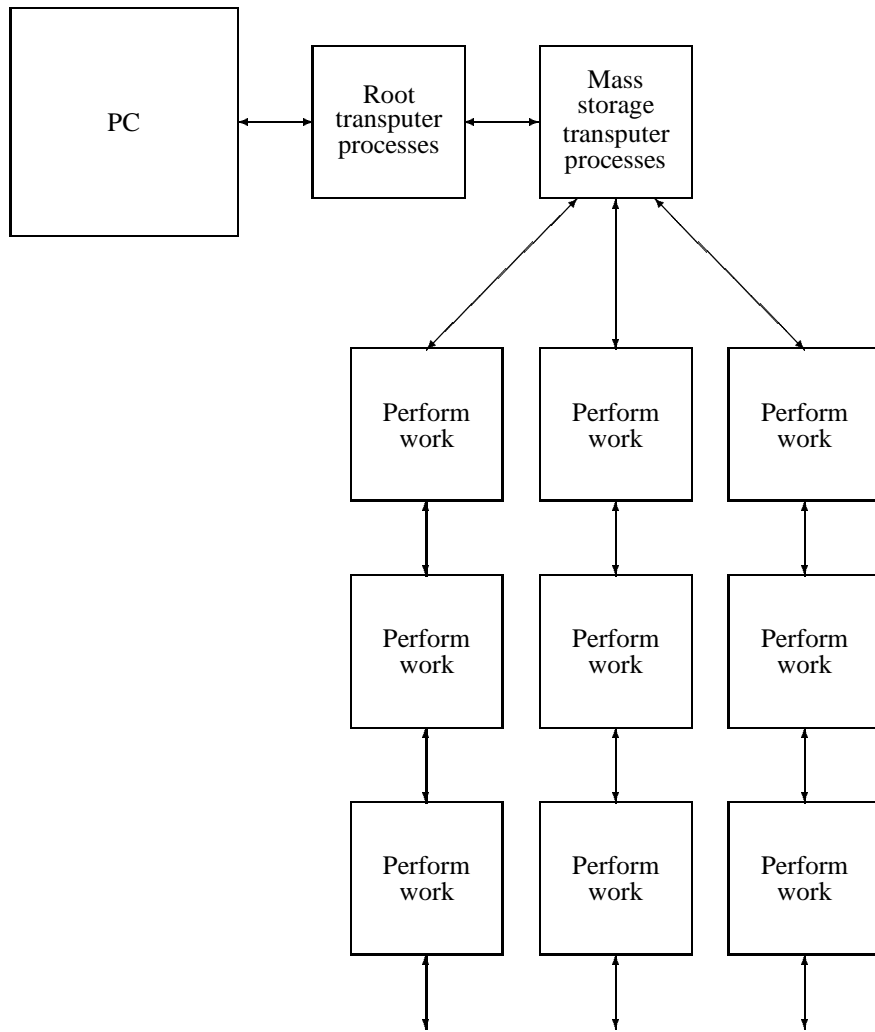


Figure 1. Triple chain transputer network configuration for parallel text searching. Rectangles represent processors. Connections between processors represent bidirectional transputer links, with the arrows indicating whether messages are passed in one or both directions

that must be travelled by the data packets as they move through the network to and from the root processor [10].

A schematic diagram of a triple chain network is shown in Figure 1, and contains the host PC, the root transputer, the worker transputers that carry out the matching of the document and query texts in parallel, and a mass storage transputer that contains the document texts. The model MK021 Mass Store that was used for this purpose is a board that incorporates a T800 transputer with 8 Mbytes of fast RAM. The MK021 is part of the Meiko model M40 Computing Surface, which has been used in all of our experiments and which has been described in detail in our previous paper [10].

### 3 EXPERIMENTAL DETAILS

#### 3.1 Implementation of nearest-neighbour searching

Our transputer system has been designed to evaluate the efficiency of nearest-neighbour retrieval using text signatures. The basic flow diagram for this system is shown in Figure 2. When a user submits a query, the following processes are executed:

1. The words in the query are compared with a stopwords list to delete commonly occurring words and the remaining, content-bearing words are then stemmed to conflate morphological variants.
2. The stems are hashed into a bit string to form the query signature; this is matched with each of the document signatures.

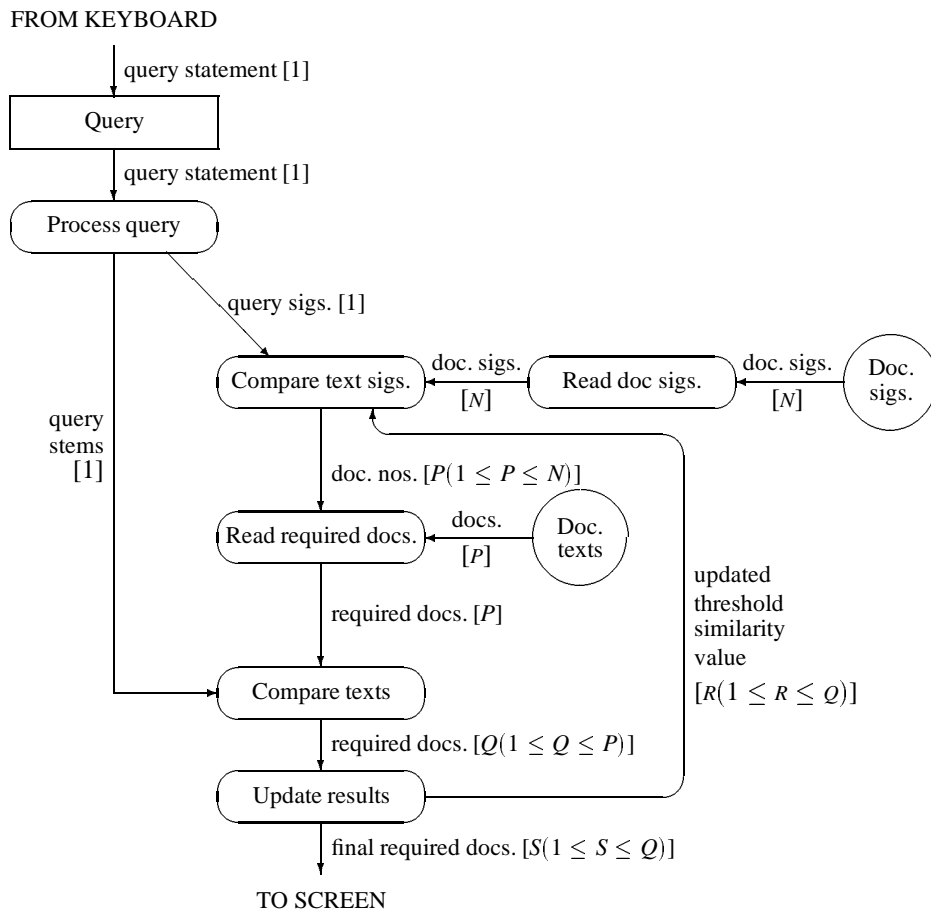


Figure 2. Basic flow diagram of two-level text searching in a serial file. Circles indicate disk storage media, rounded boxes indicate processes, and the rectangle indicates data input to the system from external sources; numbers in square brackets represent the number of times data units are transferred from one process to another

3. The most similar documents from [Step 2](#) are passed on for the second-stage search.
4. The pattern-matching search is carried out on those documents that result from [Step 3](#); this search is implemented using Horspool's version of the Boyer-Moore algorithm [31] and the results of the pattern matching are used to produce the ranking of the documents that is presented to the searcher.

A more detailed discussion of the processes in [Figure 2](#) is presented by Cringean *et al.* [10].

The ranking in [Step 4](#) is based on similarities that are calculated by means of the vector dot product between the sets of document and query terms, with the latter being weighted using inverse document frequency (IDF) weighting. This ranking derives from similarities that are calculated after the matching of the actual stems present in the document and query texts. An analogous calculation is carried out during the first-stage text-signature search, although the similarities calculated here cannot be as accurate because less information is available [10]. The similarities are used in the 'Update results' process of [Figure 2](#), which keeps a note of the best-matching documents identified at each stage of a search and which is discussed in detail in [Section 4.3](#).

We have noted in [Section 1](#) that pattern matching ([Step 4](#) above) is very demanding of computational resources and this provided the rationale for the development of the distributed implementation described by Cringean *et al.* [10]. Specifically, the parallel implementation of serial nearest-neighbour searching can be represented simply by replacing the process called 'Compare texts' in [Figure 2](#) by a processor farm. Here, the work packet sent out by the root processor is a document which is to be compared with the query (which is sent out to the processors in the farm once it has been submitted by the searcher), and the required result collected by the root processor 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. The process 'Read required docs.' of [Figure 2](#) 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.

### 3.2 Measurement of performance

The most widely used measure of performance for parallel processors, and the one that is used here, is an application-dependent one, the *speed-up*. The speed-up for  $P$  processors,  $S_p$ , is defined as

$$S_p = \frac{T_1}{T_p} \quad (1)$$

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 emphasized that linear speed-up represents an upper bound 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 [32,33]. 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 text retrieval system considered here, the requisite computation can be subdivided as follows:

- Serial processing. The main serial component is the initial preprocessing of the query. Two other serial components are the initial search of the text signatures, which is carried out on the root transputer, and the communication of the texts of the documents passing the signature search over the network to the worker transputers in the farm. However, the fact that these latter components are performed at the same time as the main parallel processing component makes it difficult to specify, *a priori*, 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. In fact, as discussed in Section 4.3, we have also considered a parallel implementation of the first-stage, text-signature search.

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 was thus of importance to note not only the speed-up relative to a single transputer but also the absolute time for each of the searches.

### 3.3 Dataset

The dataset used in our experiments contains the titles and abstracts of the 6004 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 these 35 queries: the execution times listed in the tables of results are mean values, averaged over this query set.

## 4 EXPERIMENTAL RESULTS

### 4.1 Use of word-based signatures

The first set of experiments used text signatures that were created by hashing the stem of each content-bearing word in a document or query onto a single bit position in the bit string. The results of these experiments are listed in parts (a) to (d) of Table 1. The number of worker transputers was chosen so that all three chains in the triple chain were of equal length.

Table 1. Mean search time in seconds,  $T_P$ , and speed-up,  $S_P$ , for searching using word-based text signatures

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

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

(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	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%)	

(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	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%)	

(d) 512-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%)	

Table 2. Mean search times in seconds,  $T_P$ , and speed-ups,  $S_P$ , using trigram-based text signatures

(a) Root processor alone								
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) Two added worker processors								
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	14.1	–	20.9	–	24.4	–	28.3	–
2	7.4	1.9	10.7	2.0	12.6	1.9	15.4	1.8
3	5.2	2.7	7.4	2.8	8.9	2.7	11.3	2.5
6	3.5	4.0	4.6	4.5	5.6	4.4	7.9	3.6
9	3.3	4.3	4.1	5.1	5.0	4.9	7.3	3.9
12	3.2	4.3	4.0	5.2	4.9	5.0	7.2	3.9
Docs. searched	700 (12%)		1171 (20%)		1427 (24%)		1709 (28%)	

(c) Four added worker processors								
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	14.8	–	21.6	–	24.8	–	28.6	–
2	7.6	1.9	11.1	1.9	12.9	1.9	15.5	1.8
3	5.2	2.9	7.6	2.8	9.0	2.8	11.3	2.5
6	3.0	4.9	4.4	4.9	5.4	4.6	7.6	3.8
9	2.4	6.2	3.4	6.4	4.4	5.6	6.6	4.3
12	2.1	7.0	3.0	7.2	4.0	6.2	6.3	4.5
Docs. searched	739 (12%)		1214 (20%)		1463 (24%)		1738 (29%)	

---

The initial runs 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 which give the number of documents that matched the query at the signature level and that were passed on for the pattern-matching search. 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 pattern-matching 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 pattern matching, some processing can be eliminated. This is possible because the worker transputers are informed that it is unnecessary 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.

A 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 pattern matching takes place, then near-linear speed-up behaviour is observed for networks containing up to around nine transputers. The speed-up then begins to level-off rather more than we had expected from previous simulation experiments, which had suggested that near-linear speed-ups should be obtainable with much larger farms than those used here [19]. The speed-up behaviour becomes still more disappointing as longer and longer text signatures are used: 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).

## 4.2 Use of trigram-based signatures

The results in the previous [section](#) are for nearest-neighbour searches where the natural language queries were converted to a set of right-hand-truncated word stems. However, the signatures used for these searches had been optimized for this type of query and 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. Greater speed-ups than those obtained so far were expected to be obtained if a more appropriate type of signature for these sorts of pattern-matching operations was used, e.g., one based on the presence of trigrams. The use of such signatures was therefore tested to investigate whether these expectations were justified.

Text signatures based on the trigrams of the content-bearing words 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 in which three types of wildcard character were inserted at appropriate places in the query terms. These were: the ? symbol, which represented a single unspecified letter, the # symbol, which represented up to one unspecified letter, and the \* symbol, which represented any number of unspecified letters within a word. For example, the search term *itemi?e* would retrieve both *itemise* and *itemize*; *arch#eology* would retrieve both *archaeology* and *archeology*; *comput\** would retrieve *computing*, *computers*, *computational*, etc.; and *##sul?#ur\** would retrieve *sulphur*, *desulfurize*, *sulphurisation*, etc. In all, 39% of the total number of words in the original queries had wildcard characters added to them.

The triple-chain network used previously was altered slightly to test these new text signatures. An iterative backtracking algorithm, described by McGregor [34], was adopted to modify the pattern-matching part of the search to allow searches for terms containing wildcards. Firstly, the largest part of the query term which does not contain wildcard characters is searched for using the Boyer–Moore algorithm as before. Any document term which matches this term fragment is then isolated and the query and document terms are matched from the start using the backtracking algorithm. If there is some mismatch, the Boyer–Moore search continues from where it stopped. The term-weighting calculation had to be altered, both in the pattern-matching search and in the text-signature search, to take account of the fact that exact frequency information was not available for those terms containing wildcard characters. 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 that the term frequency was the average frequency of the terms in the document collection. This meant that there was a graceful degradation in retrieval effectiveness from the case when no terms contained wildcards and all weighting information was available to the case when all terms contained wildcards and there was no weighting information.

Table 2(a) contains the results of searches using the trigram text signatures with wildcard characters. It will be observed from this table that the response times were much greater than with the previous, word-based text signatures and that the speed-ups were much lower, when compared with the results for the 256-bit signatures (which were closest in size to the trigram signatures). Analysis of the results revealed that the use of the trigram signatures had created another bottleneck to add to that noted in our previous paper, which had been caused by a problem in distributing the documents from the mass storage transputer to the farm and which had been overcome by moving from the single-chain network to the triple-chain network [10].

The new bottleneck arose from the amount of time that is needed to carry out the text-signature search, since the trigram-based signatures involve the checking of many more bit positions for each query stem than do the word-based signatures. When a document has passed the text-signature search, its number is sent to a process, 'Store docs.', on the mass storage transputer; this process makes available to the farm the text of the document with this number. The time required for the processing of the trigram signatures meant that there is a much greater delay between sending out each of the document numbers for those documents that need to be searched by the processor farm, which thus cannot be used to full effect. This problem could only be counteracted by having several processors searching the text signatures simultaneously and feeding document numbers into the mass storage transputer. A method of distributing text-signature searching over

---

several processors has been described by Walden and Sere [22] but these authors do not include a pattern-matching stage, and an alternative approach was investigated as discussed below. Specifically, whilst retaining our processor-farm implementation of the pattern-matching search, we have adopted an alternative approach to parallel processing, called *geometric parallelism* [29,30], for the implementation of the initial, text-signature search. This involves a static distribution of data across the available processors, rather than the dynamic distribution that characterizes a processor farm.

### 4.3 Distribution of the text-signature search

The method chosen initially for distributing the signatures was to add two extra processors to the root processor. The signatures were allocated to the processors statically rather than dynamically. It would not be sensible to allocate the text signatures dynamically during processing because it would probably take as long to send a text signature between processors as it would take to match it against a query signature. The first half of the text signatures were placed on one processor and the second half were placed on the other. The root processor therefore had no signature searching to do; it was responsible only for controlling the search. The ‘Compare text sigs.’ process of Figure 2 was duplicated on the two extra processors and an extra routing process was added to the root processor to send messages between the mass storage transputer and the extra transputers.

Runs were then carried out with this network using the wildcard queries: the results of the searches are detailed in Table 2(b). A comparison of these figures with those in Table 2(a) shows that the response times with large networks were substantially reduced. The speed-ups also improved, but it can be seen that the speed-up began to level off after about six worker transputers were included in the farm. Since there was still room for improvement, it was decided to modify the network still further so that the signatures were distributed on not two but four processors (as shown in Figure 3). Extra routing processes were added to the original two extra processors to send messages between the outer processors and the root processor. The new outer processors contained just the ‘Compare text sigs.’ process.

The runs with this new network are detailed in Table 2(c). Comparison with parts (a) and (b) of this table shows that there are further reductions in the response times with large networks, but not as dramatic a reduction as in the previous results. A comparison of the speed-ups with the previous figures shows that the speed-ups have also improved, more noticeably when fewer documents are required by the user. There is still a levelling-off in the speed-up after about six worker transputers are included in the farm, but this is less abrupt than previously.

It can be observed by comparison of the three parts of Table 2 that as the number of transputers involved in signature searching is increased the number of documents sent for text scanning also increases slightly. An explanation for this behaviour requires a rather more detailed explanation of the way in which the nearest-neighbour search was implemented. We have noted in Section 3.1 that a similarity calculation is carried out once the text-signature search of a document has been completed; the number of bits common to a document signature and the query signature allows the calculation of an upper bound to the similarity that would be obtained when the corresponding sets of stems are compared. The process ‘Update results’ in Figure 2 maintains a list of the top-ranked  $n$  documents and the corresponding similarities, where  $n$  is the number of documents that is to be displayed

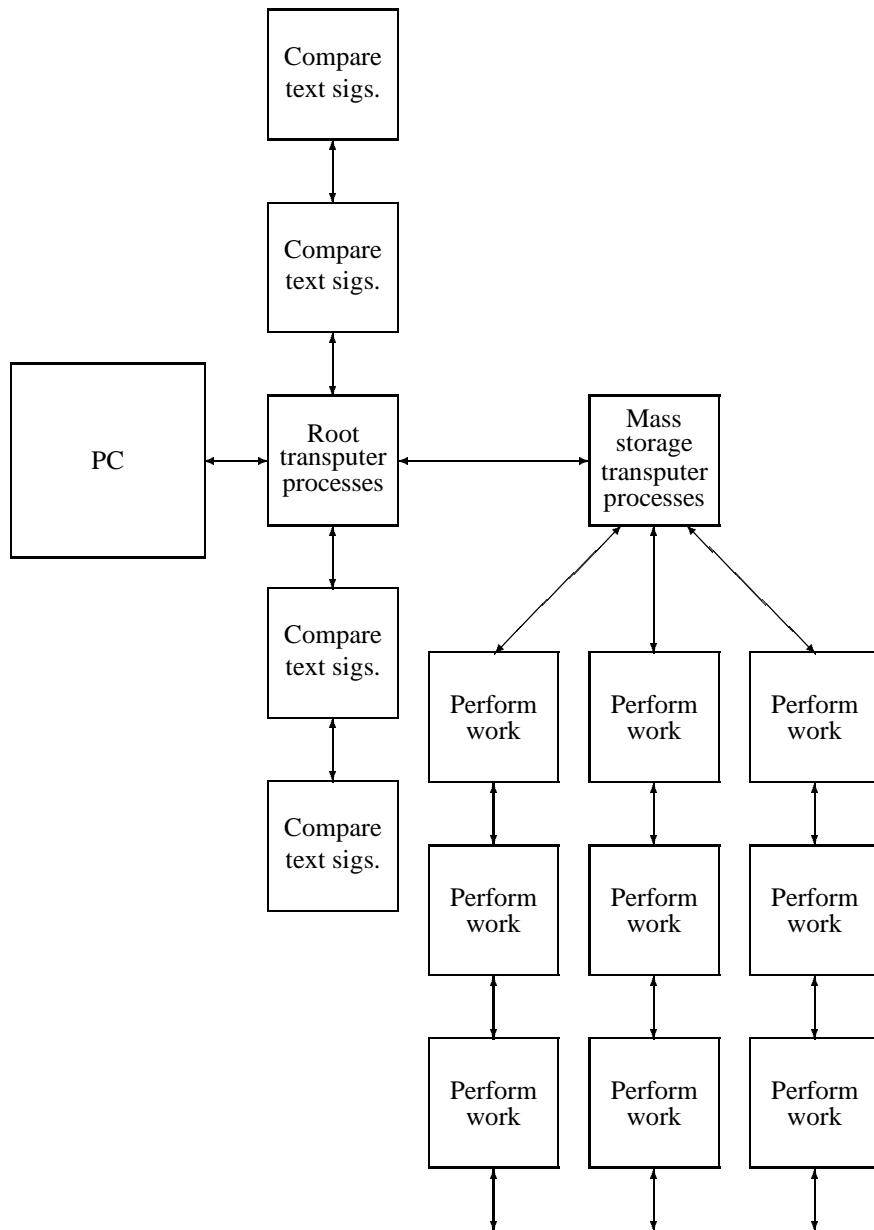


Figure 3. Triple-chain transputer network configuration for parallel text searching with signatures distributed on four processors. Rectangles represent processors. Connections between processors represent bidirectional transputer links, with the arrows indicating whether messages are passed in one or both directions

---

to the user at the end of the search. A document is passed on for the pattern-matching search if, and only if, the calculated upper bound is not less than the actual, calculated similarity for the bottom-ranked document in this list. As greater numbers of transputers are involved in the signature search, document numbers are sent out more quickly to the previously mentioned 'Store docs.' process. Therefore, more document numbers have been sent out by the time the first result arrives at the 'Update results' process. This effect also plays a part throughout the rest of the search because more document numbers are sent out before high similarity value results are returned to the 'Update results' process, meaning that more documents need to undergo the second-stage search than previously. Thus, an increase in the amount of pattern matching that is required is a direct, albeit unexpected, effect of implementing a more efficient text-signature search.

## **5 DEVELOPMENT OF THE NETWORK**

### **5.1 Processing of large datasets**

The experimental results in the previous section show that interactive nearest-neighbour searching is feasible using a transputer network, in that mean response times of just a few seconds were obtained. However, these results are limited by the fact that all of the documents were stored in memory, on the 8 Mbytes Meiko mass storage board. It was not possible to investigate the feasibility of a parallel solution with data being read from disk because only disks with very slow access rates (approximately 400 Kbytes/second) were available when these experiments were carried out. Faster equipment is now becoming available, albeit currently at a considerable price, and it is thus appropriate to consider how such equipment could be used to handle datasets larger than that used here. The fastest response time obtained during the experiments was 2.1 seconds to search the 3.7 Mbytes of LISA data. This is equivalent to a searching rate of 1.8 Mbytes/second, which is close to the transfer rate of current high-performance hard disk units. Searching of arbitrarily large files stored on disk could be achieved at this search rate if the mass storage card was attached to a disk capable of delivering data at a rate of approximately 2 Mbytes/second: this could be done via a model M212 disk-control transputer.

One point that should be noted is that the queries used here were fairly long, having 22 unique stems on average. It was observed that when shorter queries were input to the memory-based system used in our experiments, the response time was much faster than the average times listed in the tables of results. Therefore, in a disk-based system, the response times for searching for short queries would again be limited by the disk access times, unless a disk was available that had an access rate much greater than 2 Mbytes/second.

### **5.2 Implementation of Boolean searching**

Our work has considered only nearest-neighbour searching, where a large fraction of the documents undergoing the signature search also need to undergo the computationally demanding pattern-matching search. A very different situation would pertain if we wished to use transputer networks for Boolean searching, where the use of the logical operators in the signature search means that only a very small number of documents need to undergo the pattern-matching search. In this case, the overall computational requirements would be dominated by the signature search, and it would thus be necessary to focus more strongly

---

on distributed implementations of the signature search, with much less attention being given to distributed implementations of the pattern-matching search. We thus have the situation that rather different types of network would be required to optimize the efficiency of searching in Boolean and in nearest-neighbour environments: were there a need to provide both types of retrieval facility, it would be necessary to consider networks that would be hospitable to both types of search.

### 5.3 A combined implementation

The networks considered to date have all involved a dynamic distribution of the textual data, and a static distribution of the signature data. It would be possible to distribute both types of data statically, by loading one subset of a text database, together with the associated signatures, onto each transputer in a network. Each transputer would then continually scan just those documents that were local to it, rather than having documents distributed to it from a central store as in the current system. It would be of interest to compare this distributed memory system with the analogous implementations described by Walden and Sere [22], who studied signature-based retrieval without any second-level pattern matching, and by Sharma [20], who suggested the storage of clusters of documents at each node. The main advantage associated with the use of a database that has been partitioned in this way is that searching involves very little data transfer. Once the query has been input by the user and broadcast to the network, the main problem lies in communicating the threshold similarity value to the processors in the network. There is also a communication cost associated with the transfer of the texts of the matching documents from the network transputers to the control transputer. The main disadvantage with using this method is that it would be extremely expensive to implement on large datasets at present because of the cost of the memory. A disk-based version of this approach could be investigated using multiple disk units, each of which was associated with a single transputer or collection of transputers in the network and which contained the documents associated with those transputers. The network could then be filled with new documents once the members of the current subset of the database had been matched against the query.

## 6 CONCLUSIONS

The main aim of the project described in this paper was to investigate whether PC-based transputer networks could be used to provide interactive nearest-neighbour searching in serial files. The experimental results demonstrated that this is, indeed, possible with the small dataset used here, which contained 3.7 Mbytes of text. An average response time of 10.9 seconds per query was obtained when just the second-stage pattern-matching search was carried out (all figures in this paragraph correspond to the use of the largest network in each case). Word-based text signatures allowed response times of between 2.6 and 7.0 seconds to be obtained, depending on the number of documents shown to the user at the end of the search. However, the speed-up was very low when the larger text signatures were used, showing that it was the text signatures, rather than the parallel processing, which were mainly responsible for the fast response times that were achieved. The trigram-based text signatures resulted in longer response times of between 10.0 and 14.1 seconds for queries containing wildcards, depending again on how many documents were shown to the user at the end of the search. By distributing the text signature searching as well as

the pattern-matching search, the speed-ups were significantly improved and the times were reduced to between 2.1 and 6.3 seconds.

Our results 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, the speed-ups that have been achieved are not as great as we had expected when the work commenced, despite the considerable effort that has been expended in trying to maximize the efficiency of searching. In this respect, our results are analogous to those that have been obtained in previous studies of database applications on processor farms. Watson and Halsall [25] have discussed the use of a Meiko Computing Surface with 50 transputers for processing a large cartographic database containing over 150 Mbytes. They found that the required levels of speed-up were not achieved because of communication overheads in the network that they used. Similar results have been obtained in several studies of the graph-matching operations that are used to search databases of chemical structures, where it has been shown that it is often not possible to keep a processor farm fully occupied with computational work, with the result that only a limited amount of speed-up can be achieved [26,35]. Finally, Stringer and Waring [28] have investigated the use of ring and hypercube networks of transputers for searching a database containing the names and amino acid sequences of approximately 4000 proteins. They used networks containing up to 16 transputers and concluded that the work carried out by the root transputer became a limiting factor in the processing as the number of transputers was increased.

All of these problems, and those encountered in our own studies, are associated with the need to pass large amounts of data or intermediate results between the individual processors in the network that is being used. We thus conclude that while transputer networks have considerable potential for database searching, a substantial increase in the inter-processor communication speeds will be required if this potential is to be fully realized.

#### 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; Mr Andy Jackson, Mr Glenn Miller, Mr Geraint Jones and Dr Jon Kerridge of the National Transputer Support Centre for technical support; and the referees for comments on an earlier draft of this manuscript.

#### REFERENCES

1. G. D. V. Smit, 'A Comparison of Three String Matching Algorithms', *Software—Practice and Experience*, **12**, 57–66 (1982).
2. J.H. Ashford and P. Willett, *Text Retrieval and Document Databases*, Lund, Sweden: Chartwell-Bratt, 1988.
3. D. Lucarella, 'A Search Strategy for Large Document Bases', *Electronic Publishing*, **1**, 105–116 (1988).
4. G. Salton, *Automatic Text Processing: the Transformation, Analysis and Retrieval of Information by Computer*, Reading, MA: Addison-Wesley, 1989.
5. P. Willett, 'Software and Hardware Techniques for String Searching in Serial Document Databases', *World Patent Information*, **10**, 120–129 (1988).
6. W. B. Croft and P. Savino, 'Implementing Ranking Strategies Using Text Signatures', *ACM Transactions on Office Information Systems*, **6**, 42–62 (1988).

7. C. Faloutsos, 'Access Methods for Text', *ACM Computing Surveys*, **17**, 49–74 (1985).
8. C. Stanfill and R. Thau, 'Information Retrieval on the Connection Machine: 1 to 8192 Gigabytes', *Information Processing and Management*, **27**, 285–310 (1991).
9. S. F. Reddaway, 'High Speed Text Retrieval from Large Databases on a Massively Parallel Processor', *Information Processing and Management*, **27**, 311–316 (1991).
10. J. K. Cringean, R. England, G. A. Manson and P. Willett, 'Network Design for the Implementation of Text Searching Using a Multicomputer', *Information Processing and Management*, **27**, 265–283 (1991).
11. M. J. Flynn, 'Some Computer Organisations and their Effectiveness', *IEEE Transactions on Computers*, **C-21**, 948–960 (1972).
12. R. W. Hockney and C. R. Jesshope, *Parallel Computers 2. Architecture, Programming and Algorithms*, Bristol: Adam Hilger, 1988.
13. D. B. Skillicorn, 'A Taxonomy for Computer Architectures', *Computer*, **21**(11), 46–57 (1988).
14. M. Dubois, C. Scheurich and F. A. Briggs, 'Synchronisation, Coherence and Event Ordering in Multiprocessors', *Computer*, **21**(2), 9–21 (1988).
15. D. D. Gajski and J. K. Peir, 'Essential Issues in Multiprocessor Systems', *Computer*, **18**(6), 9–27 (1985).
16. P. C. Patton, 'Multiprocessors: Architecture and Applications', *Computer*, **18**(6), 29–40 (1985).
17. W. C. Athas and C. L. Seitz, 'Multicomputers: Message-Cassing Concurrent Computers', *Computer*, **21**(8), 9–24 (1988).
18. D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, Cambridge, MA: MIT Press, 1987.
19. J. K. Cringean, G. A. Manson, P. Willett and G. A. Wilson, 'Efficiency of Text Scanning in Bibliographic Databases Using Microprocessor-Based Multiprocessor Networks', *Journal of Information Science*, **14**, 335–345 (1988).
20. R. Sharma, 'A Generic Machine for Parallel Information Retrieval', *Information Processing and Management*, **25**, 223–235 (1989).
21. M. Stewart and P. Willett, 'Nearest Neighbour Searching in Binary Search Trees: Simulation of a Multiprocessor System', *Journal of Documentation*, **43**, 93–111 (1987).
22. M. Walden and K. Sere, 'Free Text Retrieval on Transputer Networks', *Microprocessors and Microsystems*, **13**, 179–187 (1989).
23. D. Crookes, P. J. Morrow and G. Philip, 'The Development of a Transputer-Based Image Database', in D. J. Pritchard and C. J. Scott, C.J. (Eds.), *Applications of Transputers 2* (pp. 189–195), Amsterdam: IOS Press, 1990.
24. C. J. Elliott, 'Very-High Performance Multiple-Instruction Multiple-Data Applications', *Philosophical Transactions of the Royal Society of London*, **A326**, 471–479 (1988).
25. M. Watson and F. Halsall, 'Concurrent Operations on Very Large Cartographic Databases', in L. Freeman and C. Phillips (Eds.), *Applications of Transputers 1* (pp. 312–317), Amsterdam: IOS Press, 1989.
26. A. T. Brint and P. Willett, 'Identification of 3-D Maximal Common Substructures Using Transputer Networks', *Journal of Molecular Graphics*, **5**, 200–207 (1987).
27. G. M. Downs, M. F. Lynch, P. Willett, G. A. Manson and G.A. Wilson, 'Transputer Implementations of Chemical Substructure Searching Algorithms', *Tetrahedron Computer Methodology*, **1**, 207–217 (1988).
28. K. Stringer and L. C. Waring, 'Transputer-Based Database Organisation—an Example Protein Database Implemented Using Pipeline and Hypercube Configurations', in A. Bakkers (Ed.), *Applying Transputer Based Parallel Machines* (pp. 296–300), Amsterdam: IOS Press, 1989.
29. A. J. G. Hey, 'Reconfigurable Transputer Networks: Practical Concurrent Computation', *Proceedings of the Royal Society*, **A326**, 395–410 (1988).
30. D. J. Pritchard, C. R. Askew, D. B. Carpenter, I. Glendinning, A. J. G. Hey and D. A. Nicole, 'Practical Parallelism Using Transputer Networks', *Lecture Notes in Computer Science*, **258**, 278–294 (1987).
31. R. N. Horspool, 'Practical Fast Matching in Strings', *Software—Practice and Experience*, **10**, 501–506 (1980).
32. G. Amdahl, 'The Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities', *AFIPS Conference Proceedings*, **30**, 483–485 (1967).

- 
33. J. L. Gustafson, 'Reevaluating Amdahl's Law', *Communications of the ACM*, **31**, 532–533 (1988).
  34. J. J. McGregor, 'Backtrack Search Algorithms and the Maximal Common Subgraph Problem', *Software—Practice and Experience*, **12**, 23–34 (1982).
  35. P. Jochum and T. Worbs, 'A Multiprocessor Architecture for Substructure Search', in W. A. Warr (Ed.), *Chemical Structures. The International Language of Chemistry* (pp. 153–200), Heidelberg: Springer, 1988.