EFFICIENT BSP/CGM ALGORITHMS FOR TEXT RETRIEVAL

Damianos G. Gavalas
Department of Cultural Technology and Communication
University of the Aegean
Mytilene, Greece
email:dgavalas@aegean.gr

Charalampos G. Konstantopoulos
Research Unit 1
Research Academic Computer Technology Institute
Patras, Greece
email:konstant@cti.gr

Basilis G. Mamalis and Grammati E. Pantziou
Department of Informatics
Technological Educational Institute of Athens
Athens, Greece
email: {pantziou,vmamalis}@teiath.gr

ABSTRACT
In this paper we present efficient, scalable and portable parallel algorithms written in the *Coarse-Grained Multicomputer (CGM)* and the *Bulk Synchronous Parallel (BSP)* models for the off-line clustering, the on-line retrieval and the update phases of the text retrieval problem based on the vector space model and using clustering to organize and handle a dynamic document collection. To the best of our knowledge, our parallel retrieval algorithms are the first ones analyzed under specific parallel models, which capture within a few parameters the characteristics of the parallel machine.

KEY WORDS
BSP model, CGM model, parallel algorithms, text retrieval, document clustering.

1 Introduction

The continuous growth of electronically available information, which is dispersed over a local or a wide area network or usually, over different internet locations, together with the increasing requirements for immediate processing and access to the information, make the use of parallel processing indispensable to the information retrieval field. Until today, many attempts have been made to apply parallel processing and build efficient and practical parallel text retrieval systems [1, 2, 3, 4], which achieve significant response-time improvements compared with serial text retrieval systems.

In this work, we present the first efficient, scalable and portable parallel algorithms written in the *Coarse-Grained Multicomputer (CGM)* and the *Bulk Synchronous Parallel (BSP)* models for the off-line clustering phase, the on-line retrieval phase and the update phase of the text retrieval problem based on the vector space retrieval model and using clustering to organize and handle a dynamic document collection.

We prove that the off-line clustering phase requires $O(p)$ computation and communication rounds/supersteps, where p is the number of processors of the parallel computing system. We also show that the local memory required in each processor for this phase is $O(\frac{n}{p}\max\{r, d_{sim}\})$ at most where n is the total number of documents of the information retrieval system and r and d_{sim} are two parameters that depend on the maximum number of terms in each document and the "similarity degree" of the n documents respectively. In addition, we show that the maximum number of operations in each computation round is $O(\frac{n^2}{p^2}r)$ due to the sparse matrix multiplication necessary for finding document similarities. It should be noted that the above amount of local computation and local memory per processor is the least possible and match in aggregate those ones of the corresponding serial information-retrieval algorithm.

We also prove that for both the on-line retrieval and the update phases, $O(1)$ communication rounds are necessary with $O(\frac{n}{p}r)$ local computation per round. The local memory per processor required for this phase is $O(\frac{n}{p}r)$ at most.

2 The BSP and CGM Computing Models

A *BSP* computer [5] is a collection of p processor/memory modules connected by a communication infrastructure that can deliver messages in a point to point fashion between the processors. A *BSP* computation is divided into a sequence of supersteps separated by barrier synchronizations. Each superstep is divided into a computation and a communication superstep. In a computation superstep the processors perform computations on data that were present locally at the beginning of the superstep. In a communication superstep data is exchanged among the processors via the communication network. The parameters of a *BSP* computer used to solve a problem are the problem size N, the number of processors p, the bandwidth parameter g i.e. the ratio of overall system computational capacity (number of computation operation) per unit time divided by the overall system communication capacity (number of messages of unit size that can be delivered by the underlying communication network) per unit time, and the latency parameter L,

which is the minimum time between two consecutive synchronization operations.

The cost of a *BSP* algorithm equals to the sum of the costs of its supersteps. The cost of a superstep is given by the following formula

$$T_{superstep} = max\{L, T_{comp} + T_{comm}\}$$

where T_{comp} is the maximum (over all processors) cost for local computations and T_{comm} is the maximum time needed for transmitting all the outgoing messages to their target processors. If we consider that during the communication superstep each processor sends and receives at most h 1-word messages (i.e. an h-relation has to be implemented during the communication superstep) then $T_{comm} = g \cdot h$. In that case, we have

$$T_{superstep} = max\{L, T_{comp} + g \cdot h\}$$

The *Coarse-Grained Multicomputer (CGM)* [6, 7, 8] model is a simplified version of the *BSP* model and its main purpose is the design of simple and practical yet theoretically optimal or efficient parallel algorithms for coarse-grained parallel systems. A *CGM(N,p)* computer consists of p processors with N/p local memory ($N/p \gg 1$) each connected by a router that can deliver messages in a point to point fashion. A *CGM* algorithm consists of an alternating sequence of *computation rounds* and *communication rounds* which are separated by barrier synchronizations. A computation round is equivalent to a computation superstep in the *BSP* model [5, 9], and therefore the total computation cost is defined analogously. A communication round consists of a single h-relation with $h = O(N/p)$, i.e., each processor sends $O(N/p)$ data and receives $O(N/p)$ data. The main advantage of the *CGM* model is that it allows us to model the communication cost of a parallel algorithm by a single parameter, i.e. the number of communication rounds [7]. A practical assumption commonly met in most proposed CGM algorithms (e.g. see [7]) is that $N/p > p^{\epsilon}$ where constant $\epsilon > 1$. The most important implication of this assumption is that the basic operation of sorting can be executed in a constant number of computation and communication rounds [10, 11].

Note that every *CGM* algorithm is also a *BSP* algorithm but not vice versa. Note also that a CGM algorithm with λ rounds and computation cost T_{comp} corresponds to a BSP algorithm with λ supersteps, communication cost $O(\lambda(g\frac{N}{p} + L))$ and the same computation cost.

3 The VSM-based Text Retrieval Problem

The *Vector Space Model (VSM)* has been used as the basis for many ranking Information Retrieval (IR) systems [12]. The model assumes that we are given a document collection of size n and an array $(t_1, t_2, ..., t_m)$ of m different terms, which are word stems extracted out of all n documents. According to the model, each document D_i in the document collection is indexed by a so-called *document-term vector* d_i, after the application of suitable *stopping* and *stemming* procedures. Specifically, each document D_i is represented by the vector

$$d_i = (w_{i1}, w_{i2}, ..., w_{im})$$

where w_{ij} is a weight value ranging from 0 to 1 (assuming normalization) and representing the significance of term t_j for the specific document D_i. Each weight value w_{ij} assigned to each document D_i is related to the frequency of occurrence of term t_j in both the specific document D_i and to the whole document collection. A weight value w_{ij} that is equal to 0 usually means that the term t_j is not appearing in document D_i. In a similar manner, the documents' term vectors may also be expanded by additional document identifiers (not only word stems) such as phrases, thesaurus classes etc.

The model assumes that each user query Q_i (which actually is a set of words) is indexed in the same way, thus resulting to a *query-term vector* q_i of almost the same type as d_i. Specifically, each query Q_i is represented by the vector

$$q_i = (q_{i1}, q_{i2}, ..., q_{im})$$

where q_{ij} is usually set either to 1 or to 0 depending on the presence of term t_j in the query or not. However, a query-term vector may consist of values ranging from 0 to 1 (assuming normalization), in the case that the user assigns by himself weights on the query terms, depending on their significance for the search.

According to the vector space retrieval model, the query vector q_i is compared (applying a suitable similarity measure) to each one of the n document term vectors, yielding to a corresponding relevance value (score) $sim(q_i, D_j)$, for each document D_j, $j = 1, ..., n$. The documents are then ranked according to their scores and the top-score ones are supposed to be the most relevant to the query.

A valuable extension [12], in order to restrict the total retrieval time, is to group the documents into a number of document clusters. This can be done by computing the pairwise similarities $sim(D_i, D_j)$ for all pairs of documents D_i, D_j ($i \neq j$) of the document collection and then applying a suitable graph-based algorithm - i.e. connected components' evaluation with a specific similarity value as the adjacency criterion - over the corresponding documents' similarity matrix. After the formation of the clusters, for each cluster C_i, one centroid-vector c_i is extracted of the following form

$$c_i = (wc_{i1}, wc_{i2}, \cdots, wc_{im})$$

where wc_{ij} i.e., the weight of term j in cluster C_i, is computed as the average of the weights of term j in documents D_{ik} belonging to cluster C_i. Therefore, the corresponding retrieval task can now be performed via a suitable selective cluster-based searching algorithm (by first comparing

q_i to the centroid-vectors c_i in order to find the most relevant clusters) and finally only a fraction of the whole set of documents in the collection (those belonging to the most relevant clusters) has to be exhaustively compared to the user-query.

Our parallel text retrieval algorithm presented in the next section follows this last approach to efficiently retrieve documents.

4 The VSM-based text retrieval algorithm

4.1 The off-line preprocessing/clustering phase

We assume that we are given p *BSP/CGM* processors and n documents $D_1, D_2, ..., D_n$. For each document D_i, $i = 1, ..., n$, its vector representation $d_i = (w_{i1}, w_{i2}, ..., w_{im})$ is given, while each processor keeps n/p documents. Note that the vectors d_i are usually very sparse. The off-line clustering algorithm computes the similarities among the documents, forms the document clusters (centroids), computes the centroid vector of each cluster and evenly distributes the documents of each cluster to the processors. The basic steps of the algorithm are as follows.

Step 1. Consider the $n \times m$ matrix D, whose row i is the vector d_i, $i = 1, ..., n$. Notice that this is a sparse matrix. For each document D_i, $i = 1, ..., n$, use a sparse matrix-vector multiplication algorithm to "multiply" the sparse matrix D with the vector d_i, and therefore to compute the similarity $sim(D_i, D_j)$ of D_i with each other document D_j, $j = 1, ..., n$, $j \neq i$, using the well-known "cosine-similarity" function i.e.

$$sim(D_i, D_j) = \frac{\sum_{k=1}^{m} w_{ik} \cdot w_{jk}}{\sqrt{\sum_{k=1}^{m} w_{ik}^2} \cdot \sqrt{\sum_{k=1}^{m} w_{jk}^2}}.$$

Then, use the similarity matrix to construct a graph $G = (V, E)$, such that there is a node in the graph for each document in the document collection, i.e., $V = \{D_1, D_2, ..., D_n\}$, while there is an edge between two nodes D_i and D_j, if and only if $sim(D_i, D_j)$ is greater than or equal to a threshold value th_1.

Step 2. Apply a connected components algorithm on the graph $G = (V, E)$. The connected components of G comprise the set of clusters C of the document collection, and therefore "similar" documents belong to the same cluster and we expect that relevant documents to each user-query will belong with high probability to some specific clusters.

Step 3. For each centroid (cluster) C_i in C, use the document vectors of its documents to compute the centroid vector c_i.

Step 4. For each centroid C_i in C, evenly distribute the document vectors of the documents of the cluster to the p processors. Therefore, after the execution of this step, the documents of the collection are almost evenly shared among the processors, i.e. each processor holds approximately $\frac{n}{p}$ document vectors in its local memory.

Detailed description and analysis of the off-line clustering algorithm.

Step 1. Each processor q needs to compute the similarity $sim(D_i, D_j)$ of document D_j with each other document D_i in q's local memory. Since each processor keeps $\frac{n}{p}$ different document vectors, processors should broadcast their own $\frac{n}{p}$ vectors to all other processors. After receiving a new set of $\frac{n}{p}$ vectors D_j, each processor computes the similarity $sim(D_i, D_j)$ among its own set of vectors D_i with the vectors D_j, i.e. $\frac{n^2}{p^2}$ similarity values in total.

The sparsity of the matrix D and its constituent vectors d_i, $i = 1, ..., n$, is exploited as follows: Only nonzero vector elements are communicated from a processor to another one, while local computations are performed only if the corresponding vector elements are nonzero. This means that if r is the maximum number of nonzero elements of a vector d_i, $i = 1, ..., n$, then each processor should broadcast at most $\frac{n}{p}r$ vector elements, while each processor q needs to perform at most $\frac{n^2}{p^2}r$ local computation steps in order to compute the $\frac{n^2}{p^2}$ similarity values $sim(D_i, D_j)$ of the received vectors D_j with the local vectors D_i $(i, j = 1, \cdots, N)$.

The whole computation is organized in p rounds. At round i, processor i broadcasts its $\frac{n}{p}$ vectors to all other processors. By using a well-known technique [9, 13], broadcasting can be completed in $O(g\frac{n}{p}r + L)$ time.

After broadcasting, each processor computes the $\frac{n^2}{p^2}$ similarity values among its own vectors and the received vectors of processor i in $O(\frac{n^2}{p^2}r)$ time. Thus, the total running time of round i is $O(\frac{n^2}{p^2}r + g\frac{n}{p}r + L)$ and hence for the p rounds of the whole computation we need $O(\frac{n^2}{p}r + gnr + pL)$ time in total.

It is worth mentioning, that after each computation round, processors need to keep only those similarity values which are above the threshold value th_1. In this way, after the completion of the p rounds, the local memory requirement in each processor for storing these values is at most $O(\frac{n}{p}d_{sim})$ where d_{sim} is the maximum number of documents D_j, such that $sim(D_i, D_j) > th_1$, over all documents D_i. It is also reasonable to assume that parameter d_{sim} is $o(n)$, i.e. much smaller than n, the number of documents in the collection.

With the local $O(\frac{n}{p}d_{sim})$ similarity values as input, each processor creates its own portion of the document similarity graph $G(V, E)$. Specifically, graph G consists of n vertices which represent the n documents of the collection and there is an edge between two vertices V_i, V_j if the

corresponding similarity value $sim(D_i, D_j)$ is above the threshold th_1. Clearly, each processor holds $\frac{n}{p}$ vertices and $O(\frac{n}{p}d_{sim})$ edges of the graph G.

Step 2. The BSP version of the CGM connected components algorithm of Dehne et al. [7] is used to compute the connected components of the graph G constructed in the previous step. Notice that G is a graph with n vertices whose number of edges is at most $m = O(nd_{sim})$. Therefore, the CGM connected components algorithm needs $O(\log p)$ communication rounds and $O(\frac{nd_{sim}}{p})$ local computation per round while the communication cost of the BSP connected components algorithm is $O(\log p(g\frac{nd_{sim}}{p} + L))$ and the computation cost is $O(\frac{nd_{sim}}{p}\log p)$.

Step 3. At the end of Step 2 each document D_i knows the cluster it belongs to while the cluster-heads are distributed to the p processors. During step 3 the following are taking place:

- Using sorting, give to the clusterheads consecutive numbering from C_1 to $C_{|C|}$, where $|C|$ is the number of clusters. Due to [10], sorting can be completed in $O(\frac{n}{p} + g\frac{n}{p} + L)$ time with a constant number of communication and computation rounds.

- By using its local document vectors, each processor q_j, $1 \leq j \leq p$ computes (i) a "local" centroid vector $lc(i, j)$ for each centroid (cluster) C_i which has at least one document vector in q_j's local memory and (ii) the number $b(i, j)$ of document vectors in q_j's local memory that belong to centroid (cluster) C_i. This requires $O(\frac{n}{p}r)$ local computation cost. A crucial observation at this point is that the number of non-zeroes in the "local" centroid vector $lc(i, j)$ is at most $b(i, j) \cdot r = O(\frac{n}{p}r)$. Thus, the computation of the centroid vectors of clusters C_i from the local centroid vectors $lc(i, j)$ should be done in such a way that no processor receive more than $O(\frac{n}{p}r)$ vector elements in the process. This can be achieved by the following three steps.

- For $i = 1, \cdots, |C|$, $j = 1, \cdots, p$, compute the expressions[1]

$$S(i, j) = \sum_{k=1}^{i-1} b(k, p) + \sum_{l=1}^{j} b(i, l)$$

  by adapting the standard prefix-sum technique [9]. It can been seen that the total running time for the calculation of $S(i, j)$ is $O(|C| + g|C| + L)(= O(g\frac{n}{p} + L))$ at most, assuming w.l.o.g. that the number of clusters $|C|$ is at most equal to $O(\frac{n}{p})$.

[1] For i=1, the first sum is equal to 0.

- Every processor q_j, $1 \leq j \leq p$, for each $i = 1, ..., |C|$, sends its "local" centroid vector $lc(i, j)$, to processor q_k, $k = 1, ..., p$, if and only if $(k-1)\frac{n}{p} < S(i, j) \leq k\frac{n}{p}$. Every processor sends all its "local" centroid vectors in one (combined) h-relation. Since each processor sends and receives at most $O(\frac{n}{p}r)$ vector elements, only one $O(\frac{n}{p}r)$-relation is required for this step.

- Each processor q_k uses the "local" centroid vectors $lc(i, j)$, $j = 1, ..., p$ received previously to locally compute the centroid vector c_i, for each cluster C_i such that q_k has received "local" centroid vectors $lc(i, j)$. This step requires at most $O(\frac{n}{p}r)$ local computation. Note that if a cluster C_i is relatively large, its "local" centroid vectors $lc(i, j)$, $j = 1, ..., p$ may be spread among a number of q processors, say processor $j_i, \cdots, j_i + q - 1$. It is possible that the final centroid vector c_i may not contain $O(\frac{n}{p}r)$ elements at most and thus it cannot be stored in one processor. In this case, we keep the non-zeroes of centroid c_i distributed with each of the q processors holding $O(\frac{n}{p}r)$ non-zeroes at most. The distributed computation of the centroid vector c_i can be performed as follows:

  - For each non-zero element of the "local" centroid vectors $lc(i, j)$, form the pair $(k, lc(i, j, k))$ where k is the position of the non-zero element inside vector $lc(i, j)$ and $lc(i, j, k)$ is the value of this element.
  - By using the algorithm in [11], sort pairs $(k, lc(i, j, k))$ by k in $O(1)$ rounds. The total computation and communication cost is $O(\frac{nr \log nr}{p})$, $O(g\frac{n}{p}r + L)$ respectively. Now after this sorting step, all vector elements at the same position k have been placed consecutively.
  - Now, the centroid vector elements can be easily computed, by first adding the proper local elements in $O(\frac{n}{p}r)$ time and then adding the partial sums in one communication and computation round with a total $O(p)$ computation and $O(gp + L)$ communication cost.

  Clearly, the three steps above demand a constant number of communication and computation rounds and the total computation and communication cost is $O(\frac{nr \log nr}{p})$ and $O(g\frac{n}{p}r + L)$ correspondingly.

  Thus, with the completion of step 3 the centroid vectors have been distributed among the processors and each processor has $O(\frac{n}{p}r)$ non-zero centroid vector elements at most in its local memory.

Step 4. This step requires a redistribution of document vectors so that the document vectors relevant to each cluster are evenly distributed among the p processors. By keeping the intermediate results of the prefix computation for $S(i, j)$ at step 3, each processor can easily determine the processors which it should communicate with. Then, the

redistribution involves a $O(\frac{n}{p}r)$-relation and thus it can be completed in $O(g\frac{n}{p}r + L)$ time.

From the analysis above, we can easily verify the truth of the following theorem.

Theorem 1 *By reasonably assuming that $r \geq \log p$ and $n \geq \log n + \log r$, the off-line clustering phase of the VSM-based text retrieval algorithm has a total of $O(\frac{n^2 r}{p})$ computation and $O(gn \max\{r, \frac{\log p}{p} d_{sim}\} + pL)$ communication cost, whereas the local memory requirement for each processor is $O(\frac{n}{p} \max\{r, d_{sim}\})$ at most.*

4.2 The on-line query processing phase

Given the query vector q_i of a query Q_i, the on-line query processing algorithm proceeds as follows:

Step 1. Compute the "cosine-similarity function" of vector q_i with each of the centroid vectors c_k, $(k = 1 \cdots |C|)$. There are two possibilities for this computation. If all the elements of the centroid vector have been stored in a single processor, the computation is local and requires $O(\frac{n}{p}r)$ time at most. Otherwise, if the centroid vector is distributed among q processors, the whole computation needs one communication and two computation rounds. The size of messages exchanged is $O(1)$ at most and the communication required is an $O(p)$-relation. The computation rounds can be completed in $O(\frac{n}{p}r)$ time overall.

Now, for each centroid C_k such that $sim(Q_i, C_k)$ is greater than or equal to a threshold value th_2, consider it as a "qualifying cluster". Then, broadcast the IDs of these clusters to all processors with $O(g|C| + L)(= O(g\frac{n}{p} + L))$ communication cost at most.

Step 2. By using the "cosine-similarity function", each processor locally computes the similarity of Q_i with each one of its local documents that belong to the "qualifying clusters" according to the previous step. Apparently, this local computation requires $O(\frac{n}{p}r)$ time at most. Then, each processor extracts the documents D_l such that $sim(Q_i, D_l)$ is greater than or equal to a threshold value th_3. These documents are considered as "qualifying documents".

Step 3. Rank the qualifying documents according to their similarity scores with the query Q_i. Clearly, this ranking can be easily done by a sorting step [11] with $O(\frac{nlogn}{p})$ computation and $O(g\frac{n}{p} + L)$ communication cost.

Now, we can easily see that the following Theorem holds.

Theorem 2 *The on-line query processing phase of the VSM-based text retrieval algorithm requires $O(\frac{n}{p} \max\{r, \log n\})$ computation and $O(g\frac{n}{p} + L)$ communication time. The local memory in each processor required for this phase is $O(\frac{n}{p}r)$ at most.*

4.3 Dynamic Cluster Maintenance

When clustering is used for organizing a dynamic document collection, a problem frequently arising is how to adapt the already built cluster structure so that it reflects the new content. Although complete re-clustering is the obvious solution, it is generally agreed [14] that this operation should be executed as rarely as possible, because it heavily penalizes the performance of a fully operational IR system. Thus, most of the proposed systems in the literature try to find a compromise between cluster adaptation cost and accuracy of the new clusters.

As a high-performance parallel technique, our VSM-based algorithm is mainly used for handling very large document collections. Thus, we can reasonably assume that the off-line indexing phase starts with a very large number of documents as input, and the clusters, the output of this phase, are relatively stable corresponding to well-separated topics of the document collection. Therefore, during the on-line operational mode of the IR system, document insertions and deletions are not expected to change dramatically the existing clusters.

Based on this assumption, a new document inclusion can be done by simply finding the most similar cluster for the new document, namely the cluster whose centroid vector is the most similar to the new document vector. This is done as follows:

- By performing the first step of the on-line query processing phase, each processor can determine which centroid vector out of its local ones has the greatest similarity with the new document vector.
- The identities of the most similar clusters along with the corresponding similarity values are sent to one processor which then computes the globally maximum similarity value. If the global maximum is greater than a threshold th_4, then the new document is considered to belong to the cluster that corresponds to this maximum.

The processor that will store the new document is selected in a round-robin fashion, i.e. each time the new document is stored in the next higher numbered processor starting from the first. This ensures that all processors take an equal share of the new documents.

On the rare occasion when the new document D_e cannot be considered to belong to any of the existing clusters (the "cosine-similarity" function less than th_4), this document is viewed as a new cluster C_e by itself whose centroid vector equals to the document vector ($c_e = d_e$). In subsequent document inclusions, each time a new document happens to belong to the recently formed centroid C_e then the centroid vector c_e is updated accordingly. Again, the distribution of the new documents to the processors can be done by using the round-robin technique.

It can be easily seen that the inclusion of the new document requires $O(\frac{n}{p}r)$ local computation and $O(gp + L)$

communication time for finding the similarity of the centroid vectors with the new document vector and $O(p)$ computation and $O(gp + L)$ communication time so that the centroid vector which is the most similar to the new document vector is determined.

Note that if the number of documents inclusions exceeds a specific fraction of the total number of documents n, then the off-line clustering phase is ran so that new document clusters are formed (re-clustering) and new centroid vectors are computed.

Regarding document deletions, the deleted document is simply removed from the processor that has been stored. If a large number of documents has been removed from a cluster, the remaining documents can be evenly redistributed among processors in $O(g\frac{n}{p}r + L)$ time.

Therefore, the following Theorem gives the bounds that the update phase of the VSM-based text retrieval algorithm requires.

Theorem 3 *The update phase of the VSM-based text retrieval algorithm requires $O(\frac{n}{p}r)$ computation and $O(g\frac{n}{p}r + L)$ communication time. The local memory per processor required for this phase is $O(\frac{n}{p}r)$ at most.*

5 Conclusions

A notably efficient parallel algorithm for the VSM-based text retrieval problem, has been presented and analyzed by means of the *CGM* and *BSP* cost models. The total cost is kept considerably low, exploiting the inherent sparsity of the documents' and query term vectors, even in the case of the off-line clustering phase, which is the most intensive task in corresponding VSM-based text retrieval approaches with clustering enhancements. Note also that the above algorithm can easily be extended in order to cover (preserving similar efficiency) additional "document identifiers" (not only word stems) for each document in the collection (i.e. phrases, thesaurus classes etc.).

Acknowledgment

This work is co-funded by 75% from E.U. and 25% from the Greek Government under the framework of the Education and Initial Vocational Training II, programme "Archimedes".

References

[1] B. Mamalis, P. Spirakis and B. Tampakas, Optimal High Performance Parallel Text Retrieval via Fat Trees, *Theory of Computing Systems (TOCS) journal, 32*(6), 1999, 591–623.

[2] B. Mamalis, P. Spirakis and B. Tampakas, Parallel Processing of Multiple Text Queries on Hypercube Interconnection Networks, *Intl. Journal on Computers and their Applications (IJCA), 10*(1), 2003, 115–132.

[3] A. Rungsawang, A. Laohakanniyom, and M. Lertprasertkune, Low-Cost Parallel Text Retrieval Using PC-Cluster, *Proc. Euro PVM/MPI 2001*, Santorini/Thera, Greece, 2001, 419–426.

[4] S.-H. Chung, H.-C. Kwon, K. R. Ryu, Y. Chung, H. Jang and C.-A. Choi, Information Retrieval on an SCI-Based PC Cluster, *The Journal of Supercomputing, 19*(3), 2001, 251–265.

[5] L. Valiant, A Bridging Model for Parallel Computation, *Communications of the ACM, 33*(8),1990, 103–111.

[6] F. Dehne, A. Fabri, and A. Rau-Chaplin, Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputer, *Proc. ACM 9th Symposium on Computational Geometry*, San Diego, USA, 1993, 298–307.

[7] F. Dehne, A. Ferreira, E. Caceres, S.W. Wong, and A. Roncato, Efficient Parallel Graph Algorithms for Coarse-Grained Multicomputers and BSP, *Algorithmica, 33*(2), 2002, 183–200.

[8] F. Dehne, A. Fabri, and C. Kenyon, Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time, *Proc. IEEE Symposium on Parallel and Distributed Processing*, Cancun, Mexico, 1994, 586-593.

[9] A. Gerbessiotis, and C. Siniolakis, Primitive Operations on the BSP Model, *Technical Report PRG-TR-23-96*, Computing Laboratory, Oxford University, 1996.

[10] A. Chan, and F. Dehne, A Note on Coarse Grained Parallel Integer Sorting, *Parallel Processing Letters, 9*(4), 1999, 533–538.

[11] M. Goodrich, Communication Efficient Parallel Sorting, *SIAM Journal on Computing*, 29(2), 1999, 416–432.

[12] R. Baeza-Yates, and B. Ribeiro-Neto, *Modern Information Retrieval*, (New York: Addison Wesley, 1999).

[13] B. Juurlink, H. Wijshoff, Communication Primitives for BSP computers, *Information Processing Letters, 58*(6), 1996, 303–310.

[14] F. Can, Incremental Clustering for Dynamic Information Processing, *ACM Transactions on Information Processing Systems, 11*(2), 1993, 143–164.