PARALLEL COMPUTING IN INFORMATION RETRIEVAL - AN UPDATED REVIEW

A. MACFARLANE*, S.E. ROBERTSON, J.A. McCANN

School of Informatics, City University,
Northampton Square, London EC1V 0HB

The progress of parallel computing in Information Retrieval (IR) is reviewed. In particular we stress the importance of the motivation in using parallel computing for Text Retrieval. We analyse parallel IR systems using a classification due to Rasmussen [1] and describe some parallel IR systems. We give a description of the retrieval models used in parallel Information Processing.. We describe areas of research which we believe are needed.

1. INTRODUCTION

THE PURPOSE OF THIS REVIEW is to chart the progress of the use of parallel computing in Information Retrieval (IR) since the last major review of the subject by Rasmussen [1]. We also review important work in the past. We describe parallel architectures for readers unfamiliar with the area of parallel computation. We analyse the different approaches to parallel IR using a classification due to Rasmussen [1]. Examples of parallel IR systems are given in a case studies section. We stress the importance of the motivation for the use of parallel computing in IR, in particular when and when not to use parallel systems. We consider the decisions needed when choosing an approach for parallel IR. The retrieval models used in parallel IR systems are described. A summary and conclusion are given, together with some suggestion for further work needed in the area.

*To whom correspondence should be addressed. Email: andym@soi.city.ac.uk

2. PARALLEL ARCHITECTURES USED IN IR SYSTEMS

2.1 Parallel architecture classification

Flynn [2] describes a taxonomy for classifying parallel architectures. A number of criticisms have been levelled at the taxonomy:

(a) There is no treatment of input/output;
(b) The instruction set used is ignored.

In the context of IR, ignoring input/output is a particular problem (see section 2.3). In spite of these limitations the taxonomy has become the most popular method for describing parallel architectures and continues to be widely used in the field of parallel computing research including parallel IR. An alternative taxonomy is given by Hockney and Jesshope [3]. The Flynn taxonomy uses the concept of streams [2] which are a sequence of items operated on by a CPU. These streams can either be *instructions* to the CPU or *data* to be manipulated by the instructions. We therefore have four broad classes of architecture:

(a) SISD - Single Instruction Single Data Stream;
(b) MISD - Multiple Instruction Single Data Stream;
(c) SIMD - Single Instruction Multiple Data Stream;
(d) MIMD - Multiple Instruction Multiple Data Stream.

The first of these, SISD is the normal sequential von Neumann architecture machine which has dominated computing since its inception. The MISD class is controversial: some argue that it is a null class and does not usefully describe any architecture [4,5] while others assert that systolic arrays can be placed in this class [6]. We address the MISD class in our discussion on special parallel hardware below. We will ignore the SISD class for the rest of the paper.

The SIMD class describes an architecture in which the same instructions operate on different data in parallel. It is therefore widely known as data parallel computing. Instructions are broadcast to n processors in the architecture which operate on the data held in that processor. Examples of this type of architecture are the ICL/AMT DAP

[3,7] and Thinking Machines CM-2 [6]: the DAP is described in more detail below. The architecture has been dominant in the use of Parallel Computing in Information Retrieval.

The MIMD class describes an architecture in which processors independently execute different instructions on different data. The programs which run on this class of machine are therefore a great deal more complex than one could envisage on any of the other architectures. There is a wide variety of this class of architecture including those in which processors share the same memory and others in which processors have their own memory. These are known are Shared Memory and Distributed Memory architectures (see examples in figure 1). Each has its own subdivision which

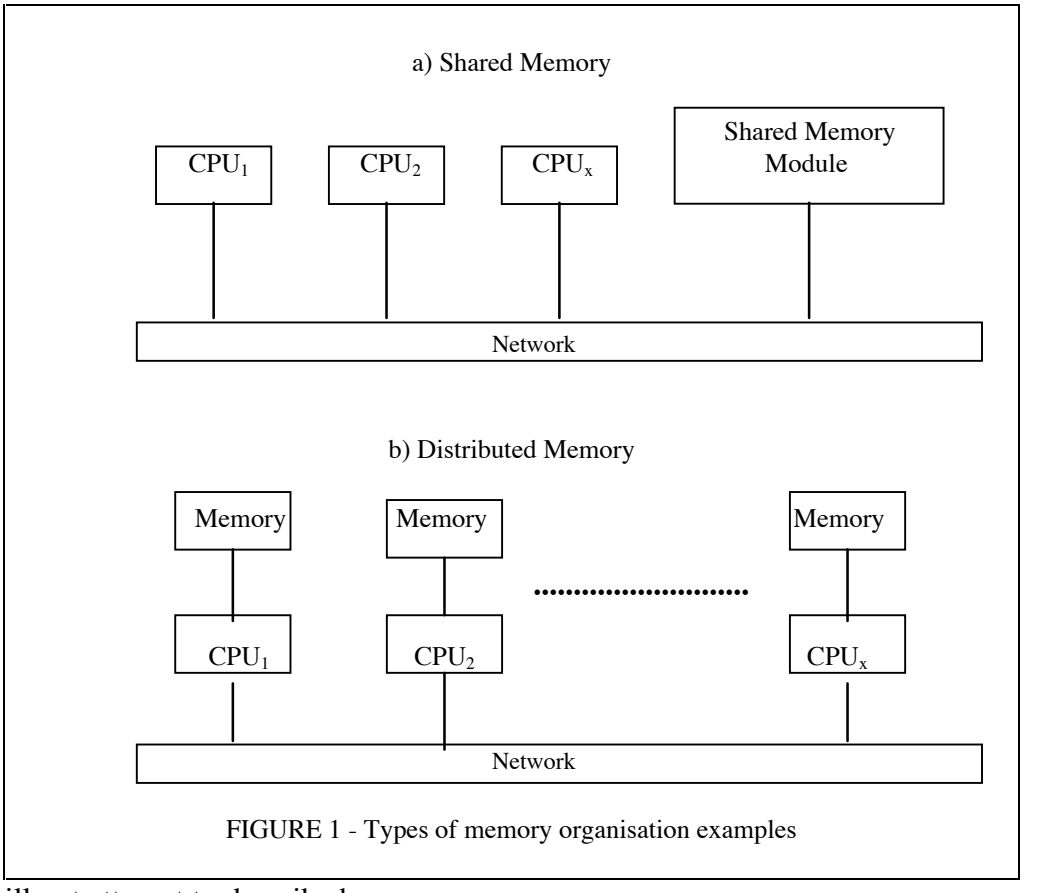


FIGURE 1 - Types of memory organisation examples

we will not attempt to describe here.

With the former, interprocessor communication is done through concurrency control mechanisms such as flags in memory, while the latter uses message passing. There is also a hybrid architecture known as Distributed Shared Memory (DSM) where programs see a single memory, but access is serviced by message passing. An

example of a machine with the MIMD class architecture is the Fujitsu AP1000 which is described in section 2.2 below.

It should be noted that a further class of architecture exists which does not fit well into Flynn's classification. Special-Purpose Hardware has been built to accommodate IR systems [8] including associative memories, finite state machines and cellular arrays [9]. Some of this work has been in building special purpose parallel architectures [10] for text retrieval and we include it in the review for completeness

2.2 Parallel architectures used in IR

We now turn to specific machine architectures which have been used for parallel IR systems. We give an example of each type of architecture from section 2.1; the DAP, Fujitsi AP1000, and special parallel hardware. We also discuss the growing impact of networked workstation technology. More information on various architectures can be found in Rasmussen [1].

A. *DAP (Distributed Array of Processors)*. The AMT (formally ICL) DAP is a SIMD class architecture. The DAP [7] organisation is an array of 1-bit processing elements (PEs) arranged in a 32 by 32 matrix for the 500 series and 64 by 64 for the 600 series; 1024 and 4096 PE's in total respectively. The 600 series has four times the memory and processing power of the 500 series. Each processor is connected to its north, south, east and west neighbour processors (known as a NEWS grid) and to the row and column of the matrix by a bus system. Each processor has at least 32 Kbits of its own local memory. The ICL DAP needed a mainframe as a front end, but workstations can be used for current varieties. The architecture has a Master Control Unit (MCU) which broadcasts instructions and data to the array to work on and also obtains the results from the array. The DAP has very fast I/O capabilities of up to 50 Mbytes per second to overcome the I/O bottleneck (the I/O problem in parallel computing for IR is discussed in section 2.3 below). The DAP is successfully used by the DapText system described by Reddaway [11] and is included in the case studies section (7.1) below. Reuters use this system for their Text Retrieval purposes. DapText has been implemented on both the 500 and 600 series of the DAP. Other work includes a British Library project for using the DAP in IR, described in [12-15].

B. *Fujitsu AP1000*. The Fujitsu AP1000 is a MIMD distributed memory architecture with up to 1,024 SPARC processors or cells which are interconnected using a two dimensional torus. Each cell can support up to 16 Mbytes of memory with a promise in the near future of 64 Mbytes per cell. Data can be moved in and out very quickly using a 50Mbyte per second broadcast network. To overcome the I/O bottleneck, the HiDIOS file system is useful with a load rate in excess of 50 Mbytes per second. The AP1000 has global reduction operations which are useful for term weighting calculations. Work on IR using the AP1000 is currently being pursued at the Australian National University through the PADRE system [16-22], which has evolved through the PADDY [23,24] and FTR [25] systems. These systems are discussed in more detail in the case studies section (7.3) below.

C. *Special parallel hardware*. A number of different special purpose parallel hardware architectures have been built for pattern matching in IR. The reader is referred to Hollaar [9] and Hurson et al [10] for more detailed information. One of the architectures, systolic arrays, can be classed under MISD. Systolic arrays (an example of cellular arrays) work by pattern matching characters every clock cycle in a pipeline where the target text and query travel in opposite directions. The associate memory architecture uses memory chips as the comparison devices, therefore patterns can be matched in parallel in the actual memory. Finite State Automata (FSA) use transition tables over single cell comparator chips. The argument made for the use of these systems is that normal computing components are not very efficient at character comparison and therefore are not particularly good for pattern matching in text retrieval. Hence special purpose components are preferable to conventional computers since they offer a faster throughput for queries. The authors do not agree with this argument. Inverted files have been shown to provide very efficient query service (at the cost of extra storage) for reasons which will become clear below. It is also very doubtful that these specially made chips could ever compete in price with general purpose chips: the cost of production and manufacturing CPUs is very expensive. We therefore do not see a future for these special purpose systems in IR. This is consistent with DeWitt & Gray's opinion that the future development of parallel systems will depend on standard components [26].

D. *Networked Workstation Technology*. The current trend in parallel computing is to use a group of networked workstations or PCs, rather than special purpose machines. A great deal of interest has been generated in programming environments such as PVM [27] and standards such as MPI [28]. One particular system discussed in this review (MARS) uses networked workstation technology for its hardware platform [29]. The growth of distributed parallel processing has dealt a severe blow to many specialist parallel computer manufacturers such as Kendall Square Research, MassPar, Thinking Machines and AMT. Kendall Square Research has gone bankrupt, while the others are either much smaller concerns (MassPar and Thinking Machines) or have metamorphosed into other companies (AMT to CPP). DeWitt and Gray's opinion quoted above, on the development of parallel systems using standard components, is reinforced by the networked workstation technology factor. The trend towards workstation networks in parallelism will have significant impact on future parallel IR systems.

2.3 *I/O implications of different architectures*

One of the main qualities of IR is that in the main it is I/O bound rather than compute bound. This means that more time is spent on reading in data from disk than actually doing computation. Thus a problem occurs where efficiency of the system is reduced because the data cannot be read in fast enough to service the computation. This problem is know as the I/O bottleneck and it is one that is shared with the area of Parallel database systems [26]. In consequence, many of the systems mentioned above have very impressive I/O rates to overcome this I/O bottleneck. One architecture that is worth consideration in this respect is the Shared Nothing architecture described by DeWitt and Gray [26]. This architecture is classed under MIMD, and has a structure where CPUs have their own local disks to read data from. This reduces network traffic and disk contention considerably because data sharing is reduced to questions and answers rather than whole data sets (which can be very large in database systems). Index maintenance costs can also be reduced. Tomasic and Garica-Molina [30,31] make a very strong case for the use of Shared Nothing. Further work in the area is merited.

However such is not the whole picture. There are some IR computations which are compute bound and require considerable CPU resources. An example of this is a very large search spaces for passage retrieval and for query modification after relevance feedback, found within the Robertson/Sparck Jones probabilistic model [32]. In such cases fast I/O cannot make much difference to the overall efficiency.

3. MOTIVATION FOR PARALLEL IR SYSTEMS

On the assumption that we want to do more and faster, there are two main reasons for using parallel computing in general. The first is that the speed of a processor is ultimately limited by the speed of light [33], when the maximum possible miniaturisation for components on a silicon chip has been achieved. The second is that the cost of placing silicon in smaller and smaller areas is very high in both the design and manufacturing of processors. The second limitation occurs long before the first and is therefore the major consideration.

A number of performance measures are used in parallel computing. We define them informally here. Speedup is the gain in speed over sequential machines and is calculated by dividing the time spent on computation on the sequential machine by the time on the parallel machine. A speedup which equals the number of processors is said to be linear, greater than the number of processors is said to be super-linear. Efficiency gives a measure of how well a particular algorithm scales when processors are added. It is found by dividing the speedup found by the number of processors used. An efficiency of 1.0 is desirable, but rarely if ever achieved. The aim is to achieve a near 1.0 efficiency result. Whilst these measures are the accepted way of examining the performance of parallel systems, it should be noted that their usefulness have been brought into question [34]. For more formal definition of these measures see Rasmussen [1].

The performance of IR systems are measured by the retrieval effectiveness and retrieval efficiency they provide [35]. Retrieval efficiency is the measure of the time taken by an IR system to do a computation on the database, although this usually means search it. The relative merits of the gain in retrieval efficiency by using parallel IR systems against their sequential counterparts can be measured by the speedup/efficiency measures defined above. Users not only want fast and interactive

access to documents, they also want to be presented with documents which are relevant to their needs; this is measured by the retrieval effectiveness of the IR system. The most commonly used measures for retrieval efficiency are recall and precision. Recall is the measure of how many relevant documents are retrieved from the database. Precision is the quality of the documents presented to the user i.e. how many are of the documents retrieved are relevant. Parallel IR systems have a place in providing retrieval efficiency for users and may well help in providing extra retrieval effectiveness.

The use of parallel computing specifically for IR has been quite controversial. Both Stone [36] and Salton/Buckley [37] have argued that an Inverted file algorithm running on a sequential machine can outperform a signature file algorithm running on a parallel machine. The discussion in both papers originate from the work done by Stanfill and Kahle on the Seed system [38]. Since the Seed system uses surrogate coding (a response time of 2 minutes is stated for an example query), a sequential system using Inverted files would in theory be able to offer a much faster response time to queries. This is because fewer comparisons and much less I/O is needed. Stone [36] compares the performance of an Inverted file on a single CM-2 node, while Salton and Buckley [37] use the example of a Sun 3 to produce their theoretical results. Of the two studies Stone's goes much further. Stone put forward an alternative parallel algorithm to be used on Inverted files in order to run the sequential Inverted file system in a more efficient manner. Salton and Buckley [37] are rather more negative and suggest that "the global vector matching systems developed over the past 25 years for serial computing devices appear more attractive in most existing text processing situations". It is hard to accept or reject this statement without knowing what they mean by most existing text processing situations, and without any analysis as to whether the global vector matching systems could also gain from parallelism. In response Stanfill, Thau and Waltz [39] report an 80 fold performance advantage for a newer CM-2 against a Sun 4, which rather lessens the impact of the Salton and Buckley [37] paper. Ultimately Stone has been proved to be correct, since two parallel systems which use Inverted files on the CM-2 and the DAP have been commercially successful. The set merge on inverted lists can be computationally very intensive.

Four main reasons for applying parallel computers to Information Retrieval have been suggested [40]: these are too improve response times, search larger databases, use superior algorithms and reduce search cost. We discuss each reason in turn below.

3.1 *Response times*

In situations where a large number of users need access to the system, a sequential IR system may not be able to offer the required performance of the application. In general when large numbers of users are logged on, the response time to the user is likely to be greatly increased. A related point is that of throughput: throughput is the number of queries or insertions which can pass through the system in a given time period. Parallel computation has the potential to offer faster response times for individual queries and a greater throughput for queries and insertions. Response time is also dependant on database size in conjunction with multiple query service.

3.2 *Very large databases*

The response to user queries in very large databases (e.g. multiple Gigabyte) are likely to degrade particularly for those which have a reasonable rate of growth. In principle parallel systems tend to offer much better scaleup than sequential systems. Scaleup is defined by DeWitt and Gray [26] as "the ability of an N-times larger system to perform an N-times larger job in the same elapsed time as the original system". A query response time on a small IR system using a small database should be the same for a large IR system using a large database. It is important to introduce a note of caution as this point. The authors do not believe that parallel computing can be usefully applied at this juncture to small databases with a very few or single user base. The emphasis in this review is very much on large scale text databases.

3.3 *Superior algorithms*

We stated in section 3.2 that we do not believe that parallel computing can be usefully applied to small text databases at this point. It may be the case at some time in the future that a given algorithm which requires more computation to complete its task will be able to a offer superior retrieval effectiveness performance in terms of precision and recall than previously implemented algorithms. For example there are a number of extended boolean models [41] which offer very good precision/recall at the

cost of extra computation (which is high in the case of the P-NORM model because of the exponentiation operations required). Some in fact argue that extra computation will deliver much better results. Skillicorn [42] argues that regular expressions offer more powerful query capabilities than other searches. MacLeod and Robertson [43] suggest that generally speaking the "most effective algorithms are among the least efficient".

There has been some debate on the merits of extra effort to achieve a better level of retrieval effectiveness. Blair and Maron [44] evaluated a large operational full-text IR system over a six month period, and hypothesise a general deterioration of recall on databases of increasing size. They argue that extra effort is needed to overcome this deterioration and keep recall at reasonable levels. (By implication, the extra effort is assumed by these authors to be human effort at the indexing stage.) Salton [45] takes issue with their arguments, and they reply [46]. Although recall deterioration with file size must be regarded as unproven (and is particularly hard to prove empirically), the possibility both that it occurs and that it may be alleviated by more complex and more effective search algorithms is worth investigation.

A further issue arises from the very large search spaces which exist within the Robertson/Sparck Jones Probabilistic model [32]. Because these search spaces are so large it is unlikely that even parallel machinery will be able to explore all of it. A time complexity of $O(n^3)$ is reported [32], where n is a text atom, for unoptimized code on passage retrieval. The search space for query modification after relevance feedback is so large no order value can be stated. However it may be possible to search part of these spaces and thereby increase retrieval effectiveness (bearing the mind the caveat on recall in the last paragraph). At present such can only be proven by empirical experiment.

3.4 *Search Cost*

Stanfill et al [39] assert the cost effectiveness of an IR system is the ratio of database size to search cost i.e. the resources used to search the database. Using the assumptions that database search is linear with the size of the database, speedup is linear for those algorithms which keep processors busy and resource costs (such as communication overheads) are static, Stanfill et al [39] show that cost effectiveness asymptotically approaches a level of optimal cost effectiveness. By increasing the size of the database for example we can move the level of optimal cost effectiveness to a more favourable figure. It is stated that for a database of 1 Gigabyte the improvement in cost effectiveness is 100 fold, but for a 100 Gigabyte database the improvement is 10,000. However as Hawking [23] points out a higher figure needs to be treated with caution because hardware (the CM-2) can only use a limited number of Data Vaults, which restricts the amount of text, let alone index information that can be stored. Hardware factors therefore limits the relevance of this cost effectiveness metric.

4. APPROACHES TO PARALLEL IR

In this section we describe approaches to parallel information retrieval using a classification due to Rasmussen [1], influenced by Faloutsos's classification of access methods for text [47]. The classification does not differentiate between a particular algorithm and storage method. It is found that they tend to be bound together quite tightly in parallel IR systems. By algorithm we mean method of searching on the storage method and by storage method we mean organisation of the data on disk. The interaction between machine type and the classification is discussed in each of the sections below. The methods discussed are pattern matching, Signatures/Surrogate coding, Two-Phase search, Inverted Files, Clustering, Connectionist approaches and other miscellaneous approaches.

Some issues need to be addressed with respect to each of the algorithms in the classification. Firstly the assignment of tasks to processors will determine the level of performance gain over sequential systems: it cannot be taken for granted that using parallelism will automatically provide enhanced performance. The placement of tasks will also determine the level of interprocessor communication: an unavoidable overhead and one which may greatly degrade algorithm performance if data or task

placement is mis-handled. Data partitioning methods have a significant effect on task assignment and the subsequent level of interprocessor communication for a particular algorithm. Secondly there is the granularity of parallelism for algorithm. Granularity can either be fine, coarse or mixed grain, meaning small, large or variable computation sizes: a computation being a single unit of work which can be done by a processor. The type of query parallelism available in an algorithm is also very important: that is, the method of parallelism used to service user queries. Intra-query parallelism is parallelism within queries, that is a single query is distributed amongst processors. Inter-query parallelism is parallelism among queries, that is a number of queries are serviced concurrently. The concepts of data partitioning, granularity and query parallelism will be discussed with respect to each of the classes.

4.1 *Pattern matching*

Pattern matching is the method of searching the raw text in a given text corpus with a string query. There are a number of methods for matching patterns efficiently including the Knuth-Morris-Pratt and Boyer-Moore string searches [48] and variations of these. In a system without parallelism, pattern matching normally involves the sequential scanning of every document in the system: no index is used. Methods include left hand truncation, variable length don't care (VLDC), a proposed implementation of the "computing as compression theory" SP [49], proximity searches and pre-computed patterns [42,50]. We describe below parallel methods which have been implemented or are proposed for pattern matching algorithms.

Using an example we can describe the operation of pattern match in parallel computing. Firstly we partition the target text among our processors. We then broadcast the whole pattern to all processors and the pattern is applied in parallel to each partition of the text. Results from the processors are sent back to the user for inspection. This is a rather simplistic scenario, but it does give a flavour of the operation. A number of issues are thrown up by this example, in particular the operation of the algorithm on SIMD and MIMD architectures. The issue how of load balancing is affected by the implementation of the algorithm are important. With MIMD systems we can allow pattern matching on different processors to proceed independently of each other. The implementation on SIMD systems is slightly more problematic. Each pattern match needs to work in lock step on every processor:

patterns may need to advance a computed distance. Unless we keep this computed distance in a local variable, a set of processors have to wait until others have 'caught up' in the computed distance and our load balance is reduced together with further loss in the efficiency within the chosen pattern matching algorithm. However with the computed distance we are likely to finish pattern matching on some processors, leading to a gradual reduction in processor efficiency as processors complete their tasks: this is a problem shared with MIMD systems. An alternative method for SIMD systems described by Pogue and Willett [12] is to broadcast individual characters to processors one by one which match them in that order. As each match is a made the presence of a hit is recorded, if and only if the previous character in the sequence was matched.

More complex patterns can be applied to text corpus in the same manner as the MIMD and Pogue & Willett algorithms. With MIMD we simply apply a utility such as grep or fgrep to each text partition on every processor in parallel. An example is the PADDY system [24] which provides tools for the use of a regular expression library on each cell (processor) of a Fujitsu AP1000. Some examples of algorithms implemented on SIMD systems which support complex patterns are left-hand truncation and variable length don't care (VLDC). With left hand truncation we identify patterns which have different prefixes but the same suffix, before applying the query pattern. For VLDC, prefix and suffix patterns are recorded: the presence of a word delimiter between the result set of prefixes and suffixes is then identified.

The SP pattern match [49] would use a completely different method. The SP algorithm works by broadcasting each character in the query, from left to right, to each character in the text corpus to make a true or false match. Given that it is impossible to have a processor for every character in the database, we can assume that each processor is given a set of characters. A tree structure is built up which records the probabilities of matches being useful, in decreasing order (matches nearer the root will have a higher probability of usefulness). The parallelism in the SP algorithm lies in the broadcast of characters and the ability to create and manipulate the tree structure for each text partition. A time complexity of O(Q) is claimed where Q is the size of the query pattern. It should be noted that the SP theory is controversial, and there has been heated argument as to the usefulness of it in practice [51-53]. We are unable to

comment on its usability in practical situations until an empirical study has been done using a parallel implementation of the SP search algorithm.

Hawking describes a method of parallel proximity searches on the PADRE system [19]. A match set for each string in a query is created: this match set contains pointers to the first character of each instance of the pattern. Using some proximity value we merge these match sets by comparing the pointers and recording those pointers which meet the proximity value criteria. The set creation and merges can be done in parallel for each portion of text being searched in their respective cells. If documents are too large to fit in a single cell's memory, the cells need to communicate in order to complete the matching process: this inter-processor communication would reduce efficiency.

Skillicorn [42,50] describes a method of search which he asserts can be defined in terms of language recognition. The proposed algorithm uses a set of pre-computed patterns. Membership of textual data to these patterns is pre-computed in order to identify search patterns that have some common attributes. If membership of text to an pre-computed pattern is found it is placed in segments. The text would be partitioned across a given set of processors, the pre-computed pattern applied to the text and the search would access only those segments which are capable of matching a query. It is stated that where text is indexed as trees, regular expressions can be executed in logarithmic time complexity on a parallel computer.

An important theme in the algorithms described above is the distribution of text in one of two ways: either by text boundary (say documents) or by character (documents may reside over several processors). If text boundaries are crossed, more inter-processor communication is needed as processors need to exchange information. We can remove this problem by keeping documents as a whole in the processors. But this strategy itself has two main problems: the document may be too big to fit in a processor's main memory, and given that documents are likely to be of widely varying sizes a problem called data skew is observed. Data skew causes some processors to complete their computations faster than others, remaining idle until the whole computation has finished. This can cause a loss of efficiency, in the worse case degrading the computation to that of sequential time complexity. Hawking has defined a useful measure of Load Imbalance LI [17] in order to understand the effects of data skew on the pattern matching computation. An LI value of 2.0 is said to halve the

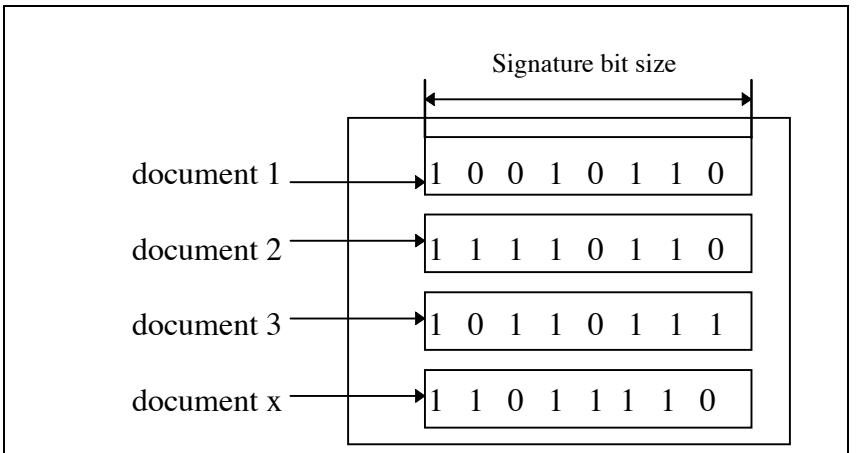
capacity of the memory and hence halve the effective speed of the parallel machine. A method to overcome the problem is to try to arrange the documents in such a way as to reduce this LI value. A simple example of this is to place as many small documents on the same processor as possible. Where practical we place large documents in neighbouring processors to reduce inter-processor costs while using smaller documents to fill up any extra memory. Breaking up the document into pages or paragraphs could also be useful.

The interaction between machine type and the classification tends to be based on the granularity of the computation. In the case of SIMD systems the granularity is that of a character, which is the finest grain that can possibly be used. With MIMD systems the granularity tends to be much coarser, but in fact is mixed granularity since documents are of varying sizes. The method supports intra-query parallelism and may be able to support inter-query parallelism with suitable processing, for example merging user queries and submitting them as a batch: we do not know of any systems which have implemented this.

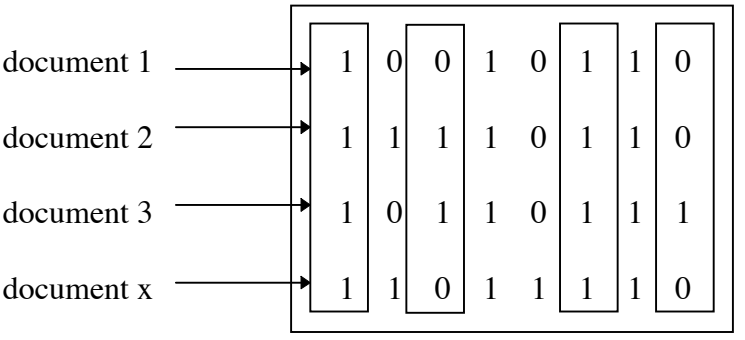
The pattern matching algorithms are very search intensive, but they have a low storage cost and allow different types of searches such as left hand truncation which are difficult to implement in the algorithms classified below.

4.2 *Signature / surrogate coding*

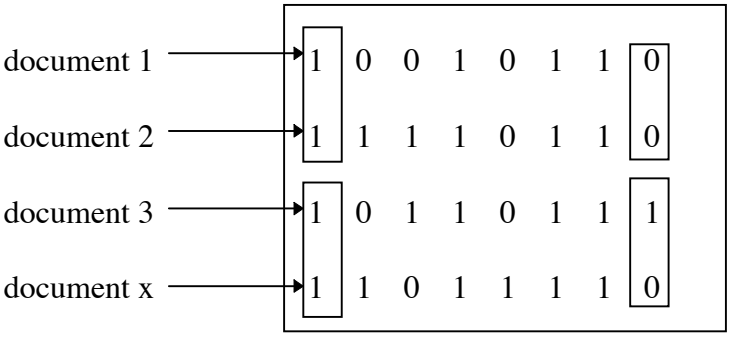
Text signatures are document surrogates which are generated by hashing terms on one or more bits of a fixed sized bit pattern [47]. Once these signatures have been generated they can be distributed to processors and searched in parallel. The search is done by applying the same hashing function to the query, as was applied to the documents. The search is therefore a fast bit comparison between the query and document surrogates. Pogue and Willett describe an alternative method where integer values of the bit positions are broadcast one by one to the processors [12]. The pioneer work described by Stanfill and Kahle on the Connection Machine has already been briefly mentioned [38,54]. Other work includes a Bit-Sliced Signature File (BSSF) method described by Panagopoulos and Faloutsos [55] and Frame-Sliced Partitioned Parallel Signature Files described by Grandi et al [56]. Detailed descriptions of these different methods are given below.



2a - Horizontal partitioning



2b - Vertical partitioning



2c - Frame partitioning

FIGURE 2 - Forms of Parallelism in Signature Files

Before we describe systems which use signatures it would be useful to review Signature files (see figure 2). Signature file can be viewed as matrices where the rows represent document signatures and the columns represent the bit size of the signature. We therefore have a number of partitioning methods for parallel computing on this matrix. The first, horizontal partitioning, represents row parallelism where signatures

are compared in parallel (figure 2a). The second, vertical partitioning, represents column parallelism where sections of the signatures are compared rather than the whole (figure 2b). Vertical partitioning can be done by the bit or by a frame: a subset of the column matrix (figure 2c). A hybrid policy of vertical and horizontal partitioning can also be used. How these partitioning methods work in practice will become clearer in the discussion below.

The Seed system described by Stanfill and Kahle [38] uses the horizontal method (figure 2a) for partitioning the signatures. Seed uses a SIMD architecture, in this case the Connection Machine CM-2. The program works by loading signatures into memory, broadcasting the query signature to the processors to compare in parallel and retrieving the results. In theory it is possible to load a document signature in every processor, but Stanfill and Kahle assert that for a 512 bit signature "a limit of 15 to 30 words is reasonable". Therefore the system creates a number of signatures and spreads them across a number of processors if this upper limit on term to signature size is exceeded. Thus document sizes in the corpus have a direct effect on how many signature comparisons can be executed in a given search. The system allows the use of Relevance Feedback to reformulate a query. Reported results include a running time of 50ms for a 200 term query on a 112 Megabyte database. Estimates for a 15 Gigabtye database are also given with a running time of 2 minutes for a 25 term query and 3 minutes for a 20,000 term query. The latter estimates cast doubt on the usefulness of the system in interactive environments when very large databases are searched. This method of search has also been used in Tranputer machines [57]

Panagopoulos and Faloutsos [55] point out that the signature file for a very large database using horizontal partitioning may not fit in main memory, which has implications for their use in interactive applications: the Seed estimates given above bear out this argument. They therefore propose a Bit-Sliced Signature File (BSSF) which is based on vertical partitioning (figure 2b) on the bit level. The method would work by storing the signature file matrix by columns rather than rows. Each term in a query is hashed to a signature. The hashed positions of the query are identified and only those relevant column slices (or bit-slices) are fetched in to main memory and compared. A processor has a given number of bits with which to store a subset of the bit-slice. The algorithm would loop through these bits and compare subsets of the bit-slice in parallel. Where the bit-slices fit in main memory a total fetch policy can be

used: where they do not a partial fetch policy would be used i.e. a subset of the bit-sliced identified from the query hashing. The proposed method would work on a SIMD architecture such as the Connection Machine CM-2. Estimates for performance of the method include a response time of 2 seconds or less for databases up to the size of 128 Gigabytes using a CM-2 with 64K processors.

The work described by Grandi et al [56] describes a hybrid method that combines both horizontal and vertical partitioning which they assert is suitable for implementation on parallel machines. The use of the Shared Nothing architecture described by DeWitt and Gray [26] is recommended. Grandi and his colleagues point out that the horizontal partitioning method used by Stanfill and Kahle cannot support inter-query parallelism as all data needs to be accessed. The architecture of the system described is divided by three dimensions: frames (which are subsets of a signature), partitions (a horizontal fragment of frames) and blocks (a horizontal fragment of partitions). The signature file is stored in terms of the frames, each disk containing a subset of the frames (figure 2c). Thus frames are stored and can be searched in parallel while other frames are being serviced. Hence the classification of the method as being Frame-Sliced Partitioned parallel signature files. Since all frames would not be needed by a search, the method can allow inter-query parallelism as well as intra-query parallelism. While the method does overcome some of the limitations of those described above, this is at the cost of a great deal of extra complexity. This complexity in parallel systems should not be underestimated. Comparable results with the systems in this class are not available.

From the above discussion we can assert that the signature partitioning method interacts with both the type of machine used and the query parallelism directly allowable. Horizontal partitioning allows only intra query parallelism directly, while vertical partitioning and the hybrid method allow inter and intra query parallelism directly. Inter query parallelism could be supported indirectly if batch queries were used; although such would be problematic (see the discussion on false drops below). The granularity of signature files can be either signature, bit-slice or frame-slice and bit level granularity can also be used if the special hardware to work at that level is available.

The advantage of the method is that it is rather amenable to implementation on parallel computers. Since the signature matrix defined above has a regular shape we

can reduce data skew quite considerably, although we may not be able to eliminate it completely given that signatures files may not fit into main memory. There is also a much lower storage overhead of about 10% compared with 50% to 300% found in Inverted files [47]. However a serious drawback is associated with the method, the problem of false drops. Since different terms may hash to the same signature bits, collisions will often occur between query and document terms. A number of criticisms of the method have been made therefore in using the signature file method in an operational environment [37], in particular that signatures cannot support sophisticated term weighting schemes. The subsequent effect of false drops on precision and recall can be profound. A further serious problem is that position information is lost, therefore proximity operations are unavailable in the class.

4.3 *Two-Phase search*

This method has been proposed to overcome the high search cost of pattern matching and the low retrieval effectiveness of the signature method. The first phase of the search compares a signature version of the query with document signatures to create a hit list. The text arising from this hit list is then searched with the required patterns to eliminate the false drops and produce the final document result set. Since the number of documents pattern matched is greatly reduced, the increase in speed and effectiveness makes the method valuable. Parallelism can be used in both phases of the search. Two-phase searches have been implemented on SIMD machines by Pogue et al [11-15] and on a MIMD transputer network by Cringean et al [58-62]. Panagopoulos and Faloutsos [55] also recommend the method's use when using signature files. Any of the signature and pattern matching methods described above could be used.

An example of the two-phase search can best be illustrated by looking at one particular system, the transputer network program described by Cringean et al [58-62]. This system uses the process farm approach to parallelism to increase efficiency on the more computationally intensive second phase. The horizontal partitioning method is used for the first phase signature comparison. In this approach a single farmer distributes work to a number of worker processes who do the search. In the first phase the query signature is compared with document signatures (pre-loaded into memory) on a number of transputers attached to the root transputer and a hit list of documents

are recorded. In the second phase the farmer distributes the documents in the hit list to the workers, receiving the final document result set from them. A triple chain of transputers was found to be the most effective topology. Data skew in the second phase is reduced since a worker is given more work on completion of a search: waiting for all workers to search a given set of documents would reduce the system's efficiency drastically. However it should be noted that documents may need to pass through several processors before reaching the target worker, because of the layout of transputer networks. The cost in extra communication and lost computation in routing processors affects the overall efficiency of the system. In the event this was found to be a significant problem: Cringean et al [62] state that a substantial increase in communication speeds would be needed for the method to achieve its full potential. A further interesting result was that a more efficient signature search on the first phase increased the amount of pattern matching needed in the second phase.

The granularity of two phase search is rather mixed depending on signatures granularity in the first phase and documents in the second phase. Given that documents are irregular structures and signatures are regular, data skew is more prominent in the second phase of the search. The method supports intra-query parallelism for both phases. Inter-query parallelism however, could be used in the first phase if Frame-Sliced Partitioned Parallel Signature Files were used and for both phases if queries where submitted as batches. The interaction between machine type and the classification relates to the signature partitioning method for the first phase and computation granularity for the second phase.

4.4 *Inverted file*

Most commercial and academic IR systems use inverted files. The reason for this is that until now query processing has been given priority over insertions, and Inverted files provide much faster searches than other methods such as pattern matching and signatures. This is because the indexing eliminates the need for searches on many irrelevant terms. However the generation and maintenance of Inverted files is very expensive and this makes its use problematic in applications where insertions are frequent. As stated in section 4.3 the storage requirements for Inverted files are far costlier than any of the other methods reviewed in this paper. The issue of Inverted

file update is addressed more fully in section 5 below. In our description of the method below, we pay particular attention to data partitioning schemes.

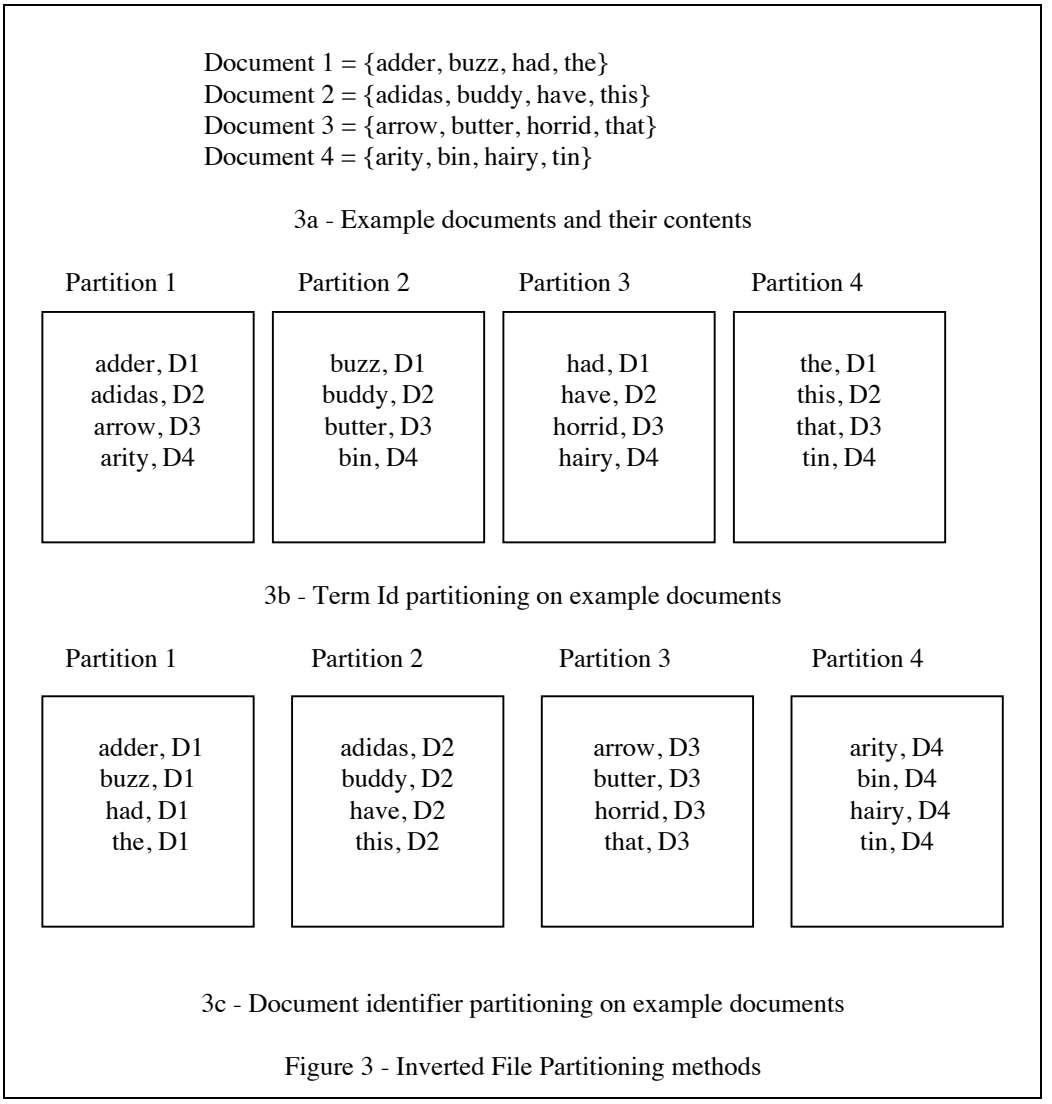


Figure 3 - Inverted File Partitioning methods

The most prominent of parallel IR systems have used Inversion as their storage technique [11,17,39,63-68]. We briefly review the structure of an Inverted file [47]: an index or dictionary file contains a list of keywords in the collection, number of documents in which that keyword occurs and a pointer to a document list: a postings file or inverted list contains the document list for all the keywords and may in some cases contain position information for each keyword in each document. There are two main Inverted file partitioning methods [69]: by term identifier and by document identifier. With document identifier partitioning the terms for a single document are placed on one disk, therefore postings for the same term are held on multiple disks.

Term identifier partitioning has all postings for a given term on one disk, therefore postings for the same document may be on multiple disks: see figure 3. Four documents are provided as examples. Term identifier partitioning is done on the first character of a word; partition 1='a', partition 2='b', partition 3='h' and partition 4='t'. Each partition is placed on one disk.

Jeong and Omiecinski [69] discuss the effect partitioning in Inverted files has on the performance of multiple disk systems. They advocate a Shared Everything approach as opposed to a Shared Nothing in order to exploit I/O parallelism. The use of a multiprocessor with shared memory is assumed. The two partitioning methods described above are considered. The results produced by simulations are that term identifier partitioning is best when the term distribution in the query is less skewed (or more uniform) and document identifier partitioning is best when term distribution is more skewed (or less uniform). Document identifier partitioning sacrifices more I/O and space in order to ensure better load balancing in a more skewed query environment. Document identifier partitioning is more expensive on I/O because multiple disk accesses have to be made. When query term distribution is a little less skewed the postings for a term can be retrieved faster since disk access times for terms are more evenly distributed. When more skewed the load balancing of the machine will be affected by large disk access times for some terms. Document identifier partitioning avoids the latter problem by providing constant disk access times so that large access times for terms with very large postings are masked. This advantage is lost in a less skewed environment and the cost is greater because multiple disks have to be consulted in document identifier partitioning (and the term accesses can be done in parallel). Inter-query parallelism cannot be done with document identifier partitioning: each query must take its turn on the disk queue. Term collection information is often needed for weighting calculations: this has an implication for the efficiency of term weighting using document identifier partitioning (see section 6.3). Based on their simulations, Jeong and Omiecinski recommend that the Shared Everything architecture be used in a medium sized Text Retrieval systems or as components in a larger Shared Nothing machine.

Tomasic and Garcia-Molina [30,31] describe hybrid methods of partitioning inverted files on distributed shared nothing systems. They assume the existence of multiple disks per single CPU. They classify distribution methods as: Disk, I/O Bus, Host and

System organisations. The Disk and System organisations are equivalent to document identifier and term identifier partitioning methods respectively. In the I/O bus organisation documents are distributed across I/O buses and inverted: this creates one inverted file per I/O bus. In the Host organisation documents are distributed to CPUs as per document identifier partitioning, but the inversion is spread across the disks connected to the CPU. Where one I/O bus exits per CPU the I/O bus organisation is equivalent to the Host organisation. Simulations of full-text system and an abstract service were done using all the organisations described: in their results the Host organisation appeared to performance well for full-text systems, while the System organisation (or term identifier partitioning) performed better on abstracts.

As can be seen from above, we can divide parallel systems which use Inverted files into two main camps, those who argue for or use term identifier partitioning (Reddaway [11], Stanfill et al [39]) and those who argue for or use document identifier partitioning (Hawking [20], Aalbersberg & Sijstermans [68], Stanfill & Thau [63] and Hollaar [8]). There are a number of factors other than those discussed by Jeong and Omiecinski above which determine the method of partitioning: indexing, insertion, load balancing and subject division.

The time to build indexes in term identifier partitioning is longer than for document identifier partitioning. The reason for this differential is simple: in document identifier partitioning the build is kept local to the disk and no interprocessor communication is needed to send data to a particular location. As the size of data grows in term identifier partitioning this interprocessor communication increases, thereby increasing the differential: but this assumes that data transfer rates across a network will remain static. Therefore both the size of a database and the data transfer rate will determine which method would be better with respect to building indexes. For those applications which require fast index building, then document identifier partitioning would be better. Depending on the factors considered below term identifier partitioning builds could be better if users can wait a period.

The cost of inserting documents in the database would also appear to favour document identifier partitioning for applications with a higher update rate, since only one section of the inversion needs to be manipulated i.e. a document update is sent to one processor rather than several thus reducing the level of interprocessor communication. However the speed of single document insertions may be faster in

term identifier partitioning depending on the term distribution of the document. This would favour applications with a lesser update rate. In the worse case insertions in term identifier partitioning degrade to those of document identifier partitioning, although it is more likely that some processors will need to update more term information than others reducing the overall efficiency of a single insertion. It should be clear that both intra and inter insertion parallelism are available for the term identifier partitioning method, whilst the document identifier partitioning method can only utilise inter insertion parallelism.

We can see from the discussion above that load balancing on insertion is an important issue. Load balancing on search is also affected by the partitioning method. Document identifier partitioning allows for a much better load balance, since computations are spread more equally across the processors as all partitions need to be searched. All partitions may not need to be searched in term identifier partitioning. Term identifier partitioning therefore allows both intra and inter query parallelism, while document identifier partitioning only allows intra query parallelism. In applications with very high query rates term identifier partitioning would be preferred.

Subject division could be useful for separating document into their various subject areas, thereby reducing the number of irrelevant documents searched for a given query. This has implications for the retrieval efficiency and effectiveness of the system. Subject division would introduce the inter-query parallelism facility to document identifier partitioning, since only a subset of subjects need be searched allowing other queries to be serviced on disjoint subjects. However load balancing may be adversely affected if one given subject was much larger or more popular than others, thereby reducing overall efficiency. Subject division would not be suitable for implementation with term identifier partitioning since it would greatly increase the cost of maintaining the inverted file, both in space and time. We would need to sub-divide each inverted file partition into sub-partitions which could run into hundreds of subjects or increase the size of the postings file by recording the subject for every posting. Some work in the area of subject division using inverted files has been done for a selective dissemination of information (SDI) service and is described by Kapaleaswaran and Rajaraman [70].

The granularity of Inverted files is based on the postings of the inverted list. Therefore granularity is much finer than the approaches described above (if we

discount the possible use of special hardware to match at the bit level). One of the main reasons for the success of SIMD machines in parallel IR, is that they are very good at computing with this level of granularity. SIMD machines cannot normally handle inter query parallelism with inverted files, but a method of using several DAPs connected together has been put forward [11] which would overcome this limitation. Three systems which use inverted files are described in the case studies section (7) in more detail.

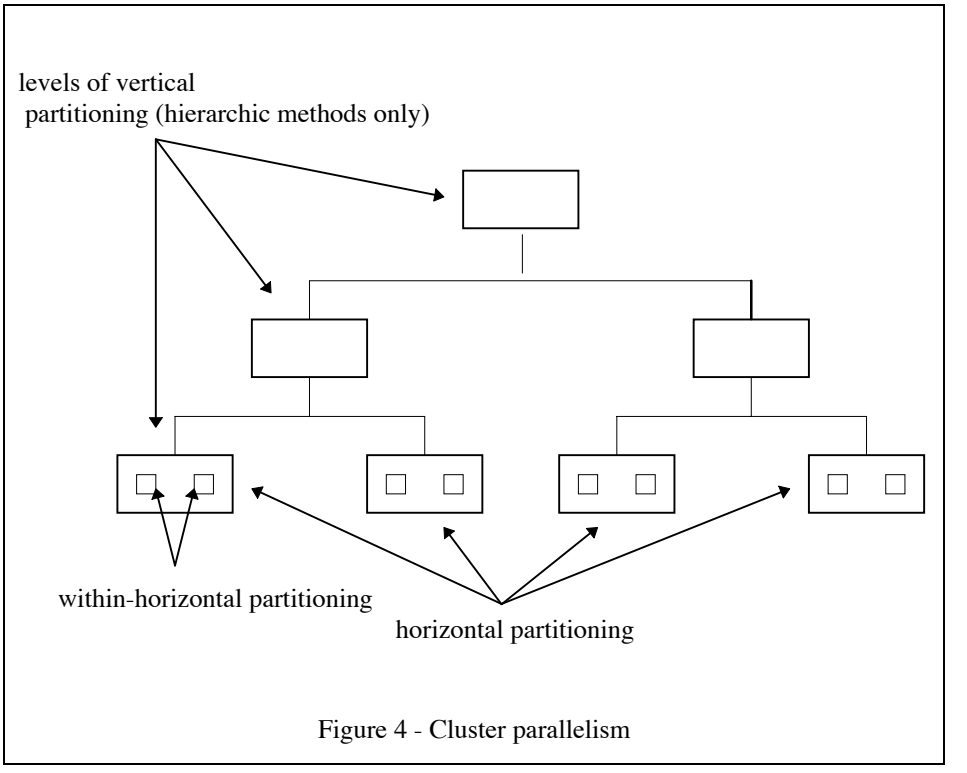


Figure 4 - Cluster parallelism

4.5 *Clustering*

Clustering is a method of identifying similar documents, based on a given similarity method. The documents are organised into groups or clusters, which in turn can consist of a single centroid and document vectors belonging to that cluster [71]. There is therefore parallelism in the Clusters as well as between them: we term this horizontal and within-horizontal partitioning. Very fine grain parallelism (e.g. at the posting level) is also available within document vectors. A further issue is the type of Cluster: they can be either hierarchic or non-hierarchic. Hierarchic methods introduce a further level of parallelism: we term this vertical partitioning. Figure 4 shows the forms of parallelism available in Clustering. It should be noted that clusters can be overlapping and non-overlapping. We describe below parallel methods for generating and searching in the clustering method.

The generation of clusters are computationally very intensive: orders of $O(n^2)$ to $O(n^5)$ are not unknown. This makes their implementation on sequential machines problematic. MacLeod and Robertson [43] describe a neural network algorithm (called the MacLeod algorithm) for document clustering using non-hierarchic methods. Neural networks are inherently parallel: Networks can be divided in layers and nodes within layers which allow parallelism in two directions. Parallelism is used in the MacLeod algorithm when a each document vector is compared with the current set of clusters, iterating until a suitable cluster has been found or learned.

Rasmussen and Willett [72] describe parallel computing for various hierarchic *agglomerative* clustering methods. Hierarchical clustering can be represented by binary trees where nodes are clusters and the position in the binary tree represents the similarity measure between objects. Agglomerative clustering consists of building the tree bottom up; the alternative is *diverse* clustering with builds the tree top down. There are three implementation approaches described:

a) stored matrix - N * N matrix containing pairwise distance values;

b) stored data - list of pairwise values N-1 elements;

c) sorted matrix - a distance matrix is constructed and sorted and then used to construct the hierarchy.

They use a method called single linkage minimum variance, where single linkage is related to minimum spanning trees. The SLINK Prim-Dijkstra and Ward algorithms are used for clustering. The SLINK algorithm only has parallelism in the calculation

of the current row of the distance matrix. The Prim-Dijkstra is almost entirely parallel except for storage of link information. The Ward uses the nearest neighbour method using recomputed nearest neighbour i.e. chain of related objects; finding the nearest neighbour is done in parallel. The parallel SLINK algorithm performed less efficiently than its sequential counterpart, a slowdown being recorded. The parallel Prim-Dijkstra performed much better in relation to its sequential counterpart with speedups of 3.6 to 6.0 recorded. The Ward speedups ranged from 2.9 to 4.0. They compared the results from an IBM 3083/BX3 mainframe against the ICL DAP and conclude that parallelism can provide significant speedups over serial systems in this type of clustering for large datasets.

While there are clearly defined partitioning methods for clustering, the arbitrary shapes of each of the levels will effect the search efficiency of the algorithm e.g. clusters do not have the same number of document vectors or a hierarchy may not have regular binary tree like structure. Organising the clusters (and hierarchies where necessary) is therefore essential for the efficient search in this method. Frieder and Siegelmann [73,74] formally argue that an optimal algorithm for assigning clusters to processors is NP complete and is therefore unusable. They propose a heuristic using genetic algorithms to address the problem. The algorithm terminates when either all document allocations are equal or after 1000 generations. Other researchers propose more conventional techniques.

Ozkarahan discusses search on non-hierarchic document clusters on the RAP.3 system [75]. The clusters of document vectors and a centroid representing the vectors are distributed to a number of processors. A query vector is applied to the centroids, which if successful applies a second search to the document vectors in that cluster. While some regard this as useful, it is unlikely that the method would be able to compete in speed with inverted files. In any case the insertion of documents is likely to be prohibitively expensive. The RAP.3 system deviates from other systems in this review as the integrated multimedia application area is addressed.

Sharma [76] describes a generic machine for parallel IR using clustering techniques for both non-hierarchical and hierarchical methods. The hypercube topology is used together with dedicated disks for each node in the hypercube (i.e. shared nothing). The key is to distribute a subset of document clusters, to get the best load balance on search. Two schemes for partitioning clusters on a hypercube are described: one said

to be for increasing efficiency and one of increasing effectiveness. In the efficiency algorithm closely related clusters are assigned to different sub-cubes such that no of documents is equal in all sub-cubes. Within a sub-cube a cluster is spread across nodes, with the centroid assigned to one node. In the effectiveness algorithm clusters are recursively distributed across sub-cubes, each sub-cube have a smaller dimension than its parent. A hierarchical clustering algorithm is used, mapping the hierarchy to the hypercube. All levels of parallelism for clustering are used in these proposed schemes. The search consists of the broadcast of a query and the application of the query to the document database. In the efficiency algorithm the query is received at each node and comparisons are done concurrently. Similarity values for clusters (centriods) are collected and sorted and sent to a designated node which chooses the highest ranked clusters; these are requested from the relevant locations. In the effectiveness algorithm the query is received at each node and comparisons are done concurrently, similarity values at all levels of the hierarchy being calculated. The results are transmitted up the hierarchy and on this basis the highest ranked documents are chosen. The simulation shows that as cluster levels increase, the efficiency scheme response time remains static, while effectiveness scheme seems to increase dramatically. In this case Amdahl's law (the asymptotic limit for the computation) hits the efficiency scheme at 128 processors and the effectiveness scheme at 1024 processors.

The granularity of the clustering approach can vary; either the cluster or the vector or even elements of a vector if an array processor such as the DAP is available. Both inter and intra query parallelism for search are available in the method. It is difficult to comment on the interaction between the machine type and the method, because of the multiplicity of clustering algorithms available. The arbitrary shape of the clustering algorithm determines the level of data skew and hence the search efficiency. Because of the expense of generating clusters, it is unlikely to be able to compete with Inverted files: unless some benefit in retrieval efficiency can be demonstrated.

4.6 *Connectionist approaches*

These approaches use a network model to represent information in an IR system [1]. Many are related to the 'neural network' and 'spreading activation' areas of computation. They are inherently parallel, but extremely complex and poorly understood. Because of this their implementation on parallel computers is difficult and little work has been done in the area: research has concentrated on sequential implementations as a consequence [77,78]. The MacLeod and Robertson algorithm [43] described in section 4.5 can also be placed concurrently in this class. It should be noted that these researchers take a very different approach to others described in this review. Because of the complexity of these methods we do not attempt to describe data partitioning, granularity or query parallelism for connectionist approaches.

One particular connectionist system is the PTHOMAS system described by Oddy and Balakrishnan [79] and has been implemented on the Connection Machine. The theoretical idea behind PTHOMAS is to represent a holistic view of the documents and their relationships. The method uses a network structure of nodes (documents, authors, terms) where the arcs (edges) between these entities represent a relationship in the index and thesaurus. The network is a global graph representing the universe of the database. The user sees a context graph which is a subset of the global graph and is created by user action. Various component graphs may be discarded in the user interaction with the system. The algorithm used is computationally very intensive: a database with 10,000 document abstracts would create a network with 1 million nodes/edges. Oddy and Balakrishnan have not addressed the issue of how to implement these ideas/methods realistically for large collections and therefore we do not see the PTHOMAS as being a practical proposition for the foreseeable future.

4.7 *Other approaches*

There are a number of different approaches to parallel information retrieval which do not fit easily into the classes described above. We therefore describe below some other work, both practical and theoretical. These include vector processing, hybrid inversion, functional programming and relational database. A vector processing system is the subject of a critique in section 7.4. Given the variety of approaches in this section we will not attempt to describe the interaction between architecture, the algorithms and the types of query parallelism used.

A. *Vector Processing*. Stewart and Willett [80] describe an algorithm for nearest neighbour search using a multi-dimensional binary search tree, using networked microprocessors. Documents are represented by vectors, as is the query: the vector contains identifiers of terms in that document. Document collection is represented as a binary tree with the nodes associated with document term vectors (all nodes at the same level of the tree having the same vector) and the leaves having buckets with documents sets. Similar vectors are inserted in the left tree and dissimilar are inserted in the right tree. Query search is done in the same manner. An upperbound value is set and the algorithm backtracks using the value to find relevant buckets. The search is bounded by $O((\log N)^k)$ where k is a collection dependant constant. The level of k determines the amount of backtracking and hence the efficiency of the search. A special simulation language for the simulation of queuing systems was used to produce the results. Search is done by broadcasting a query down the tree, the answer being broadcast back up in the opposite direction: backtracking to nodes in the tree is done where necessary. The "Overlap co-efficient" is used as the similarity measure. The level of speedup deterioration was found to vary widely and were collection dependant.

B. *Hybrid Inversion*. Yount et al [29] describe the MARS system which they have implemented to store medical records. The system contains 850,000 medical records, 2.5 million medical references and 500 million indexed words. The system runs in a standard UNIX distributed environment, with the machines linked together by ethernet. The system uses the Shared Nothing architecture. The MARS system uses many of the concepts and mechanisms of distributed systems such as threads, remote procedure calls (RPC), external data representation (XDR) and the client/server model etc. Each text word is classed as an instance, and is stored in one of the archives which are distributed amongst servers residing on different machines. The instance (or posting) is a fundamental unit for locating and manipulating records. The instances have a segment id number (SID) to identify a host, a record id (RID) for a given record and word count (WC) to locate individual elements of a word in a record. The system uses a hybrid inversion method utilising a dynamically changing hash function to identify word to word id and inverse mappings.

C. *Functional Programming*. Deerwester et al [81] describe an architecture which uses a server as an interpreter for a functional programming language that uses lazy evaluation. Clients can make requests to multiple servers, therefore the language can be evaluated in parallel. In particular the processing of inverted lists, which can in some cases be very large, is addressed. It is stated that without lazy evaluation of lists much extra computation is needed where examination of intermediate results suggests that processing of the lists is unnecessary. Such also has implications for space complexity, where the intermediate results need to be stored. They state that functional programming is a useful way to implement the lazy evaluation of lists to prevent the extra time and space complexity which may occur with certain queries.

D. *Relational Databases*. A great deal of research has been done on using parallel computing for relational databases [26]. Experiments using parallel relational databases for have been reported at TREC-3 [82] and TREC-4 [83]. The guiding principle of this work is that while parallel relational databases are common, parallel IR systems are rare. An inverted file structure is modeled using relations and keyword searches are done using SQL. The parallel database machine used is the AT&T DBC-1012 Model 4 (formerly Teradata). The I/O penalty of using relational databases in IR is addressed by using a query reduction technique based on term selectivity, which according to the results given does not affect precision and recall adversely. Clustered primary keys are used to reduce I/O even further, by placing inversion data on contiguous data pages. However, it is unlikely that parallel relational databases would be able to compete in speed with parallel IR systems because of the superior I/O performance of the latter: Inverted files only need 1 disk read to access term information, while relational databases usually need many more because of the B-Tree structure used in them.

5. CHOOSING AN APPROACH

We have seen the motivations for using parallelism in IR and some of the methods which have been used. In the context of the information given we describe a rationale for choosing a parallel IR approach. We assume that one or more of the reasons described above exists for choosing parallel IR systems in the first instance.

The central issue behind choosing an approach is that of index maintenance, in particular of the insertion rate compared with the query rate [36]. A further issue is that of index generation: Hawking [23] points out that building indexes for Inverted files with the size of 8192 Gigabytes would take so long that the document retrieved would only ever be of historic interest. We therefore suggest some criteria for choosing one of the parallel IR approaches described in section 4.

If searches such as regular expressions are required, then the pattern match method would be the most suitable. Regular expressions are difficult to implement on signature and Inverted file methods and would be restricted in the two-phase search.

If the application requires a high insertion rate compared with query throughput, we would suggest that the two-phase search be used. Insertion of documents is much cheaper than Inverted files and queries are therefore much less likely to be affected by delays engendered by insertion. Where other types of document maintenance are required such as document alteration or deletion, or all three maintenance operations, then the two-phase search would become essential. It is generally agreed that using Inverted files with highly dynamic database is problematic, because of their high maintenance cost. Block deletions are not an issue since they are relatively inexpensive.

However where the query rate exceeds the insertion rate the use of Inverted files is recommended. It is possible to reduce the cost of maintaining indexes by using parallelism [26] and thereby offer much faster access to documents than would normally be possible with the method. For very large databases it is possible to reduce downtime by using parallelism to insert documents in batches and increase system availability. Where the availability of documents in not such an important issue, batch updates would be preferred.

What of the other methods described in section 4 such as clustering and connectionist approaches? Because of the extra computation needed we would only

recommend their use if some gain in retrieval efficiency was found, using empirical experiment based on users relevance judgements. In some cases however we would not recommend the use of some methods under any circumstances: in particular the application of parallel relational databases to IR. Normalised tables increase the size of the Inversion dramatically (which can sometimes be large in any case) and the I/O problem already stated in section 4.7 above can be a considerable bottleneck. The use of parallel relational databases does not bring benefits in terms of retrieval effectiveness or efficiency.

It has been noted above that the reason for the dominance of Inverted files as a method in IR has been the ability to service queries far quicker than the other methods. Our criterion for choosing a parallel IR system is also based on this reason, but it should be noted that some applications such as News agencies live and die by the speed with which they can deliver textual information to their customers. This does not greatly affect the choice of a method, but may reflect a change in emphasis from past requirements.

6. RETRIEVAL MODELS USED IN PARALLEL IR SYSTEMS

Information Retrieval systems use models in order to extract relevant information from text databases. The application of these different models can have an effect on both the retrieval effectiveness and efficiency of Parallel IR systems, it is therefore important to consider them. We divide the models up into boolean, proximity, term weighting and regular expressions. They are discussed in turn below.

6.1 *Boolean model*

The boolean model is dominant in commercial IR systems, and most of the mainstream systems described in this review offer facilities for users to submit boolean queries. They have been implemented on systems such as the CM-2 [38] using the signature method, the DAP [11] and POOMA [68] machines using the Inverted file method and PADRE [21] using the pattern matching method. PADRE allows union, intersection and difference operations on match sets, but these are equivalent to OR, AND and AND NOT boolean operations. The MARS [29] system also uses the boolean model as the basis for its query language. Parallel systems

cannot improve the effectiveness of queries using this model, and depend on the user to generate effective queries. Naive users can find generating effective queries using the boolean model very difficult. Retrieval efficiency could be increased by parallelism, whether it be increase in speed on pattern match or fast set manipulation on inverted lists.

6.2 *Proximity models*

A very useful extension to the boolean model is proximity operations. They are used to find text atoms which are within a specified distance of each other e.g. next to each other (adjacent), in the same sentence or within a given character distance. Among the systems which use proximity models are the DAP [11], pattern matching in PADRE [21] and MARS [29]. The PADRE system provides the most detailed information on the proximity operations it allows. These include followed by (fby), not followed by (not fby) and a combined proximity/weight scheme called Z-mode [20] (znear). The fby operation finds matches on terms which are within a given number of characters of each other. The not fby operation finds text in which terms are not within a given distance. The znear operation uses proximity spans to calculate relevance scores (we can therefore place this operation concurrently in term weighting models). As with boolean models, improvements in retrieval effectiveness using parallel computing are not found: but retrieval efficiency could be improved if overall efficiency is not reduced by extra interprocessor communication or load imbalance.

6.3 *Term weighting models*

One of the main methods used to improve retrieval effectiveness is to utilise one of the myriad term weightings schemes that are available. The dominant scheme has been the vector processing model with systems such as RAP.3 [75], the DowQuest [63], Transputer Networks [58-62,84] and POOMA [68], all using it in various forms. PADRE [20] offers a number of weighting schemes based on the inverse document frequency (IDF) measure. These models may use unnormalised [20] or normalised [24,63,68,84] term weighting. Cringean et al [58-62] do not specify the normalisation method. Others such as Reddaway [11] and Jeong and Omiecinski [69] do not specify a weighting scheme in their discussion of term scoring in their papers. A very important issue will have a critical effect on the efficiency of a term weighting scheme

on a parallel architecture: some schemes require collection information to calculate the weights. If this information does not reside in one place i.e. a processor and its resident disk, the parallel machine needs to use interprocessor communication to merge the data held separately into a single figure. This bottleneck can affect the efficiency of the term weighting calculation. Many parallel machines provide facilities to do just this e.g. the DAP [7] and Fujitsu AP1000 [16] in the form of global operations. Where this special hardware does not exist, the interprocessor communication can reduce efficiency drastically. Unlike the two models discussed above, parallel implementation of term weighting may allow an improved level of retrieval effectiveness if the improvement in efficiency allows weighting methods to be used which are computationally more complex.

6.4 *Regular expressions*

Regular expressions give a user the ability to search for complex patterns in a single statement. They can be very computationally intensive and are best implemented on raw text. Examples of work using or proposing regular expression in pattern matching can be found in Pogue & Willett [12], Hawking [19] and Skillicorn [42]. They can be undoubtedly very powerful in the hands of a very experienced user, but naive users may find them difficult to use effectively. Parallel computing could improve retrieval efficiency quite considerably, but we do not see how it could improve retrieval effectiveness.

7. CASE STUDIES - "STATE OF THE ART"

We present four systems below which are regarded as the most prominent of those discussed: two of them because they have been commercially successful and two because they are the most up to date systems currently being used in research laboratories. Detailed information on the commercial systems is however limited. In our discussion in the case studies we describe the suitability of each system for the task, storage methods and granularity.

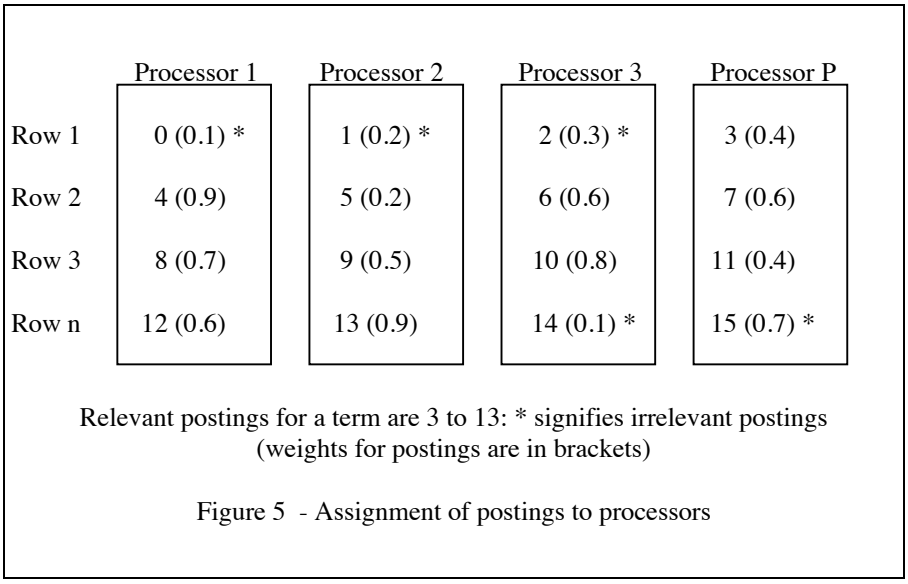
7.1 *DAPText*

DAPText [11] is a commercially successful parallel text retrieval systems used by Reuters for their text retrieval purposes. The system uses a range of compression techniques on the posting lists for terms of varying hit rates. Those terms with the highest hit rates have postings represented as bit maps, 8 bit postings and 16 bit postings, whilst 24 bit postings are used for rare terms. The higher the hit rate for a term the more compact the compression method. Boolean operations on bit maps are reported to be very fast on the DAP. The main aim of the system is to provide very fast query processing on common terms, since merges on them are more computationally intensive than rarer terms. Position data is also held (in 12 bits), but is kept separately from the inverted list. The reasons for holding position data separately are for efficiency on queries which do not require position data and the variety of compression techniques used. Updates on the indexes are not done immediately: documents are added to a separate area of the DAP memory and merged with the main index data in a given timeframe. Processing of documents takes half a second for those of an unspecified average size. The DAP 610 can handle 35 boolean queries a second. Each query is handled one at a time, since SIMD machines do not allow separate threads of execution. Therefore no inter-query parallelism is possible, unless several DAPs are connected together.

Information about the system is limited. There is very little information on how keyword and inverted lists are manipulated. The system appears to offer a very fast search on the back of the compression techniques described. The granularity of the computations are determined therefore by the required compression method for a given term. There is no discussion on those terms whose distributions may hover between different compression methods, and the subsequent effect this may have on performance. Some recent work on using hypertext and the DapText system is reported by Wilson [85,86].

7.2 *DowQuest*

The DowQuest system is also a commercially successful system. The Dow Jones News Retrieval Service uses the system for its Text Retrieval needs [29]. The algorithms and data structures for the system are described by Stanfill & Thau [63] and Stanfill [64,65]. We outline some related work done by Thinking Machines which is described in Stanfill et al [39] and Stanfill [64]: the contrast between the two algorithms is instructive. We also describe some further work done on an IR testbed for a more recent version of the Connection Machine.



Relevant postings for a term are 3 to 13: * signifies irrelevant postings
(weights for postings are in brackets)

Figure 5 - Assignment of postings to processors

The algorithm described in Stanfill et al [39] works by multiplying a query weight with stored postings weights in parallel and sending the result to a mailbox somewhere on the machine. The term identifier partitioning method is used. Using a data map (the keyword index), rows of postings are identified and placed in memory ready for computation. Processors are given an equal number of postings (n). The algorithm then iterates through each posting row of the processor i.e. from row 1 to n, calculating weights for terms if and only if the posting in that row is identified as being relevant: otherwise the processor is deactivated (see figure 4). The weights are then routed to the relevant mailbox in the machine after an iteration using a Send and Add command. When weights have been computed, the top documents are identified by sorting the weights in the mailboxes. This mailbox algorithm has been criticised by Reddaway [11] who points out that term distribution will have an impact on its

efficiency. If the postings lists are too large to be fitted in the machine at one go, the remaining postings can be loaded in from disk and processing can start again from row 0. SIMD machines are very good at this kind of fine-grain computing. However the algorithm suffers from a data skew problem when a row of postings only has a small number of active processors e.g. one or two in a 64k processor machine. The effect on efficiency can be drastic, reducing the complexity to that of sequential machines in the worst case. To address this problem, partitioned posting file methods are discussed [63-65].

The partitioned posting file method described in Stanfill & Thau [63], does not eliminate data skew but does reduce it considerably. Essentially postings are partitioned such that all term postings for a document are handled by a single machine node: thus the document identifier partitioning method is used. This eliminates the need for the routing process for the mailbox algorithm. Postings are placed into blocks of a partition. The data map is used to identify the required partitions. The partitions are then loaded into memory and computed in parallel. The algorithm iterates through the partitions until a weight for every hit document has been calculated. The granularity of the computation is still the posting. Extra space is added to the postings file in order to retain alignment as far as possible. As with the DAP the system would appear to offer very fast search facilities.

DowQuest was written for the CM-2 version of the Connection Machine. A prototype [66,67] was written for the Connection Machine CM-5: a more powerful machine with a hybrid SIMD/MIMD architecture. Massand and Stanfill [66] and Linoff and Stanfill [67] describe methods and data structures implemented on an IR testbed for the CM-5. They take the standard boolean model and extended it with proximity operators. Techniques for distributed databases are considered in particular the problem of term weighting across distributed collections. Compression methods are used to reduce the size of the inverted file i.e. pre-fix omission, run length encoding and n-s encoding: compression is applied to position data, but not to weighing data. They claim the decreased time in I/O can fully compensate for uncompress computation (based on a study of two corpus; the King James bible which is 4.5 Mbytes and a sample of Wall St Journal articles which is 12.3 Mbytes). The issue of updates is considered as well as deletes: they use an in-core technique for the text database using the document id partitioning method for inversion. Fixed sized

blocks are used to distribute documents and re-adjust to text boundary accordingly (each processor looks after its own document set). A two pass index algorithm is used: see figure 6. This algorithm took 20 minutes in comparison with the 90 or so seconds on the Fujitsu AP1000 reported by Hawking running the PADRE system [16]. Part of the differential could be the cost of compression, and part in having to do the indexing twice. In the event the prototype or test-bed did not become a product.

1: Postings are generated, counted and discarded. This information is used to calculate the space needed for posting and space is allocated for each Inverted List.
2: Text is parsed again & indexed again from scratch, compressed & the Inverted Lists are put into the pre-allocated blocks

Figure 6 - Algorithm for generating compressed Inverted File

7.3 *PADRE*

More information is available on PADRE and its precursors than any other system covered in this review. We have already imparted much information on the system ranging from the hardware it uses (section 2.2), methods of operation (section 4.1 and 4.4) and query models available for the system (section 6). We therefore restrict our discussion to the history and philosophy of PADRE.

The system started life as PADDY [23,24] and concentrated on linguistic and lexicographic research on the *Concise Oxford English Dictionary* structured in SGML. Searches are based on the PAT indexing method [87], to implement pattern match, proximity and regular expression operations. Results from searches on the indexes show speedups ranging from 30 to 1000, where the speed of indexed matching depends largely on hit matches [23]. There is much discussion on the time to load data, a problem overcome by the introduction of the HiDIOS file system. A vision of the libraries of the future is given by Hawking [24] who argues that a number of advantages lie with using parallel supercomputers including: libraries would be open for much longer, a number of people could read the same book, books are never lost or mis-shelved, catalogues are never out of data etc. He does however point out that there may be many practical reasons, such as legal and financial, which may prevent the complete replacement of libraries by parallel supercomputers.

The *ftr* system [25] builds on work done in PADDY and while retaining its capabilities is oriented towards more conventional IR problems such as retrieving text. A user interface called *retrieve* is introduced in order to provide a more user friendly access to the applications services, rather than a command line interpreter (although this is still available in *ftr*). A significant decrease in load times is recorded for *ftr* over PADDY. The system also has the ability to load more than one text database.

The PADRE system retains many of the features of both *ftr* and PADDY, while introducing others such as inversion of text [16], term weighting [17]. natural language processing techniques [18], multiple user facilities [19] and proximity spans (z-mode) [20]. The document identifier partitioning method is used with partitioned indexes and postings.

The reasoning behind the partitioning method is to provide fast update on Inverted files while providing fast responses to user queries. Near linear speedups for indexing are reported. The searches on indexes are reported as being constant, whereas the search time for pattern matches decreases with increase in the number of AP1000 cells. Responsiveness to additions and deletions to a text corpus are recorded. Using 509 Mbytes from the Wall Street journal and 10 Mbytes of Associated Press reports a merge time of 18.7 seconds, of which half was the approximate load time from the host. A time of 9.2 seconds is reported for the deletion of all documents with the word 'computer' in them: this reduced the Wall Street journal collection by 57 Mbytes. The implementation of time-outs on searches [19] is recommended to ensure reasonable responses times for users and to avoid 'killer queries' which can greatly reduce system throughput.

As reported in section 2.2 the AP1000 system has hardware support for global reduction operations, which are useful for calculating collection information, given that such does not reside in one location on the machine. It is stated that PADRE could be ported to workstation clusters, re-coding the message passing in software libraries such as MPI and PVM [16]. Without the hardware support provided by the AP1000, it is hard to see how term weighting methods could be efficiently implemented in MPI or PVM with the style of data partitioning used in PADRE.

While PADRE does not explicitly provide query expansion facilities using relevance feedback [19], it is possible to use the technique providing a client has the ability to offer the service. It is further stated that relevance feedback is unlikely to benefit

greatly from parallelism. This largely depends on the method of relevance feedback being used (see section 8 on further work for more details). PADRE does not perform the term operation stemming [20].

7.4 *FIRE*

Efraimidis et al [84] describe a system called FIRE which uses a transputer based supercomputer to implement a parallel IR system based on the vector space model. They use an automatically constructed thesaurus based on a connected components evaluation algorithm. They appear to be confused as to the difference between implementation and storage methods, and retrieval models. They state that the vector space model is used as the basis for the retrieval task rather than Inverted Files or Signature Files. Since the vector space model can be implemented on Inverted Files, it is difficult to know precisely what they mean by this. Hence our reluctance to place their algorithm in a class of its own in section 4.

Their basic approach is either to keep the vectors in main memory or to load vectors in chunks, and then to compare them with a query using the cosine similarity function. There is no discussion of vector storage and insertion costs with respect static or dynamic text databases. An argument for their method could be that the insertion of a document vector to the end of a vector file is much less expensive than that of posting information to an Inverted file and would therefore be good for dynamic text environments. They refer to Stone [36] who discusses the offset of computation against storage and maintenance costs in detail, but without justifying the method of storing vectors separately, it is hard to see how the FIRE system avoids falling foul of his arguments. While the system may offer speedup in an absolute sense, a sequential Inverted file system may offer a better performance.

Experiments are conducted on a simulated large document collection, actually constructed by duplicating the (small) CRANFIELD collection X times. Their argument that an X-times CRANFIELD collection can effectively represent a typical large scale computational load does not hold water, since the computation does not reflect a realistic or practical problem. Term distributions together with the number of hits are important factors. The number of hits per term will vary with the term distribution. There is no guarantee that a collection of size 1 will have the same term distribution as size X. Therefore we believe that the results given in respect of

retrieval efficiency should be treated with a great deal of caution. Zipf's law cannot apply to such duplicated collections. Our doubts were confirmed when they state that performance is affected when more documents (hits) are found: a more representative collection may vary greatly in its term distribution, thereby affecting overall performance. With this and the other problems described above, at this point we see no use for the FIRE system in an operational environment, unless it can be demonstrated that insertion of document vectors is cheaper than that the insertion of documents postings into Inverted files.

8. FURTHER RESEARCH

A number of very important issues in Parallel IR have yet to be addressed. This includes Concurrency Control on Inverted files, fast update of Inverted files in dynamic environments, relevance feedback, the portability of parallel text retrieval systems, the large search space in the Robertson/Sparck Jones Probabilistic model referred to above, extended boolean models and connectionist approaches.

To be able to service updates and multiple-queries simultaneously, an effective Concurrency Control mechanism must be used. It has been argued that Inverted file maintenance, where the update rate is high, is very expensive and other methods such as the two-phase algorithm would be more suitable. Whilst we do not disagree with the argument, we believe that there is scope for using parallel computing to reduce the performance penalty involved in Inverted file maintenance. In particular it is argued that updating the index for each individual arriving document is inefficient [88]. Many systems such as the DAP [11] do not update the main index, keeping new document separately and merging them at a quiet time e.g. overnight. This is problematic in certain applications where quiet times do not occur since queries and updates are being delivered to the system 24 hours a day. Such applications cannot afford downtime, therefore keeping new document updates separate from the main body of the index is not practical. Further research is needed to using parallel computing for efficient update on inverted files and concurrency control mechanisms on the Inverted file to prevent loss of retrieval efficiency and effectiveness [89]. The shared nothing architecture described in section 2.3 is regarded as being useful in the most part to overcome the I/O bottleneck. The MultiText system uses an alternative data structure

called skip lists and its own memory management routines to maintain the inversion [90] in a distributed processing environment.

Relevance feedback has long been recognised as a mechanism for improving retrieval effectiveness in Text Retrieval systems [91]. In spite of its importance, very little work has been done in the area apart from that on the Seed system [38]. The constituent parts of relevance feedback are parsing, query formulation and search. Work would be most useful in query formulation and search. In particular it has been reported by Robertson et al [32] that an alternative to term selection based on a ranking method would be to evaluate every possible combination of terms on an input set to see which was most effective. The computation needed to evaluate these terms combinations is very intensive and the application of parallel computing could be very useful.

Most of the systems described above have at least one aspect that they all share: very few of them can be ported to other architectures, particularly the SIMD systems. The MARS [29] and MultiText [90] systems, having been written for distributed technology, seems to be the only ones which addresses the issue. This is a problem which is applicable to parallel computing as a whole, and has been a major obstacle to its acceptance. The reason for this is that many algorithms have to be specifically optimised for a specific architecture, as there is no general model of parallel computing. The consequence is that portability of parallel systems needs to be offset against efficiency, one of the main motivating factors for parallelism in the first place. We believe that work in the area of portable and parallel IR systems is merited in order to examine the offset against efficiency, tackle the offset and hopefully provide some useful information for parallel computing as a whole. In connection with the issue of portability and the general direction of parallel computing as a whole, further work in using workstation clusters for parallel IR is regarded as important.

To the best of our knowledge, no work has been done on applying parallel computing to extended boolean models such as MMM, P-NORM and Paice [41]. The MMM and Paice models use fuzzy set theory, while the P-NORM model uses a distance based theory. The models have been shown to produce better results than ordinary boolean systems at the cost of extra computation. In the case of P-NORM this computational cost is very large. Work in the area of applying parallel techniques to these models is merited. The large search space for passage retrieval in the Probabilistic model often

referred to in this review also merits investigation. Both the large search space and the extended boolean models provide the possibility of an increase retrieval effectiveness: therefore evaluation of these methods in terms of precision and recall is regarded as essential.

Rasmussen [1] identified the need for more work in the area of connectionist approaches, pointing out that there had been very little work at that point in the intersection between network models in IR and parallel computing for network models outside of IR. To the best of our knowledge there has been little further progress in the area, and Rasmussens statement still holds true.

9. SUMMARY AND CONCLUSION

This review has attempted an overview of the application of parallel computing to Information Retrieval systems. We describe a classification much used in parallelism and describe some of the architectures which have been used to implement parallel IR systems. Issues such as the implication of I/O on different architectures are discussed. We describe a classification of approaches to IR due to Rasmussen [1] which includes pattern matching, signature/surrogate coding, two phase search inverted file clustering and connectionist approaches. The importance of such issues as data partitioning and data skew are stressed in the discussion of each class. Other approaches such as parallel relational databases are also described. We describe the motivation for using parallel computing in IR as being good response times for users providing added retrieval efficiency, scaleup and machine efficiency on very large databases, allow for the use of superior algorithms (which provide a higher level of retrieval effectiveness) and lower search cost. In contrast we do not believe that parallel computing can be usefully applied at present to small databases with a small user base. We describe a methodology for choosing an approach based on criteria within the framework of the motivations. The retrieval models used in parallel IR systems such as boolean, proximity, term weighting and regular expressions are described as is the impact of parallel computing on the retrieval effectiveness and efficiency of the models. The case study section gives detailed information on the DAPText, DowQuest, PADRE and FIRE parallel IR systems. For further information on many of the systems described in this paper, the reader is referred to Rasmussen [40] a special issue on

parallel processing in IR as well as Willett and Rasmussen [92] for a large body of work done on the DAP. Further work needed in the area is also described.

There is some evidence in the literature of an increase in retrieval efficiency of IR systems through the use of parallel computing. There is also evidence that parallel IR systems have had some commercial success. In spite of this success we have no general model of parallelism for Information Retrieval systems. We believe that there is scope for more work in the area and such is well worth pursuing.

The current trend, which we believe to be an appropriate development, is for parallel computing to be based on standard workstations, networked using standard methods. This is a very much more economic approach than using special purpose hardware such as specially built parallel hardware or very large parallel machinery.

We intend to do work in most of the areas described in the further work section. We have already published some work in the area of Concurrency Control on inverted files [89] and work is currently in progress on a parallel IR system which will embody many of our ideas. We are encouraged by the success of parallel computing in IR and believe that it will continue to provide impetus to both research and commercial applications.

ACKNOWLEDGEMENTS

This research is supported by the Department for Education and Employment, grant number IS96/4197. We are grateful to Ephraim Vishniac and Dennis Parkinson for information on various aspects of the CM-2 and DAP systems described in this paper. We are also grateful to the two anonymous referees who gave us valuable comments on an earlier draft of this paper, improving it considerably.

REFERENCES

1. RASMUSSEN, E. Parallel Information Processing. *In*: WILLIAMS, M.E., *Annual Review of Information Science and Technology (ARIST),* Volume 27, N.J.: American Society for Information Science, 1992, 99-130.
2. FLYNN, M. J. Some Computer Organisations and Their Effectiveness. *IEEE Transactions on Computers*, *c-21* (9), 1972, 948-960.
3. HOCKNEY, R. W. *and* JESSHOPE, C.R. *Parallel computing 2*. Bristol: IOP Publishing, 1988.
4. DEITEL, H. M. *Operating Systems*. 2nd Edition, Massachusetts: Addison-Wesley, 1990.
5. TANENBAUM, A.S. *Structured Computer Organisation*. 3rd Edition, N.J.: Prentice-Hall, 1990.
6. HWANG, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. Singapore: McGraw-Hill, 1993.
7. BALE, A.G., LITT, J. *and* PAVELIN, J. The AMT DAP 500 System. *In*: FOUNTAIN, T.J. *and* SHUTE, M.J., eds. *Multiprocessor Computer Architectures*. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 1990, 155-184.
8. HOLLAAR, L.A. Special-Purpose Hardware for Text Searching: Past Experience, Future Potential. *Information Processing & Management*, *27* (4), 1991, 371-378.
9. HOLLAAR, L.A. Special-Purpose Hardware for Information Retrieval. *In*: FRAKES, W.B, *and* BAEZA-YATES, R., eds. *Information Retrieval, Data Structures and Algorithms*. N.J.: Prentice-Hall, 1992, 443-458.
10. HURSON, A.R., MILLER, L.L., PAKZAD, S.H. *and* CHENG, J.B. Specialized Parallel Architectures for Textual Databases. *In*: YOVITS, M., ed. *Advances In Computers*. *Vol. 30*, Academic Press, 1990. 1-37.
11. REDDAWAY, S.F. High Speed Text Retrieval From Large Databases On a Massively Parallel Processor. *Information Processing & Management*, *27* (4), 1991, 311-316.
12. POGUE, C.A. *and* WILLETT, P. *Text Searching Algorithms for Parallel Processors*. British Library Research Paper 11, London: British Library, 1987.

13. CARROLL, D.M., POGUE, C.A., *and* WILLETT, P. Bibliographic Pattern Matching Using the ICL Distributed Array Processor. *Journal of the American Society for Information Science*, *39*(6), 1988, 390-399.
14. POGUE, C.A., *and* WILLETT, P. Use of text signatures for document retrieval in a highly parallel environment. *Parallel Computing*, 4, 1987, 259-268.
15. POGUE, C.A., RASMUSSEN, E.M., *and* WILLETT, P. Searching and clustering of databases using the ICL Distributed Array Processor. *Parallel Computing*, 8, 1988, 399-407.
16. HAWKING, D. *The Design And Implementation Of A Parallel Document Retrieval Engine*. Technical Report TR-CS-95-08, Department of Computer Science. Canberra: Australian National University, 1995.
17. HAWKING, D. PADRE - A Parallel Document Retrieval Engine. *In*: ISHII, M., ed. *Proceedings of the 3rd Parallel Computing Workshop, Kawasaki, Japan, November 1994*. Kawasaki: Fujitsu Parallel Computing, Research Facility, 1994, P2-C.
18. HAWKING, D. *and* THISTLEWAITE, P. Searching For Meaning With The Help of A PADRE. *In*: HARMAN, D.K., ed. *Proceedings of Third Text Retrieval Conference, Gaithersburg, USA, November 1994*. Gaithersburg: NIST, 1995, 257-268
19. HAWKING, D., BAILEY, P., CAMPBELL, D., THISTLEWAITE, P. *and* TRIDGELL, A. A PADRE in MUFTI (A Multi User Free Text retrieval Intermediary). *In*: DARLINGTON, J., ed. *Proceedings of the 4th Parallel Computing Workshop, Imperial College, London, September 1995*. London: Imperial College / Fujitsu Parallel Computing Research Facility, 1995, 75-84.
20. HAWKING, D. *and* THISTLEWAITE, P. Proximity Operators - So Near And Yet So Far. *In*: HARMAN, D.K., ed. *Proceedings of Fourth Text Retrieval Conference, Gaithersburg, USA, November 1995*. Gaithersburg: NIST, 1996, (To Appear).
21. HAWKING, D. *and* BAILEY, P. *PADRE User Manual*, Department of Computer Science, Canberra: Australian National University, 1995.

22. BAILEY, P. *and* HAWKING, D. *A Parallel Architecture for Query Processing Over a Terabyte of Text*. Technical Report TR-CS-96-04, Department of Computer Science, Canberra: Australian National University, 1996.
23. HAWKING, D. High Speed Search of Large Text Base on the Fujitsu Cellular Array Processor. *In*: GUPTA, G., *and* PRITCHARD, P., eds. *Proceedings of 4th Australian Supercomputer Conference, Bond University, December 1991*. Gold Coast: Bond University, 1991, 83-90.
24. HAWKING, D. PADDY's Progress (Further Experiments In Free-Text Retrieval On The AP1000). *In*: ISHII, M., ed. *Proceedings of the 1st Parallel Computing Workshop, Kawasaki, Japan, November 1992*. Kawasaki: Fujitsu Parallel Computing, Research Facility, 1992, ANU-8.
25. HAWKING, D. *and* BAILEY, P. Towards a Practical Information Retrieval System for the Fujitsu AP1000. *In*: ISHII, M., ed. *Proceedings of the 2nd Parallel Computing Workshop, Kawasaki, Japan, November 1993*. Kawasaki: Fujitsu Parallel Computing, Research Facility, 1993, P1-S.
26. DEWITT, D., *and* GRAY. J. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, *35* (6), 1992, 85-98.
27. SUNDERAM, V.S. PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience*. *2* (4), 1990, 315-339.
28. DONGARRA, J.J., OTTO, S.W., SNIR, M., *and* WALKER, D. A message passing standard for MPP and Workstations. *Communications of the ACM*, *39* (7), 1996, 84-90.
29. YOUNT, R.J., VRIES, J.K., *and* COUNCILL, C.D. The Medical Archival System: an information retrieval system base on distributed parallel processing. *Information Processing & Management*, *27* (4), 1991, 379-389.
30. TOMASIC, A., *and* GARCIA-MOLINA, H. *Performance of Inverted Indices in Shared-Nothing Distributed Text Document Information Retrieval Systems*. Technical Report STAN-CS-92-1434, Department of Computer Science, C.A.: Stanford University, 1992.

31. TOMASIC, A., and GARCIA-MOLINA, H. Caching and Database Scaling in Distributed Shared-Nothing Information Retrieval Systems. *In*: BUNEMAN, P., *and* JAJODIA, S., eds, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. N.Y.: ACM Press, 1993, 129-138.

32. ROBERTSON, S.E., WALKER, S., JONES, S., HANCOCK-BEAULIEU, M.M. *and* GATFORD, M. Okapi at TREC-3. *In*: HARMAN, D.K., ed. *Proceedings of Third Text Retrieval Conference, Gaithersburg, USA, November 1994*. Gaithersburg: NIST, 1995, 109-126.

33. BELL, G. Ultracomputers: A Teraflop before its time, *Communications of the ACM*, *35* (8), 1992, 27-47.

34. HOCKNEY, R.W. Performance parameters and benchmarking of supercomputers. *In*: DONGARRA, J.J., *and* GENTZSCH, W., *Computer Benchmarks: Advance in Parallel Computers 8*. Amsterdam: North-Holland, 1993, 41-63.

35 FRAKES, W.B. Introduction to Information Storage and Retrieval Systems. *In*: FRAKES, W.B, *and* BAEZA-YATES, R., eds. *Information Retrieval, Data Structures and Algorithms*. N.J.: Prentice-Hall, 1992, 1-12.

36. STONE, H.S. Parallel querying of large database: A case study. *IEEE Computer*, 20 (10), 1987, 11-21.

37. SALTON, G., and BUCKLEY, C. Parallel text search methods. *Communications of the ACM*, *31* (2), 1988, 202-215.

38. STANFILL, C. *and* KAHLE, B. Parallel Free-Text Search on the Connection Machine System. *Communications of the ACM*, *29* (12), 1986, 1229-1239.

39. STANFILL, C., THAU, R., *and* WALTZ, D. A parallel Indexed algorithm for Information Retrieval. *In*: BELKIN, N.J., *and* VAN RIJSBERGEN, C.J., eds. *Proceedings of the 12th annual conference on research and development in Information Retrieval, SIGIR'89,* New York: ACM Press, 1989, 88-97.

40. RASMUSSEN, E.M. Introduction: parallel processing and information retrieval. *Information Processing & Management*, *27* (4), 1991, 225-263.

41. FOX, E., BETRABET, S., KOUSHIK, M., *and* LEE, W. Extended boolean models. *In*: FRAKES, W.B, *and* BAEZA-YATES, R., eds. *Information Retrieval, Data Structures and Algorithms*. N.J.: Prentice-Hall, 1992, 393-418.

42. SKILLICORN, D.B. *A generalisation of indexing for parallel document search*. External Technical Report, Ontario: Queen's University, Canada, 1995.
43. MACLEOD, K.J. *and* ROBERTSON W. A neural algorithm for document clustering. *Information Processing & Management*, *27* (4), 1991, 337-346.
44. BLAIR, B.C. *and* MARON, M.E. An evaluation of retrieval effectiveness for a full-text document retrieval system. *Communications of the ACM*, *28* (3), 1985, 289-299.
45. SALTON, G. Another look at automatic text-retrieval systems. *Communications of the ACM*, *29* (7), 1986, 648-656.
46. BLAIR, B.C. and MARON, M.E. Full-text information retrieval: further analysis and clarification. *Information Processing & Management*, *26* (3), 1990, 437-447.
47. FALOUTSOS, C. Access methods for text. *ACM Computing Surveys*, *17* (1), 1985, 49-74.
48. WIRTH, N. *Algorithms & Data Structures*. N.J.: Prentice-Hall, 1986.
49. WOLFF, J.G. A scaleable technique for best-match retrieval of sequential information using metrics-guided search. *Journal of Information Science*, *20* (1), 1994, 16-28.
50. SKILLICORN, D.B. *Structured parallel computation in structured documents*. External Technical Report, Ontario: Queen's University, Canada, 1995.
51. STEPHEN, G.A., *and* MATHER, P. What is SP? *The Computer Journal*, *37* (9), 1994, 745-752.
52. WOLFF, J.G. What is SP?: A Reply, Correspondence. *The Computer Journal*, 38 (3), 1994, 253-255.
53. STEPHEN, G.A. What is SP?: A Reply, Correspondence. *The Computer Journal*, 38 (3), 1994, 255-256.
54. WALTZ, D. Applications of the Connection Machine. *IEEE Computer*, *20* (1), 1987, 85-97.

55. PANAGOPOULOS, G. *and* FALOUTSOS, C. Bit-Sliced Signature Files for very large text databases on a parallel machine architecture. *In*: MATTHIAS, J., BUBENKO, J., *and* JEFFERY, K., eds. *Proceedings of EDBT'94*. Heidelberg: Springer-Verlag, 1994. 379-392.

56. GRANDI, F., TIBERIO, P. *and* ZEZULA, P. Frame-Sliced Partitioned Parallel Signature Files. *In*: BELKIN, N.J., INGWERSEN, P., *and* PEJTERSEN, A.M., eds. *Proceedings of the 15th annual conference on research and development in Information Retrieval*, *SIGIR'92*. New York: ACM Press, 1992, 286- 297.

57. WALDEN, M., *and* SERE, K. Free text retrieval on transputer networks. *Microprocessors and Microsystems*, 13(3), 1989, 179-187.

58. CRINGEAN, J.K, MANSON, G.A., WILLETT, P., *and* WILSON, G.A. Efficiency of text scanning in bibliographic databases using microprocessor-based multiprocessor networks. *Journal of Information Science*, 14(6), 1988, 335-345.

59. CRINGEAN, J.K, LYNCH, M.F., MANSON, G.A., WILLETT, P., *and* WILSON, G.A. Parallel Processing techniques for Information Retrieval. Searching of textual and chemical databases using transputer networks. In: *Online Information 89*, Oxford: Learned Information, 1989, 447-452.

60. CRINGEAN, J.K, ENGLAND, R. MANSON, G.A. *and* WILLETT, P. Parallel Text Searching In Serial Files Using a Processor Farm. *In*: VIDICK, J.L, ed, *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*. New York: ACM, 1990, 429-453.

61. CRINGEAN, J.K, ENGLAND, R. MANSON, G.A. *and* WILLETT, P. Network Design for the Implementation of Text Searching using a Multicomputer. *Information Processing & Management*, *27* (4), 1991, 265-283.

62. CRINGEAN, J.K, ENGLAND, R. MANSON, G.A. *and* WILLETT, P. Nearest-neighbour searching in files of text signatures using transputer networks. *Electronic Publishing*, *4*(4), 1991, 185-203.

63. STANFILL, C, *and* THAU, R. Information Retrieval on the Connection Machine: 1 to 8192 Gigabytes. *Information Processing & Management*, *27* (4), 1991, 285-310.

64. STANFILL, C. Parallel Information Retrieval Algorithms. *In*: FRAKES, W.B, *and* BAEZA-YATES, R., eds. *Information Retrieval, Data Structures and Algorithms*. N.J.: Prentice-Hall, 1992, 413-428.

65. STANFILL, C. Partitioned Posting Files: A Parallel Inverted File Structure for Information Retrieval. *In*: VIDICK, J.L, ed, *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*. New York: ACM, 1990, 413-428.

66. MASSAND, B., *and* STANFILL, C. An Information Retrieval Test-bed on the CM-5. *In*: HARMAN, D.K., ed. *Proceedings of Second Text Retrieval Conference, Gaithersburg, USA, November 1993*. Gaithersburg: NIST, 1994, 117-122.

67. LINOFF, G., and STANFILL, C. Compression of Indexes with Full Positional Information in Very Large Text Databases. *In*: KORFHAGE, R, RASMUSSEN, E.M., *and* WILLETT, P., eds, *Proceedings of Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York: ACM, 1993, 88-95.

68. AALBERSBERG, I.J, *and* SIJSTERMANS, F. InfoGuide: A full-text document retrieval system. *In*: TJOA, A.M., *and* WAGNER, R., eds. *Proceedings of the international conference of database and expert systems applications, DEXA'90*. Berlin: Springer-Verlag, 1990, 12-21.

69. JEONG, B., *and* OMIECINSKI, E. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, *6* (2), 1995, 142-153.

70. KAPALEASWARAN, T.N., *and* RAJARAMAN, V. Parallel search methods of a document database in a distributed computer system: a case study. *Journal of Information Science*, *16*, 1990, 291-298.

71. SALTON, G., *and* BERGMARK, D. Parallel Computations in Information Retrieval. *In*: HANDLER, W., ed, *Proceedings of CONPAR'81*, Berlin: Springer-Verlag, 1981, 328-342.

72. RASMUSSEN, E.M., *and* WILLETT, P. Efficiency of Hierarchic Agglomerative Clustering using the ICL Distributed Array Processor. *Journal of Documentation*, *45* (1), 1989, 1-24.

73. FRIEDER, O., *and* SEIGELMANN, H.T. On the Allocation of Documents in Multiprocessor Information Retrieval Systems. *In*: KORFHAGE, R, RASMUSSEN, E.M., *and* WILLETT, P., eds, *Proceedings of Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York: ACM, 1993, 230-239.

74. SEIGELMANN, H.T., *and* FRIEDER, O. Document Allocation in Multiprocessor Information Retrieval Systems. *In*: ADAM, N.R., and BHARGAVA, B.K. eds. *Advanced Database Systems*. Berlin: Springer-Verlag, 1993, 289-310.

75. OZKARAHAN, E. System architectures for information processing. *Information Processing & Management*, *27* (4), 1991, 347-369.

76. SHARMA, R. A Generic Machine for Parallel Information Retrieval. *Information Processing and Management*, *25* (3), 1989, 223-235.

77. KWOK, K.L. A Neural Network for Probabilistic Information Retrieval. *In*: BELKIN, N.J., *and* VAN RIJSBERGEN, C.J., eds. *Proceedings of the 12th annual conference on research and development in Information Retrieval, SIGIR'89*. New York: ACM Press, 1989, 21-30.

78. KWOK, K.L., and GRUNFELD, L. TREC2 Document Retrieval Experiments using PIRCS, *In*: HARMAN, D.K., ed. *Proceedings of the Second Text Retrieval Conference, Gaithersburg, USA, November 1993*, Gaithersburg: NIST, 1994, 233-242.

79. ODDY, R.N. *and* BALAKRISHNAN, B. PTHOMAS: An Adaptive Information Retrieval System on the Connection Machine. *Information Processing & Management*, *27* (4), 1991, 317-335.

80. STEWART, M. *and* WILLETT, P. Nearest Neighbour searching in binary search trees: simulation of a multiprocessor system. *Journal of Documentation*, *43*(2), 1987, 93-111.

81. DEERWESTER, S.C., ZIFF, D.A., *and* WACLENA, K. An architecture for full text retrieval systems. *In*: TJOA, A.M., *and* WAGNER, R., eds. *Proceedings of the international conference of database and expert systems applications, DEXA'90*. Berlin: Springer-Verlag, 1990, 22-29.

82. GROSSMAN, D.A., HOLMES, D.O., *and* FRIEDER, O. A Parallel DBMS Approach to IR in TREC-3. *In*: HARMAN, D.K., ed. *Proceedings of Third Text Retrieval Conference, Gaithersburg, USA, November 1994*. Gaithersburg: NIST, 1995, 279-288.

83. GROSSMAN, D.A., HOLMES, D.O., FRIEDER, O., NGUYEN, M.D. *and* KINGSBURY, C.E. Improving Accuracy and Run-Time Performance for TREC-4. *In*: HARMAN, D.K., ed. *Proceedings of Fourth Text Retrieval Conference, Gaithersburg, USA, November 1995*. Gaithersburg: NIST, 1996, (To Appear).

84. EFRAIMIDIS, P. GLYMIDAKIS, C. MAMALIS, B. SPIRAKIS, P. *and* TAMPAKAS, B. Parallel Text Retrieval on A High Performance Supercomputer Using the Vector Space Model. *In*: FOX, E.A., INGWERSEN, P *and* FIDEL, R. eds. *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, *Special Issue of SIGIR forum*. New York: ACM Press, 1995, 58-66.

85. WILSON, E. *Using hypertext and parallel processing to integrate multi-purpose, multi-structural databases*. Hypertext paper, Kent: University of Kent at Canterbury, 1996

86. WILSON, E. *Hypertext and Parallel Processing: Browsing and Retrieval*. Hypertext paper, Kent: University of Kent at Canterbury, 1996.

87. GONNET, G.H., BAEZA-YATES, R.A., *and* SNIDER, T. New indices for text: PAT trees and PAT arrays. *In*: FRAKES, W.B, *and* BAEZA-YATES, R., eds. *Information Retrieval, Data Structures and Algorithms*. N.J.: Prentice-Hall, 1992, 66-82.

88. SHOENS, K., TOMASIC, A., *and* GARCIA-MOLINA H. Synthetic workload performance analysis of incremental updates. *In*: BRUCE CROFT, W., and VAN RIJSBERGEN, C.J., eds. *Proceedings of the 17th annual international ACM-SIGIR conference on research and development in Information Retrieval. SIGIR94*, London: Springer-Verlag, 1994, 329-338.

89. MACFARLANE, A., ROBERTSON, S.E., *and* MCCANN, J.A. On concurrency control for Inverted files. *In*: JOHNSON, F.C., ed. *Proceedings of the 18th BCS IRSG Annual Colloquium on Information Retrieval Research,*

March 26-27 1996, Manchester. Manchester: BCS IRSG, 1996, 67-79.

90. CLARKE, C.L.A., and CORMACK, G.V. *Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System*. MultiText Project Technical Report MT-95-01, Department of Computer Science, Ontario: University of Waterloo, 1995.
91. HARMAN, D. Relevance feedback and other query modification techniques. *In*: FRAKES, W.B, *and* BAEZA-YATES, R., eds. *Information Retrieval, Data Structures and Algorithms*. N.J.: Prentice-Hall, 1992, 241-263.
92. WILLETT, P., *and* RASMUSSEN, E.M. *Parallel Database Processing*. London: Pitman, 1990.

GLOSSARY

BSSF	Bit Sliced Signature File.
CM-2	Thinking Machines Connection Machine 2.
CPU	Central Processing Unit.
DAP	Distributed Array Processor.
Distributed memory	Architecture in which memory is distributed amongst processors.
DSM	Architecture in which memory is physically distributed, but logically shared amongst processors.
FSA	Finite State Automata.
Gigabytes	2^{30} bytes.
Granularity	Measure or size of individual computation in parallel computing.
IDF	Inverse Document Frequency.
Intra-query	Methods available within queries i.e. parallelism.
Inter-query	Methods available between queries i.e. parallelism.
Inverted File	Index organisation of keywords and the documents they occur in.
I/O	Input / Output.
LI	Load Imbalance.
Megabytes	2^{20} bytes.
MCU	Master Control Unit.
MIMD	Multiple Instruction Multiple Data machine architecture.
MISD	Multiple Instruction Single Data machine architecture.
MMM Model	Fuzzy set based extended boolean model.
NEWS grid	North South East West interconnect for parallel architecture.
PADRE	PArAllel Document Retrieval Engine.
Paice Model	Fuzzy set based extended boolean model.
PE	Processing Element.
P-NORM Model	Distance based extended boolean model.
Precision	Measure of relevant documents retrieved.
Process farm	A set of processes where a farmer process distributes work to worker processes.
Recall	Measure of retrieved relevant documents.
Regular Expressions	Used to search for a number of patterns rather than a single pattern.
signature	Document surrogate of n bits, where terms are hashed to m bits.
SIMD	Single Instruction Multiple Data machine architecture.
SISD	Single Instruction Single Data machine architecture.
Shared everything	Architecture in which memory and disk are shared among processors.

Shared memory	Architecture in which memory is shared amongst processors.
Shared nothing	Architecture in which a processor has its own memory and disk.
SP	Theory of computing as compression, applied to pattern matching.
Streams	A sequence of instructions or data operated on by a CPU.
Surrogate coding	see signature.
VLDC	Variable Length Don't Care pattern match.