Parallel Text Retrieval On A High Performance Supercomputer Using The Vector Space Model

Efraimidis P., Glymidakis C., Mamalis B.
Spirakis P. and Tampakas B.

Computer Technology Institute
Department of Computer Engineering and
Information, University of Patras
Box 1122, GR-26110, Patras, Greece

Abstract

This paper[1] discusses the efficiency of a parallel text retrieval system that is based on the Vector Space Model. Specifically, we describe a general parallel retrieval algorithm for use with this model, the application of the algorithm in the FIRE system [1], and its implementation on the high perfomance GCel3/512 Parsytec parallel machine [2]. The use of this machine's two-dimensional grid of processors provides an efficient basis for the virtual tree that lies at the heart of our retrieval algorithm. Analytical and experimental evidence is presented to demonstrate the efficiency of the algorithm.

1. Introduction

Parallel processing has the potential to provide fast, cost-effective access and management of large amount of data. The rapid growth in the power of parallel machines over the last few years had led to a proliferation of interest in the use of parallel processing techniques, where some or many processors operate together so as to reduce the elapsed time required for a computational task. Consequently, many attempts have been made to apply such techniques to the text retrieval systems. These attempts vary concerning the text processing model they use (signatures, inverted file structures) and the type of parallel environment they adopt.

With regard to the text processing model, initial parallel implementations were based on overlap-encoded signatures ([3],[4],[5],[6]). Other attempts have been made using frame-sliced partitioning on signature files ([7]). However, the various limitations of these methods (i.e. constrained interactive access for large signature files - [10]; only binary document weights supported - [5]) led to studies and implementations of parallel inverted file structures ([8],[9]). These attempts using the inverted file model, showed considerable performance, even under the above limitations and the limited power parallel environment they used.

Concerning the type of parallelism, we can distinguish between the SIMD and MIMD environments (due to Flyn, [11]). Most parallel text retrieval systems ([3],[8],[9]), have been implemented on the Connection Machine, which is a classical type of SIMD architecture using pipelined vector processors. However the last few years have seen increasing interest in retrieval systems for use in MIMD environments, mainly using microprocessor networks where each processor had its own local memory ([6],[12]). High performance transputer networks with message passing communication mechanisms have been used and they have led to very efficient implementations ([12]).

Our work, following the latter direction is aimed at building an efficient multiprocessor system on a transputer network, which will meet all the requirements that such a system must satisfy, including :

1. Low total communication times.
2. High retrieval effectiveness (in terms of recall & precision).
3. Interactive response times within a friendly user interface environment.
4. Significant response-time improvements compared to serial machines.

In order to satisfy the above requirements the design and implementation of our system makes use of the advantages offered by our integrated parallel machine (the GCel3/512 Parsytec machine, [2]). Considerable efficiency (low total communication times) is achieved via the use of virtual tree topologies over the physical network of 512 high performance transputers. Our work demonstrates how tree–topologies are appropriate for the document scoring and ranking task, when multiple processors are available. Additionally, the retrieval task is based clearly on the Vector Space Model (use of FIRE system, [1]) as opposed to the signature and inverted file parallel implementations that have been studied previously. The high capacity of the local memory of each transputer allows convenient management of the demands of

[1] This research was partially funded by the EC Project No. 9072 (GEPPCOM).

SIGIR'95 Seattle WA USA© 1995 ACM 0-89791-714-6/95/07.$3.50

the Vector Space Model, which makes the parallel algorithm quite simple.

In the following two sections the FIRE system and the GCel3/512 machine (the two basic tools of our work) are briefly presented. In section 4, we demonstrate a simple algorithm that can be used for parallelizing almost any information retrieval system based on the Vector Space Model. This algorithm is partially used in the design and implementation of our system. In section 5, the virtual tree topologies that are used over the two-dimensional grid interconnection network of the GCel3/512 machine are presented. The behaviour of such topologies with the parallel retrieval algorithm, is discussed and analytically evaluated in this section. Finally, in section 6, the performance of searching obtained from our parallel implementation is discussed. Various evaluation results are demonstrated in this section, in order to validate the efficiency provided by using specific tree topologies for text searching in a high performance parallel machine.

2. The FIRE System

FIRE ([1]) is an information retrieval system whose design and implementation is based on the Vector Space Model ([13]) plus the use of an automatically constructed thesaurus. Thus, each document D_i is represented by a unique term vector expanded by the corresponding thesaurus classes:

$$D_i = (d1_{i1}, d1_{i2}, ..., d1_{in}, d2_{i1}, d2_{i2}, ..., d2_{im})$$

where $d1$ stands for the weighted terms of D_i and $d2$ stands for the weighted thesaurus classes of D_i. The thesaurus classes are constructed via a specific method (based on a connected components evaluation algorithm, [1]), that is very close to the relevant method stated in [14].

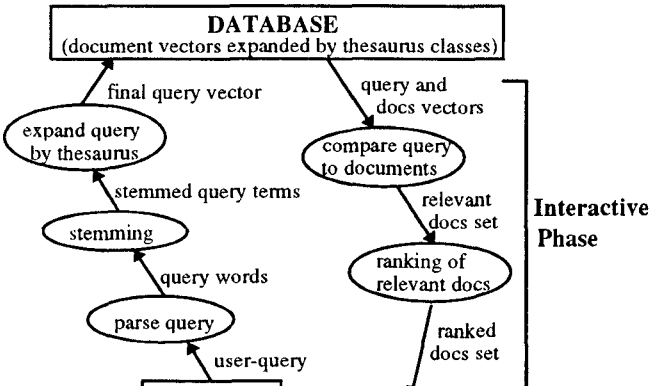

Figure 1 : The operational model of FIRE system

The user queries are represented in the same way leading to one vector for each query. Thus, the retrieval task of the FIRE system (fig. 1) involves comparing the user query vector to all document vectors. This comparison is performed by computing a similarity measure based on the cosine function ([15]). Then the documents' scores are ranked and the most relevant documents to the query, are presented to the user.

Additionally, FIRE is supported by a well-designed X-Windows user interface. This interface also supports the parallel implementation of FIRE leading to a quite friendly multiprocessor information retrieval system. Finally, FIRE provides significant retrieval effectiveness (in terms of recall & precision, see [1]) which is fully preserved by the multiprocessor version that is presented in this paper.

3. The GCel3/512 Supercomputer

The hardware that we use in our implementation is the Parsytec GCel3/512 machine which belongs to the MIMD class of parallel computers. It is a massively parallel machine consisted of 512 processor–units formed in a two dimensional grid interconnection network with dimensions 32 x 16. More precisely each processor–unit is an Inmos T805 transputer, a member of the well–known transputer family.

3.1 The Transputer

A transputer is a high performance processor, especially developed for use in multiprocessor systems. Each transputer consists of a processor, a cache memory and four links which support the transputers' interconnections on the grid network. These features are integrated on a single chip. Members of the family of transputers are the T400,..,T805,.. The T805 that is used in the GCel3/512 machine has the following characteristics:

- 32 bit RISC processor with frequency 30 MHz and peak performance 4.3 MFLOPS, 30 MIPS
- 4 KBytes on–chip cache memory and 4 MBytes external DRAM local memory
- 4 bidirectional high speed links of max data rate: 20 Mbits/sec (unidirectional) and 18.8 Mbits/sec (bidirectional)

3.2 Hierarchy - Partitions

The 512 transputer processors of the GCel3/512 computer are organized in a hierarchical structure (fig. 2). Each transputer is a processing unit. Eight processing units form a processing board. Two processing boards compose a cluster. A cluster contains 16 transputers: it is the atom of the GCel3/512 machine and is the smallest unit that can be accessed by a user. Four clusters make a GigaCube and eight gigacubes make the whole GCel3/512 machine. From the user's point of view the whole 512–transputer network is split up in partitions of varying sizes. Each partition consists of one or more clusters. A partition may contain parts of other partitions.

3.3 Hosts

The two–dimensional grid network of transputers is connected to the outside world through 2 external hosts which provide 12 high speed links (20 Mbits/sec). The hosts are two Sun/SPARC stations running the PARIX operating system. PARIX (PARallel extensions to unIX) is a UNIX–based operating system with extensions (tools and programs) that

support the transputers' communications with each other. PARIX is responsible for the efficient usage of the transputers' grid network. The PARIX programming model is based on:

- Assigning a specific ID to each processor and running identical program code on each processor.
- Execution of the code segments depending on the processor ID

Transputers communicate with each other using virtual links which reside on the physical links. The user can define his own virtual topologies, by specifying the appropriate virtual links. The most common topologies (like the Ring or Tree structures) are implemented in a library which is available to the user. This library gives reliable, flexible and completely tested topologies, which are mapped in the best way on the physical transputer network and they can scale up as the partition size increases.

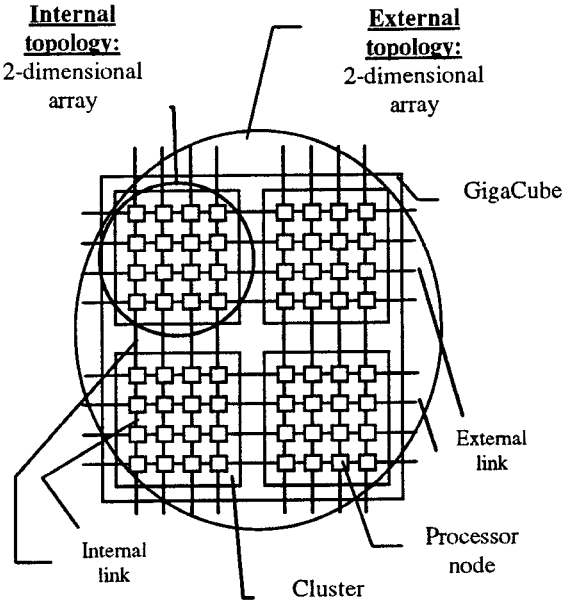

Figure 2: The hierarchical structure of GCel 3/512 machine

4. The General Algorithm

In this section, a general algorithm is presented for parallelizing an information retrieval system based on the Vector Space Model. In order to develop such an algorithm several assumptions have to be considered, concerning both the retrieval techniques and the parallel environment that are used. Consequently, our work is based on the following assumptions:

1. Each document is represented by a term vector (D_t) in the form stated by section 2. The user queries are also represented by a term vector (q_t) of the same type.
2. The retrieval task consists of the classical scoring and ranking subtasks ([15]). The scoring subtask is performed by computing the inner product between the query vector and all the document vectors.
3. The parallel environment that is used provides a host processor P_h and a specific number P of working processors $P_t (i = 1...P)$. Each of the working processors has its own local memory of size M (Mbytes), whereas P_h is linked to all P_t via a specific interconnection network. We also assume that one of the working processors (P_{tr}) serves as the "root" of the other working processors.
4. All the processors have access to a common disk space ($DISK$). All the document vectors (D : total # of doc-vectors, C : total space of doc-vectors in MBytes) are stored in this common space.

An efficient parallelization of an information retrieval system that meets the above assumptions can be achieved via the following simple algorithm:

Algorithm GNR ($\forall$ processor P_t, $i = 1...P$)

1: Read specification file SF_t
2: Determine (based on SF_t) which N_t doc-vectors of total space $\frac{C}{P}$ must be read from $DISK$.
3: Read the first N_{t0} (of total N_t) doc-vectors of total size $M - d$.
4: Receive query–vector q_i from P_{tr}.
5: Compute inner products between q_t and each of the N_{t0} doc-vectors.
6: Rank the results (keeping the X high-score doc-vectors' IDs) in set RD_t.
7: DiskReadingTimes (DRT) := 1 ;
8: *WHILE* ($DRT \leq \frac{C}{(M-d)\times P}$) *DO*
    Read the next N_{tDRT} doc-vectors of total size $M - d$ from $DISK$.
    Compute the relevant inner products.
    Rank the results and merge the X high-score ones in set RD_t.
    $DRT := DRT + 1$;
9: Send RD_t to P_{tr}.

P_h (host processor)

1: Formulate query-vector q_i
2: Send q_t to P_{tr}
3: Receive ranked results (X docs) in RD_h
4: Present the X most relevant documents

P_{tr} (root working processor)

1: Receive q_t from P_h
2: Send q_t to all P_i
3: Behave as a common P_t (without step 9)
4: Receive $P - 1$ ranked sets RD_t from the other $P - 1$ P_t.
5: Merge your own RD_{tr} with all others
6: Send the final RD_{tr} to P_h

The above algorithm implies that the total amount of data (document vectors) is shared almost equally between the P working processors (steps 1,2). This is achieved by the use of one specification file (SF_t) for each processor. This file is constructed off-line and describes the specific document vectors corresponding to each processor. This file may contain multiple entries, one for each possible value of P (# of processors used). Then each processor iteratively read-

s an amount $M - d$ (d stands for the other main memory needs of P_i, i.e. virtual links) of data which correspond to a specific number of N_{iDRT} doc-vectors that fit in the main memory of the P_i processor (steps 3,8). Consequently, the scoring and ranking subtasks (steps 5,6) are performed for these N_{iDRT} doc-vectors. File SF_i also stores the information about which N_{iDRT} doc-vectors each processor has to read each time. The result consists of the X highest–scored doc–vectors (set RD_i) whose identity numbers are then sent to the root processor P_{ir}.

The host processor (P_h) should normally have to perform *a)* the user–interface tasks (query formulation and presentation of results) and *b)* the communication task over the transputer network (query transmission and collection of the results). However, it is preferable if the latter is given to one of the working processors so that all of the processing required for the information–retrieval task is carried out by the working processors $P_i(i \in 1...P)$, whereas P_h is completely dedicated to the user interface task. Additionally, the above selection allows the construction of a virtual tree network structure over the P working transputers. Through this tree network (see section 6 for detailed description and analysis), the transmission of the query and the collection and merging of the ranked results are distributed to the h levels ($h = \log P$, for a binary tree) of the tree, providing high communication efficiency.

There are several problems concerning the above general algorithm. *First*, it is difficult to achieve the near–equal sharing of data amongst the processors that is required for the proper synchronization of the system. *Second*, the number of times ($\frac{C}{(M-d)\times P}$ times) that each processor has to read from $DISK$ N_{iDRT} document vectors of total size $M - d$, is critical and it dominates the performance of the system. In the best case (each processor has to read from $DISK$ only once during the system's start-up) the whole system can manage a total of $P \times (M - d)$ Mbytes of data. *Third*, the communication times due to query transmission (step 2 of P_{ir} and collection of results (step 4 of P_{ir}) seem to be quite ineffective for large values of P, since there is a significant communication overhead on the links of the root processor (P_{ir}). For simplicity, the algorithm does not incorporate clustering mechanisms and corresponding cluster-based search techniques which would lead to several additional problems (i.e. the MDAP problem, see [16],[17]). Such cluster based search techniques (i.e. centroid mechanisms, [15]) could help in the case of very large collections that don't fit in main memory. In this case, an appropriate clustering algorithm could lead to a final document set (for each query) that fits properly in main memory. Thus, the total time spent for the retrieval of relevant documents in response to a specific query (see evaluation results in section 6) would be overloaded by only one $DISK$ access cycle, and it would be independent of the total size of the collection.

In our implementation we partially overcome the above problems by adopting the assumptions that *a)* the document vectors are shared equally among all the processors and *b)* each processor accesses the $DISK$ for reading only once during the start–up of the system. However, the latter implies (concerning the GCel3/512 machine) that the whole system can manage efficiently a total of almost 2 GBytes of data, which is quite satisfactory. Thus, we focus our efforts on the minimization of the communication overhead over the two–dimensional grid interconnection network of the GCel3/512 machine. This is achieved by the use of virtual tree topologies that connect all the working processors P_i to the root processor P_{ir}. This is discussed in the following section. Moreover, in section 6 we discuss the problem of the size of the exchanged messages (i.e. the query-vector q_i message and the message RD_i of the ranked document vectors).

5. The Implementation

5.1 Virtual Tree Topologies

Tree structures have been widely used in parallel processing systems. They have the potential to parallelize a computational task efficiently, while keeping the total communication times low by distributing them all over the levels of the tree (i.e. a complete binary tree–structure of P processors implies only $\log P$ communication steps that are performed in parallel). In our implementation we use complete tree structures which maximize the performance of the underlying parallelism.

As mentioned in section 3, the physical network of the GCel3/512 machine consists of a two–dimensional grid. The way that a virtual complete tree structure resides on this network of transputers is of great interest. In fig. 3 the complete binary tree (degree $d = 2$) topology of $P = 15$ processors is indicated as an example ($h = \log(P + 1) - 1 = 3$ communication levels).

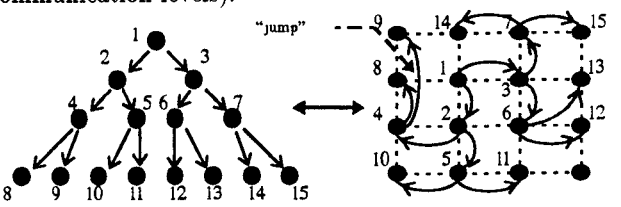

Figure 3: The binary tree structure over the 2-D grid

Fig.3 also indicates a communication overhead (called "*jump*"). A *jump* is the execution of one communication step between two adjacent nodes of the virtual tree, through two or more communication steps on the physical network. However the architecture of the GCel3/512 machine guarantees that this overhead is not proportional to the number of physical communication steps that one *jump* needs. In fact, the overhead is strictly minimized because the processors that are included in a *jump* serve only as communication switches without performing any computational task. Thus a high communication efficiency is preserved, especially when the binary tree structure is used.

5.2 Modification of the GNR Algorithm

Applying such a tree structure (of degree d) over the physical network that connects P_{ir} with the other P_i processors, the GNR algorithm can be modified as follows:

Algorithm MoGNR ($\forall$ processor P_i, $i = 1...P$, $P = 2^h - 1$)

1: *IF* (not Root) *THEN*
Receive query vector q_i from the parent node

2: *IF* (not Leaf) *THEN*
Send query vector q_i to the d children nodes
3: Compute the inner products between q_i and the local $\frac{D}{P}$ doc–vectors
4: Rank the results into the local RD_i set (keeping the X high–scored docs)
5: *IF* (not Leaf) *THEN*
5.1: Receive the d RD sets from the corresponding children
5.2: Merge the above sets into the local RD_i set (keeping the X high scored ones)
6: *IF* (not Root) *THEN*
Send the local RD_i set to parent node
ELSE Send the local RD_i set to host processor P_h.

Each node–processor (starting from P_{ir}) sends the query–vector q_i to its d children till q_i arrives at the leaf–processors. Then all the processors compute the similarities between q_i and their $\frac{D}{P}$ doc–vectors. Consequently, each processor ranks the results into its RD_i set. Finally the merging is performed distributively on the h levels of the tree. Each node–processor collects the RD sets from its d children and it merges them into its RD_i set which is then sent to the corresponding parent node. This step is repeated once for each level of the tree until the final RD set is constructed on the root P_{ir}.

5.3 Performance Analysis of the Modified GNR Algorithm

In this section we'll try to show how the above tree-structured algorithm is appropriate to the retrieval task. This is done by analytically measuring the total time T_{total} that is spent by the root working processor P_{ir}. T_{total}, consists of the following intervals:

1. The time for delivering the query–vector q_i (steps 1,2) to every processor (T_q).
2. The time for scoring (step 3) in every processor (T_s).
3. The time for local ranking (step 4) in every processor (T_{lr}).
4. The time for the transmission of the results (steps 5.1, 6) to the parent nodes (T_{tr}).
5. The time for merging the results (step 5.2) in every level of the tree (T_{mr}).

Thus, $T_{total} = T_q + T_s + T_{lr} + T_{tr} + T_{mr}$. Since there are P processors, virtually organised in a complete tree structure of h levels and degree d (# of children for each node), in the following we compute each of the above times.

T_q consists of the communication times spent by each node–processor in order to submit the query q_i to its d children. This task is executed in parallel for all the processors at the same level. Thus, the query vector arrives at the leaves of the tree in $h-1$ steps. We assume that the time that one node needs in order to submit q_i to one of its children (one query–submission step) is t_q. We also accept that there is a constant time overhead in each step equal to c, caused by the *"jumps"* of the virtual tree and by the broadcasting overhead of the information from node P_i to its d children. This means that

$$T_q = c\,(h-1)\,t_q \tag{1}$$

Making the same assumptions we can say that

$$T_{tr} = c\,(h-1)\,t_{tr} \tag{2}$$

where t_{tr} is the transmission time of the ranked results from node P_i to its parent node.

T_s is equivalent to the time that the leaf processors need to compute their documents' scores. The corresponding time that the node–processors need is not envolved in the calculation because the node processors start their work before the levels do so. Thus, since the leaf–processors work in parallel, T_s is equal to the work time of one leaf processor. Consequently, supposing that the whole scoring work needs time t_s for serial processing, T_s becomes:

$$T_s = \frac{1}{P}\,t_s \tag{3}$$

T_{lr} can be analysed in the same way as (3). Thus, supposing that the whole ranking work needs time t_{lr} for serial processing, T_{lr} becomes:

$$T_{lr} = \frac{1}{P}\,t_{lr} \tag{4}$$

Finally, T_{mr} represents a work task (merging of d ranked RD sets) which is performed in parallel on the nodes of each one of the levels of the tree. Thus, assuming that the time spent at each processor is t_{mr}, T_{mr} becomes:

$$T_{mr} = (h-1)\,t_{mr} \tag{5}$$

From equations (1)–(5),

$$T_{total} = c\,(h-1)\,(t_q + t_{tr}) + \frac{1}{P}\,(t_s + t_{lr}) + (h-1)\,t_{mr} \tag{6}$$

5.4 Discussion

With regard to the local – for each node – computational tasks (scoring and ranking), equation (6) implies that they are properly shared between all processors. Thus, the total time required for them is minimized almost perfectly (divided by P). The only overhead is the time required for merging $((h-1)T_{mr}$. However this overhead is not significant since merging is performed over ranked sets of very small size. In fact, each node–processor has to merge d ranked sets of size X where X (# of relevant retrieved documents) usually ranges from 10 to 100. Since the merging algorithm takes time proportional to the sum of the sizes of the ranked sets that are to be merged, the reader can see that T_{mr} becomes too low compared to T_s and T_{lr}.

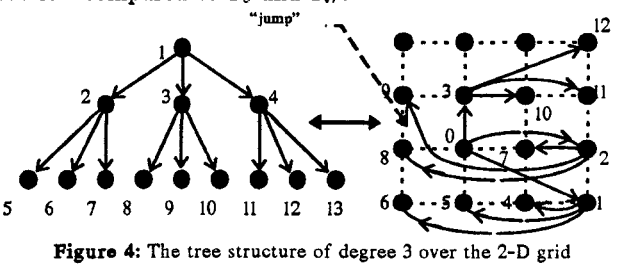

Figure 4: The tree structure of degree 3 over the 2-D grid

Concerning the communication times (T_q, T_{tr}), equation (6) clearly implies that they are proportional to the height (h) of the tree. This is an acceptable performance since h becomes equal to $\log P$, for a complete binary tree structure (fig. 3). Thus, as P increases, the communication times grow very slowly (linearly on h and logarithmically on P since $h = \log_d(n)$, where d is the degree of the tree and n is the # of the leaves). Consequently, we shall focus on the influence of the factors c, t_q, t_{tr}.

The coefficient c is affected by *a)* the "*jumps*" of the virtual tree structure over the physical transputer network and *b)* the "crowd" that is caused around the links of the processors. Although the former is not significant (as discussed in the begining of this section) a comparison between fig. 3 and fig. 4 indicates that the influence of the corresponding time overhead grows as d increases.[2] However the latter reason, (b), is more critical. In fact, as the value of d increases the corresponding overhead increases proportionally, leading to undesired communication traffic. Consequently, the behaviour of the coefficient c over the communication tasks of the processors leads to the intuitive conclusion that the total performance is better for low values of d (i.e. $d = 2$, binary tree), although the value of h decreases as d increases.[3]

Finally, it's obvious that the size of the message of the query vector q_t influences the time t_q almost proportionally. Also, the size of the RD sets messages (equal to X) influences the time t_{tr} in the same manner. Evaluation results that indicate the performance of the system are presented in the following section. All the above critical values (h, d, size of q_t, size of RD sets) are varied to ascertain their effect on perfomance.

6. Efficiency of Searching

In this section, the performance of our system is measured experimentally. Probably the most widely used performance measurement for parallel systems is an application dependent one, the so–called *speed up* measurement ([6]). As a derivative of the speed–up, the *utilization* measurement can also be directly evaluated. Specifically, having P processors that work in parallel the speed–up S_P and utilization U_P measures are defined as

$$S_P = \frac{T_1}{T_P} \ , \ U_P = \frac{S_P}{P}$$

where T_1 is the time to carry out the whole system's work on one processor and T_P corresponds to the time needed when P processors are used for the same work in parallel. In the ideal case, $S_P = P$, and the system is said to exhibit a *linear speed–up*. It must be emphasised that it is practically impossible to achieve linear speed–up. The actual degree of speed–up that can be obtained is controlled by Amdahl's law ([18]). This states that if a given problem has a fraction f of sequential operations, then the maximum speed–up which can be achieved with P processors is

$$S_P \leq \frac{1}{f + \frac{1-f}{P}}$$

This implies that, the longer the serial processing required the smaller the speed–up that can be obtained. Moreover the *utilization* U_P stands as a measure of the degree of the speed–up that is achieved when P processors are used. In our system the serial processing consists of the communication tasks, whereas the parallel processing consists of the scoring, local ranking and merging subtasks. Therefore, the minimization of the former and the maximization of the latter can lead to high speed-up and utilization values.

Keeping in mind the above considerations, we experimentally evaluate our system's performance over the standard CRANFIELD collection which consists of 1400 documents in aerodynamics. As stated in section 5 the values of P and d, the query length and the # of documents that are retrieved, influence the system's performance significantly, and thus these parameters are varied in our experiments. In addition to the speed–up and the utilization measurements, the *response time* (R_P) of the system is also evaluated and presented. In our implemenation the response time is the time that is spent by the root processor from the begining of the query submission till the end of the collection of the RD sets and the merging of the results.

In Table 1 the three basic performance measurements (S_P, U_P in percent values and R_P in units of 64×10^{-6} sec.) are presented for the complete binary tree structure (d=2). As we will see later, this structure is the best among all of the possible tree structures of varying degree. For each of the above three measurements there are three different columns for varying query lengths (10, 20 and 100 terms). If we focus on one column (i.e. for query length = 10) we can obtain some general conclusions about the behaviour of the performance measures as the # of processors increases. Each row of the table corresponds to a specific height of the tree structure ($h = 1...9$, $P = 2^h - 1$). Obviously, as the height of the tree increases the speed–up increases too, whereas the response time decreases. Observing the rate of the speed–up increment, which is represented by the utilization measure the reader can see that it slows down significantly for large values of P. Naturally, the bigger the number of P, the bigger the number of communication levels on the corresponding binary tree structure. Thus, for large values of the tree–height (parameter h in eq. (6)), the root–processor remains idle for more and more time until the results arrive from its children. The experimental results that are presented in table 1 imply that the use of up to 63 processors provides quite satisfactory system utilization (almost 50%), whereas the use of 15 processors offers the exceptional utilization value of 80%. Also, the reader may notice that the speed–up measure increases even for 255 processors with query lengths 20, 100. However the above limitations on the system's utilization concerning the number of processors that are used, do not actually correspond to our work.

[2] In fact, a perfect matching without "jumps" could be chosen for $d = 3$ and $h = 3$. The matching indicated in fig. 4, is the result of the corresponding PARIX algorithm used by default from the GCel3/512 machine, which is a generalized algorithm for greater values of d, h, P. In this case there is an obvious and accepted overhead caused by the "jumps".

[3] The corresponding experimental results (section 6) validate this intuition.

		Speed Up - Sp			Utilization - Up			Response Time - Rp		
h	P	10	20	100	10	20	100	10	20	100
1	1	1	1	1	100	100	100	6924	8873	24426
2	3	2,87	2,9	2,91	95,7	96,6	97,3	2411	3056	8366
3	7	5,96	6,12	6,57	85,2	87,4	93,9	1161	1449	3716
4	15	11,63	11,99	12,79	77	79,9	85,3	595	749	1909
5	31	18,86	19,5	21	60,8	62	68	367	455	1158
6	63	26,5	27,05	29,7	42	42,9	47	261	328	821
7	127	31,9	32,7	34,9	25	25,7	27	217	271	699
8	255	31,7	33,2	35,29	12,4	13	13,8	218	267	692
9	511	29,7	30,8	33,14	5,8	6	6,4	233	288	737

Table 1: Measurements for varying query length

		Speed Up - Sp			Utilization - Up			Response Time - Rp		
h	P	1	5	20	1	5	20	1	5	20
1	1	1	-	-	100	-	-	8873	-	-
2	3	2,9	2,92	--	96,66	95,57	--	3056	14577	--
3	7	6,12	6,51	--	87,4	93,1	--	1449	6546	--
4	15	11,99	13,58	13,99	79,9	90,5	93,3	740	3140	12106
5	31	19,5	25,3	26,9	62	81,6	86,7	455	1686	6296
6	63	27,05	44,4	50,9	42,9	70,47	80,8	328	961	3332
7	127	32,7	68,58	87,63	25,7	54	69	271	622	1936
8	255	33,2	90	138,9	13	35,5	54,49	267	474	1221
9	511	30,8	107,2	208,17	6	20,98	40,7	288	398	815

Table 2: Measurement for varying collection size

# retr	Sp	Up	Rp
10	11,99	79,9	740
20	10,42	69,5	872
50	7,22	48,18	1416
100	5,40	36,04	2497

# retr	Sp	Up	Rp
10	12,79	85,3	1909
20	11,96	79,7	2055
50	9,69	64,6	2642
100	7,47	49,8	3814

Table 3: Measurements for varying # of retrieved documents

h	P	Sp	Up	Rp
1	1	1	100	8873
2	6	5,36	89,3	1655
3	31	18,1	58,5	489
4	156	24,17	15,49	367

h	P	Sp	Up	Rp
1	1	1	100	8873
2	4	3,77	94,27	2353
3	13	10,32	79,45	859
4	40	22,2	55,6	399
5	121	23,1	19	384
6	364	14,49	3,9	612

Table 4: Measurements for tree structures of degree 3 and 5

In fact the results that are presented in table 1 refer to a very small document collection (almost 2MB). This means that for large values of P the data that are processed by each transputer, and thus the overall computational task that is parallelized, becomes extremely small. Therefore, Amdahl's law implies only a limited degree of the speed–up. In fact, the maximum utilization of the system is achieved when each processor keeps in its local memory as many data as possible.

In table 2, [4] we present the results that are obtained by applying our algorithm to document collections of larger sizes. We have produced such collections by reproducing the standard CRANFIELD collection multiple times (1, 5 and 20 times, keeping the query length on 20 terms). Certain-

[4] The '–' in some cells means that the corresponding x–times CRANFIELD collection does not fit to the total memory of the relevant # processors denoted by that specific row.

ly, concerning the main indexing features (such as distribution of term frequencies, documents lengths etc.), an x–times CRAN collection is not expected to behave in the same way as a unique large collection of the same size. However, an x–times CRAN collection can effectively represent a typical large-scale computational load which is necessary when the communication load of the system tends to be critical (as Amdahl's law implies).

The reader may notice the exceptional utilization value of 80% that is obtained when 63 processors are used with the 20–times CRANFIELD collection. The speed–up inreases continously even for 511 processors. The 40% utilization that is provided when 511 processors are used is also of great interest. Additionally, trying to achieve the maximum possible efficient use of the GCel3/512 machine we've obtained an almost 60% utilization over the 1000–times CRAN collection using all the 512 transputers. However the above limitation can be overcome if this maximum–size collection could be shared to the processors completely uniformally. A recent simulation study indicated that a utilization of almost 90% can be achieved if the 1000-times CRAN collection is uniformally allocated to the 512 processors of the GCel3/512 machine. In conclusion, the GCel3/512 machine can offer very efficient parallel information retrieval even when 512 processors work in parallel. The high local memory capacities of the transputers and the minimization of the total communication times via the virtual binary tree structure make the latter possible.

Concerning the varying query lengths, table 1 indicates that the performance is better (in terms of utilization measure) with long queries. This happens because an increase in query length leads to a corresponding increase in the computational task of each processor. Specifically, there is an almost linear increase on the scoring subtask (time T_s, equation (3)). Thus, as Amdahl's law implies, the speed–up and utilization measures normally increase too. However, the longer queries influence significantly the query submission time T_q (eq. (1)). Eventually, as table 1 states, the performance improvement (due to increase of query length) is clear but substantially restricted by the communication overheads.

Table 3 shows the influence on system perfomance of the number of top-ranked documents that are retrieved. The results that are presented in the two tables correspond to the binary tree structure of 15 processors with a query length of 20 and 100 terms. The rows of each table correspond to the varying # of retrieved documents. Again, both the computational and the communication tasks are increased. As the number of retrieved documents increases, the times T_{lr} and T_{tr} (eq. (2),(4)) increase too, and they adversely affect the speed–up measure (as Amdahl's law implies). However, the increase in the computational task here is not significant enough (as in the case of queries length parameter) and thus it is dominated by the corresponding communication overhead. Eventually, as table 3 indicates there is a clear decrease of the utilization measure when the number of retrieved documents grows in size.

In order to demonstrate that the binary tree structure is the best among the other tree structures, we present in table 4 some experimental results for tree–structures of degree 3 and 5 (keeping the query length on 20 terms). A comparison of these results to the relevant ones of table 1 is quite difficult since the values of P are not the same. As an example, the reader can see that for $P = 31$ the binary tree structure provides a utilization of 62%, whereas the structure of $d = 5$ provides a utilization 58%. The "crowd" that is caused around the links of each processor and the longer "jumps" over the two–dimensional grid are responsible for this situation.

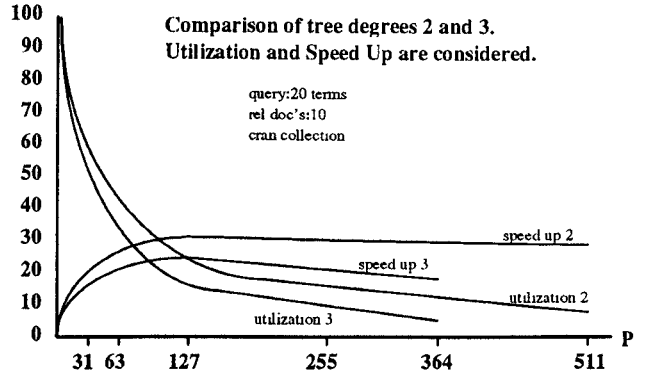

Figure 5: Comparison of tree structures. Speed up and Utilization curves for varying tree degrees.

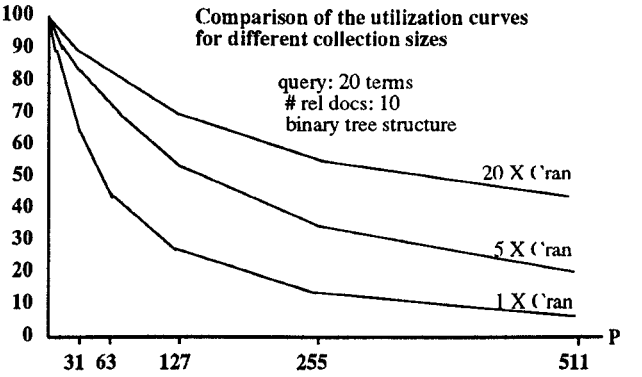

Figure 6: Comparison of utilization for varying size of document collection

A better comparison between the tree structures of degrees 2 and 3 can be obtained from the diagram of figure 5. The corresponding *speed–up* and *utilization* curves over the single CRAN collection are presented in this diagram. Generally, for each tree structure, the speed–up curve starts at '1' (the trivial case of $P = 1$) and it rises to higher values. The curve tends to a straight line over a wide range of larger P values. For some very large value of P the curve may start to go down again meaning that the communication overhead tends to dominate the parallelizable computational tasks. This happens when the single CRANFIELD collection is used and $P > 256$.

The utilization curve behaves in almost the opposite way, as the corresponding definitions imply. Specifically, figure 5 clearly indicates that the curves for the binary structure are uniformly better than the curves which correspond to

$d = 3$. Additionally, in figure 6 the utilization curves for 1, 5 and 20–times CRAN collection are presented. These curves (especially the one for 20–times) validate the great efficiency that is provided when the processors are heavily utilized, as described in table 2.

Conclusion – Further Work

Our work demonstrates the worth of using a high performance, massively parallel machine (a Parsytec GCel3/512 machine) for applying parallelism on information retrieval tasks.

Outstanding performance (in terms of the speed–up measure) is achieved for up to 64 working transputer–processors, by organising them into virtual binary tree structure. As the evaluation results indicate, the binary tree structure is the best among all the other tree structures of varying degree, when the physical network is a two–dimensional grid. Because of the logarithmic height of such a structure the performance remains quite satisfactory even when using 256 or 512 transputers in parallel. Also, the performance measures remain high with long queries and with large numbers of retrieved documents. The corresponding efficiency that is provided by the massive use of even 512 processors with 4 MBytes local memory in parallel allows the effective application of parallelism to text retrieval systems that are based on the Vector Space processing model. Thus, a parallel information system based on this model has been developed which preserves both the needs for high retrieval effectiveness (recall & precision) and for low total response times.

The incorporation of clustering mechanisms and efficient document allocation algorithms on the high performance parallel environment of the GCel3/512 machine remains to be studied. Also the possible use of more efficient virtual structures over the 2–dimensional grid processors' network is of great interest. For example *fat–tree* structures (i.e. meshes of trees) could help further to minimize the overall communication times, thus leading to more efficient parallel implementations.

Acknowledgements

We would like to thank the unknown referees and Prof. P. Willett for their helpful comments.

References

[1] M. Lafazanis, B. Mamalis, P. Spirakis, B. Tampakas, and A. Tsakalidis, "*FIRE: A Flexible Tool for Efficient Information Retrieval*", RIAO '94 Conference, New York, October 11–13, 1994 (accepted for prototype demonstrations).

[2] Frank Tiedt, "*Parsytec GCel Supercomputer*", Technical Report, Parsytec Computer GmbH, July 1992.

[3] C. Stanfill and B. Kahle, "*Parallel Free Text Search on the Connection Machine System*", Communications of the ACM, Vol. 29, No 12, pp. 1229–1239, 1986.

[4] C. Pogue and P. Willett, "*Use of Text Signatures for Document Retrieval in a Highly Parallel Environment*", Parallel Computing, Vol.4, pp. 259–268, 1987.

[5] G. Salton and C. Buckley, "*Parallel Text Search Methods*", Communications of the ACM, Vol. 31, No 2, February 1988, pp. 202–215.

[6] J. Cringean, R. England, G. Manson and P. Willett, "*Parallel Text Searching In Serial Files Using A Processor Farm*", ACM SIGIR'90, pp. 429–452, 1990.

[7] F. Grandi, P. Tiberio and P. Zezula, "*Frame Sliced Partitioned Parallel Signature Files*", ACM SIGIR'92, Denmark, pp. 286–297, June 1992.

[8] C. Stanfill, "*Partitioned Posting Files: A Parallel Inverted File Structure for Information Retrieval*", ACM SIGIR'90, pp. 413–428, June 1990.

[9] C. Stanfill, R. Thau and D. Waltz, "*A Parallel Indexed Algorithm for Information Retrieval*", ACM SIGIR'89, June 1989, pp. 88–97.

[10] H. Stone, "*Parallel Querying of Large Databases: A Case Study*", IEEE Computer, October 1987, pp. 11–21.

[11] M. Flynn, "*Some Computer Organisations and their Effectiveness*", IEEE Transactions on Computers, C–21, pp. 948–960.

[12] J. Cringean, M. Lynch, G. Manson and P.Willett, "*Best Match Searching in Document Retrieval Systems Using Transputer Networks*", London: British Library Research and Developement Department, 1989.

[13] G. Salton et al., "*A Vector Space Model for Automatic Indexing*", Communications of the ACM, 18: 613–620, 1975.

[14] C.J. Crouch, "*A Cluster Based Approach to Thesaurus Construction*", ACM SIGIR'88, pp. 309–320, June 1988.

[15] G. Salton and McGill, "*An Introduction to Information Retrieval*",2nd edition, MacGraw–Hill, c1983.

[16] O. Frieder and H.T. Siegelmann, "*On the Allocation of Documents in Multiprocessor Information Retrieval Systems*", ACM SIGIR'91, pp. 230–239, 1991.

[17] R. Sharma, "*A Generic Machine for Parallel Information Retrieval*", Information Processing and Management, Pergamon Press, pp. 223–235, 1989.

[18] G. Amdahl, "*The validity of the single processor approach to achieving large scale computing capabilities*", AFIPS Conference Proceedings, 30, 483–485.