Chapter 2

Review Of Parallel Computing in Information Retrieval

2.1 INTRODUCTION

In this chapter we chart the progress of the use of parallel computing in information retrieval published in MacFarlane et al (1997) and updated here, following on from a review of the subject by Rasmussen (1992). We review important work in the past. We describe parallel architectures used for parallel IR systems. We analyse the different approaches to parallel IR using a classification due to Rasmussen (1992). Examples of parallel IR systems or methods are given in a case studies section. The motivation for the use of parallel computing in IR is an important strand in this chapter, in particular when and when not to use parallel systems. The retrieval models used in parallel IR systems are described. We give a summary at the end, and conclude by identifying areas not tackled in previous research to be addressed in the main body of this thesis.

2.2. PARALLEL ARCHITECTURES USED IN IR SYSTEMS

2.2.1 Parallel architecture classification

Flynn (Flynn, 1972) describes a taxonomy for classifying parallel architectures. A number of criticisms have been levelled at the taxonomy, e.g. there is no treatment of input/output and the instruction set used is ignored. In the context of IR, ignoring input/output is a particular problem (see section 2.2.3). 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 Jesshop, 1988). The Flynn taxonomy uses the concept of streams (Flynn, 1972) 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 (Deitel, 1990; Tanenbaum, 1990) while others assert that systolic arrays can be placed in this class (Hwang, 1993). We address the MISD class in our discussion on special parallel hardware below. We will ignore the SISD class for the rest of the chapter.

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/CPP DAP (Bale et al, 1990) and Thinking Machines CM-2 (Hwang, 1993): the DAP is described in more detail below. The architecture had been dominant in the use of parallel computing in information retrieval until recently.

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.

Fig 2-1. Types of memory organisation examples
There is a wide variety of this class of architecture including those where processors share the same memory and others in which processors have their own memory. These are known as shared memory and distributed memory architectures (see examples in fig 2-1). Each has its own subdivision which we will not attempt to describe here.

With the former, interprocessor communication is done through concurrency control mechanisms such as semaphores, 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 2.2.2 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 (Hollaar, 1991) including associative memories, finite state machines and cellular arrays (Hollaar, 1992). Some of this work has been in building special purpose parallel architectures (Hurson et al, 1990) for text retrieval and we include it in the review for completeness.

2.2.2 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 2.2.1; the DAP, Fujitsu 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 (Rasmussen, 1992).

1). DAP (Distributed Array of Processors). The CPP (formally ICL) DAP is a SIMD class architecture. The DAP organisation (Bale et al, 1990) 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 PEs 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 2.2.3 below). The DAP is successfully used by the DapText system described by (Reddaway, 1991) and is included in the case studies section (2.6.1) below. Reuters use this system to provide Text Retrieval services to their...
customers. 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 (Pogue and Willett, 1987a and 1987b; Carroll et al, 1988; Pogue et al, 1988)

II). *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 network (this network looks rather like a flattened donut). Each cell can support from 16 Mbytes to 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 facilities which are useful for term weighting calculations. Work on IR using the AP1000 was pursued at the Australian National University through the PADRE system (Hawking, 1994 and 1995; Hawking and Thistlewaite, 1994 and 1996; Bailey and Hawking, 1996; Hawking et al, 1995), which has evolved through the PADDY (Hawking, 1990 and 1991) and FTR (Hawking and Bailey 1993) systems. These systems are discussed in more detail in the case studies section (2.6) below.

III). *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, 1992) and (Hurson et al, 1990) 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 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. This author does 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 (DeWitt and Gray, 1992).
IV). 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 (Sunderam, 1990) and standards such as MPI (Dongarra et al, 1996). One particular system discussed in this review (MARS) uses networked workstation technology for its hardware platform (Yount et al, 1991). 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. Most have gone bankrupt apart from AMT which has metamorphosed 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 has had significant impact on parallel IR systems (see section 2.6.4 below).

2.2.3 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 can be 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 known as the I/O bottleneck and it is one that is shared with the area of Parallel database systems (DeWitt and Gray, 1992). In consequence, many of the systems mentioned above have very impressive I/O rates to overcome this I/O bottleneck. One architecture that addresses the issue is the shared nothing architecture described by DeWitt and Gray (1992). 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 sub-sets of the whole data (which can be very large in database systems). Index maintenance costs can also be reduced. Tomasic and Garica-Molina (1992;1993a) make a very strong case for the use of shared nothing. We use the shared nothing architecture for experiments to be described in this thesis.

However such is not the whole picture. There are some IR computations which are compute bound and require considerable CPU resources. Examples are very large search spaces for passage retrieval and for query modification after relevance feedback, as used in Okapi at TREC experiments (Robertson et al, 1995). In such cases fast I/O cannot make much difference to the overall efficiency.
2.3 MOTIVATION FOR PARALLEL IR SYSTEMS

On the assumption that we want to do more and/or do it 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 (Bell, 1992), 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.

The performance of IR systems is measured by the retrieval effectiveness and retrieval efficiency they provide (Frakes, 1992). 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 and throughput measures defined in chapter 3. 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 effectiveness are recall and precision (see chapter 3). 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 (1987) and Salton and Buckley (1988) 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 (Stanfill and Kahle, 1986). 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 (1987) compares the performance of an inverted file on a single CM-2 node, while Salton and Buckley (1988) 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 (1988) 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 (1989) 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 (1988) paper. Ultimately Stone has been proved to be correct, since most parallel IR systems use inverted files. The set merge on inverted lists can be computationally very intensive.

Four main reasons for applying parallel computers to IR have been suggested (Rasmussen, 1991): these are to improve response times, search larger databases, use superior algorithms and reduce search cost. We discuss each reason in turn below.

2.3.1 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. We may consider the use of query processing acceleration techniques with a resultant loss in retrieval effectiveness (Hawking, 1998) or parallel computing to improve response times. Parallel computation has the potential to offer faster response times for individual queries without loss in retrieval effectiveness and offers a greater throughput for queries and insertions as more memory resources are available. In extreme circumstances we may consider using both techniques to speed up processing. Response time is also dependant on database size in conjunction with multiple query service.

2.3.2 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 in chapter 3. 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. This author does not believe that parallel computing can be usefully applied at this juncture to small databases with few or single user base (Stone, 1987). The emphasis in this review is very much on large scale text databases.
2.3.3 Superior algorithms

We stated in section 2.3.2 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 offer a superior retrieval effectiveness performance in terms of precision and recall than previously implemented algorithms. For example there are a number of extended boolean models (Fox et al, 1992) 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 (1995b) argues that regular expressions offer more powerful query capabilities than other searches. MacLeod and Robertson (1991) 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 (1985) evaluated a large operational full-text IR system over a six month period, and proposed a hypothesis in which deterioration of recall is a function of increasing database size. They argue that extra human effort at the indexing stage is needed to overcome this recall deterioration. Salton (1986) takes issue with their arguments, and they reply (1990). 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 have been investigated in the Okapi at TREC research (Robertson et al, 1995). Because such a search space is 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 for unoptimized passage retrieval (Robertson et al, 1995), where $n$ is the number of text atoms. The search space for term selection on routing or filtering queries is so large no order value can be stated. Chapters 9 and 10 give results of our experiments on these search spaces using parallel computing methods by searching part of the overall space.

2.3.4 Search cost

Stanfill et al (1989) 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 (1989) show that cost effectiveness asymptotically approaches a level of optimal cost effectiveness. By increasing the size of the database 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 (1991) 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 (the CM-2 storage system), which restricts the amount of text, let alone index information that can be stored. Hardware factors therefore limit the relevance of this cost effectiveness metric.

2.4. APPROACHES TO PARALLEL IR

In this section we describe approaches to parallel information retrieval using a classification due to Rasmussen (1992), influenced by Faloutsos’ sclassification of access methods for text (Faloutsos, 1985). 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 distribution 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 an algorithm. Granularity can either be fine, coarse or mixed grain, meaning small, large or variable computation sizes: a computational unit being a single ‘atom’ 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.

2.4.1 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 (Wirth, 1986) 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 algorithm (Wolff, 1994a), proximity searches and pre-computed patterns (Skillicorn, 1995a and 1995b). 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 scenario is rather simplistic, 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 of how load balancing is affected by the implementation of the algorithm is 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 (1987a) is to broadcast individual characters to processors one by one which match them in that order. As each match is 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 (Hawking, 1992) which provides tools for the use of a regular expression library on each cell (processor) of a Fujitsu AP1000. An example of an algorithm implemented on SIMD systems which support complex patterns is variable length don’t care (VLDC). 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 (Wolff, 1994a) 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 (Wolff, 1994b; Stephen and Mather, 1994; Stephen, 1994). 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 et al (1995) describes a method of parallel proximity searches on the PADRE system. 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 (1995a;1995b) 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 a 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 is caused when some processors complete their computations faster than others, which remain 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 (1994) has defined a measure of load imbalance LI in order to understand the effects of data skew on the pattern matching computation (this metric is defined formally in chapter 3). 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, large documents are placed in neighbouring processors to reduce inter-processor costs while smaller documents are used 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 as the original text is needed.

2.4.2 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 (Faloutsos, 1985). 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 (1987a) describe an alternative method where integer values of the bit positions are
broadcast one by one to the processors. The pioneering work described by Stanfill and Kahle on the Connection Machine has already been briefly mentioned in section 2.3 (Stanfill and Kahle, 1986; Waltz, 1987). Other work includes a parallel Bit-Sliced Signature File (BSSF) method described by Panagopoulos and Faloutsos (1994) and Frame-Sliced Partitioned Parallel Signature Files described by Grandi et al (1992). Detailed descriptions of these different methods are given below.

Before we describe systems which use signatures it would be useful to review signature files (see fig 2-2). Signature file can be viewed as matrices where the rows represent document signatures and the columns represent individual bit positions of the signature across documents. 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 (fig 2-2a). The second, vertical partitioning, represents column parallelism where sections of the signatures are compared rather than the whole (fig 2-2b). Vertical partitioning can be done across the collection or by a frame: a subset of the collection (fig 2-2c). 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 (1986) uses the horizontal method (fig 2-2a) 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 Gigabyte 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 (compare these results with ours in chapter 7). This method of search has also been used in Transputer machines (Walden and Sere, 1989).

Panagopoulos and Faloutsos (1994) 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 (fig 2-2b) 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 (1992) describes a hybrid method that combines both horizontal and vertical partitioning methods which they assert are suitable for implementation on parallel machines. The use of the shared nothing architecture described by DeWitt and Gray (1992) is recommended. 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 (fig 2-2c). 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. Comparative results with the systems in this class are not available.

From the above discussion we can assert that the signature partitioning method interacts with 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 (Faloutsos, 1985). 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 (Salton and Buckley, 1988), 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.
2.4.3 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 at Sheffield University (Pogue and Willett, 1987a and 1987b; Carroll et al., 1988) and on a MIMD transputer network by Cringean et al. (1988; 1989; 1990; 1991a; 1991b). Panagopoulos and Faloutsos (1994) 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. (1988; 1989; 1990; 1991a; 1991b). 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 grid 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. (1991b) 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 were 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.

2.4.4 Inverted file

Most commercial and academic IR systems use inverted files. The reason for this is that until recently 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 2.4.2 the storage requirements for inverted files are far costlier than any of the other methods reviewed in this chapter. In our description of the method below, we pay particular attention to the data partitioning schemes introduced in chapter 1.

The most prominent of parallel IR systems have used inversion as their storage technique (Reddaway, 1991; Stanfill et al, 1989; Aalsbersberg and Sijsternans, 1990; Stanfill and Thau, 1991; Stanfill, 1992; Linoff and Stanfill, 1993; Massand and Stanfill, 1994; Hawking, 1994). We briefly review the structure of an inverted file (Faloutsos, 1985): 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.

Jeong and Omiecinski (1995) 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 described in section 2.2.3. Multiple disks are used to exploit I/O parallelism. The use of a multiprocessor with shared memory is assumed. The two partitioning methods described in chapter 1 above are considered. The results produced by simulations are that TermId partitioning is best when the term distribution in the query is less skewed (or more uniform) and DocId partitioning is best when term distribution is more skewed (or less uniform). DocId partitioning sacrifices more I/O and space in order to ensure better load balancing in a more skewed query environment. 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. DocId 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 DocId partitioning (and the term accesses can be done in parallel). Inter-query parallelism is more difficult to service with DocId partitioning; each query term 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 DocId partitioning (see section 2.5.3). Based on their simulations, Jeong and Omiecinski recommend that the Shared Everything architecture be used in medium sized Text Retrieval systems or as components in a larger shared nothing machine.

Tomasic and Garcia-Molina (1992;1993a) 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 DocId and TermId 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 DocId partitioning, but the inversion is spread across the disks connected to the CPU. Where one I/O bus exists 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 TermId partitioning) performed better on abstracts.

We can divide parallel systems which have implemented inverted files into two main camps, those which use TermId partitioning (Reddaway, 1991; Stanfill et al 1989; Ribeiro-Neto et al, 1999) and those which use DocId partitioning (Hawking, 1996; Aalbersberg & Sijstermans, 1990; Stanfill & Thau 1991; Hollaar, 1991). None of these groups have done comparative studies to ascertain which of the partitioning methods is best. Most of the research done on parallelism for inverted files has been done on the search task. Some work in the area of selective dissemination of information (SDI) has been done by Kapaleaswaran and Rajaraman (1990) using a subject division rather than data partitioning method.

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 (Reddaway, 1991) which would overcome this limitation. Three systems which use inverted files are described in the case studies section (2.6) in more detail. More recent work at TREC can also be found in the case studies section.

2.4.5 Clustering

Clustering is a method of identifying similar documents, based on a given similarity method, e.g. distance in vector space using the cosine function. The documents are organised into groups or clusters, which in turn can consist of a single centroid and document vectors belonging to that cluster (Salton and Bergmark, 1981). There is 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. Fig 2-3 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 (1991) describe a neural network 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 each document vector is compared with the current set of clusters, iterating until a suitable cluster has been found or learned.

Rasmussen and Willett (1989) describe parallel computing for various hierarchic agglomerative clustering methods. Agglomerative clustering consists of building the tree bottom up. 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. Three algorithms are used for clustering; SLINK, Prim-Dijkstra and Ward. The SLINK algorithm only has parallelism in the calculation of the current row of the distance matrix. The Prim-Dijkstra algorithm is almost entirely parallel except for storage of link information. The Ward algorithm uses parallelism the on nearest neighbour method, i.e. identify a chain of related objects concurrently. The parallel SLINK algorithm performed less efficiently than its sequential counterpart, considerable slowdown figures being recorded. The parallel Prim-Dijkstra performed much better in relation to its sequential counterpart with speedups of 3.6 to 6.0 recorded on 4096 1-bit processing elements. 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 (1993) 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 (1991) discusses search on non-hierarchic document clusters on the RAP.3 system. 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 (1989) describes a generic model 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 retrieval efficiency and one of increasing retrieval effectiveness. In the efficiency algorithm closely related clusters are assigned to different sub-cubes such that the number 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 has 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 (centroids) 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, response times in the efficiency scheme remain static, while in the effectiveness scheme time increases 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.
2.4.6 Connectionist approaches

These approaches use a network model to represent information in an IR system (Rasmussen, 1992). 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 (Kwok, 1989; Kwok and Grunfield, 1994). The MacLeod and Robertson algorithm (1991) described in section 2.4.5 which uses neural networks 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 (1991) 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.

2.4.7 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. 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.

I). Vector Processing. Stewart and Willett (1987) 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 was collection dependant.

Efraimidis et al (1995) 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. 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 (1987) 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. A sequential inverted file system may outperform their parallel vector processing system.

Some vector processing models use reduced dimensionality methods which focus on semantic structure of documents in order to increase retrieval effectiveness. In such cases a single document vector is used to represent documents as the methods do not keep keywords in their representation (hence inverted files cannot be used). These methods are highly parallel as processing of the query vector over document vectors is completely independent. Such a method is Latent Semantic Indexing (LSI) first proposed by Deerwester et al (1990). Letsche and Berry (1997) describe a system called LSI++ which uses a back end processor method to hide parallelism from the programmer. The backend system uses a master/slave process topology broadcasting the query vector to the processors and collecting results using a global sort. A collection of 24 SPARC5 machines were used, connected by fast Ethernet. Good speedup was demonstrated on a 100k record USENET collection reducing times from serial
processing by nearly 180 times. A variation of LSI is DSIR (Rungsawang et al, 1999) which uses word co-occurrence to find document vectors which are close to queries vectors in a ‘semantic space’. They use the same master/slave topology as Letsche and Berry (1997) using a network of Pentium processors implemented on the PVM system. Their results show problems with NFS bottlenecks when distributing document vectors and load imbalance from poorly distributed computations.

II). Hybrid Inversion. Yount et al (1991) 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.

III). Functional Programming. Deerwester et al (1990) 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. The implications for space complexity are also significant, 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.

IV). Relational Databases. A great deal of research has been done on using parallel computing for relational databases (DeWitt and Gray, 1992). Experiments using parallel relational databases for have been reported at TREC-3 (Grossman et al, 1995) and TREC-4 (Grossman et al, 1996). The guiding principle of this work is that while parallel relational databases are common, parallel IR systems are rare. An inverted file structure is modelled 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 do 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. The I/O
performance reduction occurs because the amount of space needed to store the index is
dramatically increased. The increase is due to the way data has to be normalised in order to be
placed in relations. Whereas inverted files may need only 1 disk access to read data for a term,
relational databases will need to read in more data and also require more disk accesses.

2.4.8 Summary of parallel IR approaches

We have examined a wide variety of approaches which have had a varying degree of
success in speeding up search using parallelism. For search it is difficult to see how, of all the
techniques studied, the inverted file method can be surpassed with respect to response time for
users. We therefore intend to use that technique in research described in this thesis. Much of
the research concentrates on the search task with some discussion on the indexing and
routing/filtering tasks. We believe it would be productive to look at a wider range of tasks and
examine the issue of data distribution methods and performance. It is particularly useful to look
at the index update task to see if parallelism can provide improvement in an area where
inverted file technology has been weak.

2.5 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.

2.5.1 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 (Stanfill and Kahle, 1986) using the signature
method, the DAP (Reddaway, 1991) and POOMA (Aalbersberg and Sijstermans, 1990)
machines using the inverted file method and PADRE (Hawking and Bailey, 1995) 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 (Yount et al, 1991) 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. Naïve 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.

2.5.2 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 (Reddaway, 1991), pattern matching in PADRE (Hawking and Bailey, 1995) and MARS (Yount et al, 1991). 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 (znear) (Hawking and Thistlewaite, 1996). 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.

2.5.3 Term weighting models

One of the main methods used to improve retrieval effectiveness is to utilise one of the myriad term weighting schemes that are available. The dominant scheme had been the vector processing model with systems such as RAP.3 (Ozkarahan, 1991), DowQuest (Stanfill and Thau, 1991), Transputer Networks (Cringean et al, 1988; Efraimidis et al, 1995) and POOMA (Aalbersberg and Sijstermans, 1990), all using it in various forms. More recently variations of BM25 term weighting model have been used (Hawking et al, 1999), the technique we use throughout this thesis. PADRE (Hawking and Thistlewaite, 1996) offers a number of weighting schemes based on the inverse document frequency (IDF) measure. These models may use
unnormalised term weighting (Hawking and Thistlewaite, 1996) or normalised (Hawking, 1992; Stanfill and Thau, 1991; Aalbersberg and Sijstermans, 1990; Efraimidis et al, 1995). Cringean et al (1988) do not specify the normalisation method. Others such as Reddaway (1991) and Jeong and Omiecinski (1995) do not specify a weighting scheme in their discussion of term scoring in their papers. A very important issue has 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 could affect the efficiency of the term weighting calculation. Many parallel machines provide facilities to do just this, e.g. the DAP (Bale et al, 1990) and Fujitsu AP1000 (Hawking, 1995) in the form of global network operations. Where this special hardware does not exist, the interprocessor communication may reduce efficiency. 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.

2.5.4 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 and Willett (1987), Hawking et al (1995) and Skillicorn (1995b). 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.

2.6 CASE STUDIES - "STATE OF THE ART"

We present four case studies 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 or methods being used in research laboratories. Inverted file technology is used by all of the systems discussed in the case studies. 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.
2.6.1 **DAPText**

DAPText (Reddaway, 1991) is a commercially used parallel text retrieval system 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 collection frequencies have postings represented in 8 bits and 16 bits, whilst 24 bits are used to represent rare terms. The higher the collection frequency 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 computation is 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. More recent work on using hypertext and the DapText system is reported by Wilson (1996a;1996b).

2.6.2 **DowQuest**

The DowQuest system is also a commercially successful system. The Dow Jones News Retrieval Service uses the system for its Text Retrieval needs (Yount et al, 1991). The algorithms and data structures for the system are described by Stanfill & Thau (1991) and Stanfill (1990;1992). We outline some related work done by Thinking Machines which is described in Stanfill et al (1989) and Stanfill (1992): 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.
The algorithm described in Stanfill et al (1989) works by multiplying a query weight with stored postings weights in parallel and sending the result to a mailbox somewhere on the machine. The TermId 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 Fig 2-4). 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 (1991) 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 (Stanfill and Thau, 1991).

The partitioned posting file method described in Stanfill & Thau (1991), 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 DocId 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 (Massand and Stanfill, 1994; Linoff and Stanfill, 1993) was written for the Connection Machine CM-5: a more powerful machine with a hybrid SIMD/MIMD architecture.

Massand and Stanfill (1994) and Linoff and Stanfill (1993) 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: compression is applied to position data, but not to weighting data. They claim the decreased time in I/O can fully compensate for the decompress computation (based on a study of two corpora; 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 DocId 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: the first pass calculates the space needed for each inverted list and the second pass indexes the text and puts inverted data into pre-allocated blocks. This algorithm took 20 minutes in comparison with the 90 or so seconds on the Fujitsu AP1000 reported by Hawking running the PADRE system (Hawking, 1995). 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.

2.6.3 PADRE

More information is available on PADRE and its precursors than any other system covered in this chapter, and the system continues to be used for research purposes. We have already imparted much information on the system ranging from the hardware it uses (section 2.2.2), methods of operation (section 2.4.1 and 2.4.4) and query models available for the system (section 2.5). We therefore restrict our discussion to the history and philosophy of PADRE.
The system started life as PADDY (Hawking, 1991 and 1992) and concentrated on linguistic and lexicographic research on the *Concise Oxford English Dictionary* structured in SGML. Searches are based on the PAT indexing method (Gonnet et al, 1992), 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 (Hawking, 1991). 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 (1992) 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 (Hawking, 1993) 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 (Hawking, 1995), term weighting (Hawking, 1994), natural language processing techniques (Hawking and Thistlewaite, 1995), multiple user facilities (Hawking et al, 1995) and proximity spans (z-mode) (Hawking and Thistlewaite, 1996). The *DocId* 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 (Hawking et al, 1995) is recommended to ensure reasonable responses times for users and to avoid ‘killer queries’ which
can greatly reduce system throughput. More recently PADRE has moved to the cluster computing model (see below).

2.6.4 Cluster Computing

It is clear from recent research that standard components as part of networks of workstations is now dominant in the field of parallelism in IR. Cluster computing has revived the field of parallelism for IR after a three or four year moratorium. Cluster computing has been used within the framework of NIST's Text Retrieval Conference (TREC) series to examine performance over very large databases: initially over a 20 Gigabytes collection (Hawking and Thistlewaite, 1998) and then a very large 100 Gigabyte web collection (Hawking et al, 1999; Hawking et al, 2000). Participants have used a variety of architectures such as DEC Alpha, SGI, Intel and Sun, while using processor sets ranging from just 2 up to 20. Most of these systems split the collection and place the index on local disk using the DocId partitioning method. The granularity of parallelism used in cluster computing is very coarse grain. One group (ANU) tried using a RAID disk with subsequent performance degradation (Hawking et al, 1998). Some use the shared nothing architecture while others used a shared memory machine configuration. It is difficult to make any comment on which of these architectures are best as participants in these TREC experiments use very different methods from each other. We give our results in the Web Track experiments throughout the thesis (MacFarlane et al, 2000a).

We also contacted various web search engines to find out what type of parallelism they use. Understandably they were very reticent about giving out information on the methods and architectures they use and few responded to my enquires. The most helpful was Dixon (2000) who informed us that Google uses 4,000 Intel Pentium PIII boxes in three server farms which in their words are used to "index, categorise and prioritise the Web and return results to searchers". These Linux servers are stripped down and customised for Google's requirements. Altavista used to use 16 nodes each with 10 TurboLazer Alpha processors together with 200 Gigabytes of disk space and 8 Gigabytes of in-core memory. They have recently moved to using Compaq equipment, but were not willing to describe either the configuration or the architecture they now use (Shissler, 2000). Northern Light (Kim, 2000) were only able to give brief details on the ranking, NLP and clustering methods they use, which in itself was interesting but not very useful for our purposes.
2.7 SUMMARY AND CONCLUSION

This chapter gives an overview of the application of parallel computing to IR 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 (Rasmussen, 1992) 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. 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 parallel IR systems plus recent work in Cluster computing for IR. For further information on many of the systems described in this chapter, the reader is referred to Rasmussen (1991) a special issue on parallel processing in IR as well as Willett and Rasmussen (1990) for a large body of work done on the DAP.

Much of the work described above focuses on searching of text using various parallel methods, but there has been little focus on indexing text, using passage retrieval techniques, index maintenance or routing/filtering. In particular there is no overall survey of data distribution techniques for inverted files for all of the tasks being considered. The purpose of this thesis is provide this overall survey and to find out which data distribution method is best overall and for a particular task. Many of the algorithms described above are specifically written for one particular parallel machine and would be difficult to port. We aim to provide a system which is portable across machines.