A Parallel Implementation of the SOM Algorithm for Visualizing Textual Documents in a 2D Plane

Gustavo Arroyave
sgaal295@udea.edu.co

Oscar Ortega Lobo
oortega@udea.edu.co

Andrés Marín
amarin@udea.edu.co

***Abstract*— With the increase of computer usage, the number of digital documents is reaching values that make unviable conducting the tasks of text organization by humans. There is a demand for text organization tools that can operate with little human intervention and that can display the results of the organization in the most commonly used visual interface: the two-dimensional (2D) plane. One of the techniques used for automatic text organization is the Self Organized Map (SOM), a class of artificial neural network introduced by kohonen in 1982. With SOM, documents can be automatically arranged into groups associated with areas of a 2D plane. Most of SOM implementations are secuential. That is, implementations in which a processor carries out one instruction at once. SOM secuential implementations work fine in domains where the number of objects to organize is low. In large-scale domains like text organization, where is not rare to deal with thousands of documents and millions of words, the application of secuential SOM can become unfeasible in terms of computational time. In an attempt for improving the performance of SOM in large-scale domains, parallel implementation of SOM have been devised. Such implementations take profit of the inherent parallel architecture of SOM. Provided that in the parallel implementation of SOM there are several processors, each one carrying out one instruction at once, there is a reduction of the time required for organizing the objects of the domain.**

In this paper, a parallel implementation of SOM is achieved by using a Beowulf Cluster of personal computers each with low processing capacity. The speed-up gained by the cluster parallel implementation was measured. This was achieved by measuring the execution time of both a secuential implementation and a parallel implementation. The evaluation was conducted on a constant text corpus. The cluster implementation obtained will help make viable ongoing research on text organization at University of Antioquia where there is no funding for investing in costly parallel processors.

***Index Terms*— Self-Organizing Map, Parallel Processing, Text Mining, Cluster Beowulf**

I. Introduction

WITH the increase of computer and internet usage, the number of digital documents has became intractable for the task of text organization by humans. In this context, it is evident there is the need of an automatic tool for text organization. The Self-Organized Map (SOM) has gained great acceptance in the last decade for managing large volumes of documents. The Self-Organized Map, proposed by Kohonen [1] in 1982, is often used to project and visualize high-dimensional spaces on an usually 2D plane while preserving the relationships among the input data as faithfully as possible [2]. The output space of SOM is a regular 2D grid of nodes. The input data are distributed all over the grid forming groups of elements with similar features. With the use of SOM for text organization, documents can be automatically arranged into groups that define specific areas of knowledge (i.e. documents belonging to the same group or neighboring groups deal with similar topics). Most of SOM implementations are done using just one monoprocessor computer. Such an implementation is subject to hardware specifications because of the limitations they put both on the dimensionality of the data to be analized or the size of the map [2]. For this reason, we have decided to make a parallel implementation of the SOM algorithm. The selected architecture for the SOM algorithm implementation is the Beowulf Cluster of our laboratory. The Beowulf Cluster let us achieve good performance at low cost compared with those parallel computers such as CNAPS, Connection Machine, DECmpp which are high-performance computers that have cost of several hundred of thousands of dollars, even millions. The speed-up gained by the cluster implementation was measured by comparing the execution time of both a secuential implementation and a parallel implementation. For evaluating the execution time of both implementations we have designed an experiment in which the number of processors was fixed to four, also the input dataset was always the same.

The remainder of the paper is organized as follows. Section III provides an introduction to the Self-Organized Maps. A parallel implementation is discused in section IV. In section V we give a short description of the Beowulf Cluster that we have in our laboratory. We present an experiment design and a short discussion of the obtained results in sections VI and VII respectively. Finally section VIII presents some conclusions and future work is discussed too.

II. Previous Work

Most papers dealing with parallel implementations of the SOM algorithm have based their implementations on parallel computers [2], [5], [6]. The GLOBALOR [5] is a parallel method implemented on a DECmpp 12000/sx massively parallel computer. With 8192 processing elements, it uses the Single Instruction Multiple Data (SIMD) architecture. The *par-SOM* is a software-based parallel implementation of the SOM algorithm for the analysis of high-dimensional input data using distributed memory systems and clusters [2]. The test were conducted on a number of multi-processor machines with dissimilar memory architecture: (a) *SGI PowerChallenge XL* is a 16-processor system with a shared bus. Each processor has a 2MB level 2 cache with a total throughput of the main memory of 1.2GB/s. (b) *[*SGI Cray Origin 2000] is a 64-processor system which adheres to a ccNUMA architecture using 32 SMP node.

Nordström [6] describe different ways to implement SOM on parallel computers. He conducted the test on many paralle computers such as CNAPS, Connection Machine, L-Neuro, MasPar, REMAP, Transputer, Warp and TinMANN.

III. Self-Organizing Maps

The *Self-Organized Map* (SOM) is an unsupervised neural network method that produces a similarity graph of input data. It consists of a finite set of models that approximate the open set of input data, and the models are associated with nodes ("neurons") that are arranged as a regular, usually two-dimensional grid [3]. The models are produced by a learning process that automatically orders them on the 2D grid along with their mutual similarity. The SOM algorithm is a recursive regression process. Regression of an ordered set of model vectors $m_i \in R^n$ into the space of observation vectors $x \in R^n$ can be made recursively as:

$$m_i(t+1) = m_i(t) + h_{c(x),i}(t)\left[x(t) - m_i(t)\right], \quad (1)$$

$$c(x) = arg\min_i \|x \cdot m_i\|, \quad (2)$$

where *t* is the index of the regression step, and the regression is performed for each presentation of a sample of x, denoted $x(t)$. The scalar multiplier $h_{c(x),i}(t)$ is called the *neihgborhood function*, and it is like a smoothing kernel over the grid. Its first subscript $c(x)$ is defined by (2), that is, $m_{c(x)}(t)$ is the model (called the *"winner"*) that matches best with $x(t)$. The comparison metric we use in this paper is the inner product. If the samples $x(t)$ are stochastic and have a continuous density function, the probability for having multiple minima in (2) is zero. With discrete-valued variables, however, multiple minima may occur; in such cases one ot them can be selected at random for the winner.

The neighborhood function is often taken as the Gaussian:

$$h_{c(x),i}(t) = \alpha(t) exp\left(-\frac{\|r_i - r_{c(x)}\|^2}{2\sigma^2(t)}\right), \quad (3)$$

where $0 < \alpha(t) < 1$ is the learning-rate factor which decreases monotically with the regression steps, $r_i \in R^2$ and $r_{c(x)} \in R^2$ are the vectorial locations on the display grid, and $\sigma(t)$ corresponds to the width of the neighborhood function, which is also decreasing monotically with the regression steps. In practice, for computational reasons, $h_{c(x)i}(t)$ is truncated when $\|r_i - r_{c(x)}\|$ exceeds a certain limit.

In agreement with the above description the SOM algorithm can be formalized as follows:

1. Find the node (or model vector) m_i closest to input x
   $\|x(t) \cdot m_i(t)\| = \min_i \|x(t) \cdot m_i(t)\|$
2. Make the nodes in the neighborhood closer to input as given in expression (1)
3. Repeat from step 1 until convergence with ever decreasing neihborhood kernel $h_{c(x),i}(t)$

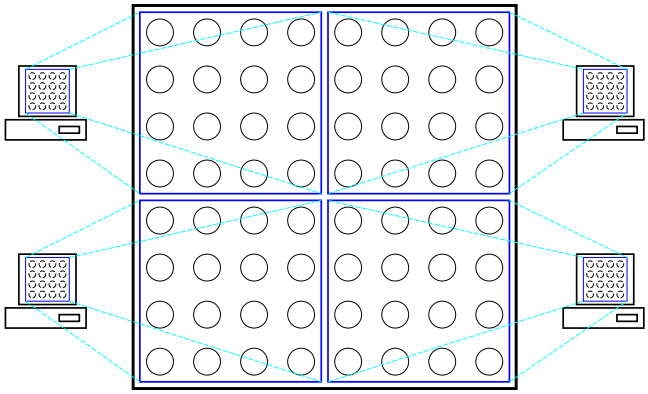

Fig. 1. Multiple map segments of the same size which are maintained by different processors. Thus, the required memory and processing time are distributed across multiple processors.

IV. Parallel SOM Implementation

The map is partitioned into multiple sections of the same size which are maintained by different processors as shown in Fig. 1. One of the major beneifts of this strategy is the distribution of required memory across multiple processors. This allows both for the handling of far larger maps than serial implementations and for a faster execution, as multiple processors are used [2]. In our implementation, we use partitioning with a master controlling multiple slaves. The master process is as follows: firstable, one input data is selected at random and broacast it to all its slaves, then the process waits until all slaves return their local winner, i.e. the coordinate and the distance of the node that matches best with the input data. After this, the global winner is identified and all slaves are notified of that. These steps are repeated until no input data remain, as shown in Fig. 2. At this point master process checks whether the map is stable, to get finish, otherwise the process returns at the beggining (this part is not depicted in Fig. 2). On the other hand, the slave process begins waiting for the selected input data in the master process. Then a local winner is determined and the master process is notified of that. The process waits until the master broadcast the global winner. Once the global winner is known the process proceeds to update those nodes that belongs to its map segment and to the global winner neighborhood. When all elements in the input dataset have been processed the slave process checks whether its map segment is stable and sends a message to the master indicating the state of the map segment (stable or not).

As depicted in Fig. 2 three major synchronization points occur in this implementation. The data transfers are very small, as only the selected input data is broadcast; the data packets for computing the *winner* consist of a cartesian coordinate and a floating point number signifying the error at that coordinate. The data packets exchanged at the synchronization points are very small, consisting of a spatial coordinate and an error value at most. Communications points are depicted by dashed boxes in Fig. 2

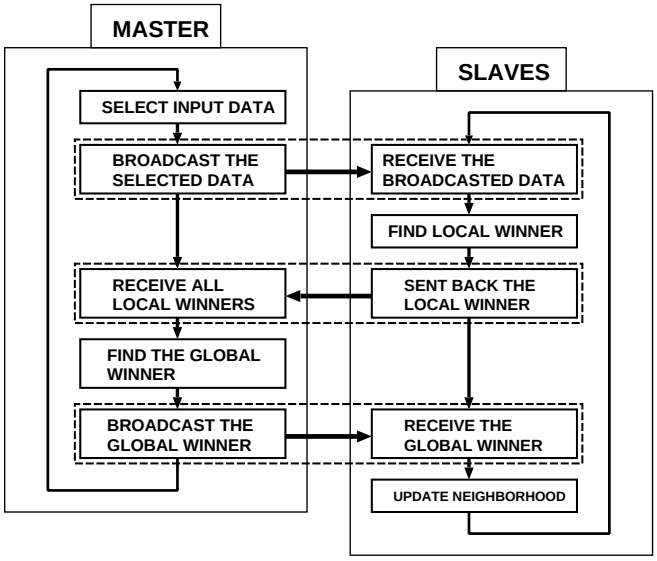

Fig. 2. Master-Slave communication scheme where all slaves have their own map segment. Searching for winners and updating the neighborhood are task that all slaves execute concurrently.

V. Beowulf Cluster

Beowulf is a multicomputer architecture which can be used for parallel computations [7]. It is a system which usually consists of one server node, and one or more client nodes conected together via Ethernet or some other network. It is a system built using commodity hardware components, like any PC capable of running Linux, standard Ethernets adapters, and switches. It does not contain any custom hardware components and is trivially reproducible. Beowulf also uses commodity software like the Linux operating system, Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). The server node controls the whole cluster and serves files to the client nodes. In most cases client nodes do not have keyboards or monitors, and accesed only via remote login or possibly serial terminal. Beowulf nodes can be thought as a CPU + memory package which can be plugged in to the cluster, just like a CPU or memory module can be plugged into a motherboard.

VI. Experiment Design

In order for measuring the speed-up gained by using a parallel implementation of the SOM algorithm we have compared the execution time of both a secuential and a parallel implementations. The experiment was conducted on a constant text corpus and several sizes of the map. Table I shows the execution time for training several maps. The number of processor used in the parallel implementation is constant (four processors) because the main problem with the SOM algorithm is the increase in computation time resulting from increasing the number of nodes used in the map [4].

VII. Discussion

Table I and Figure 3 summarize the execution time for training several maps of different sizes. The evalutation was conducted using the same input dataset for both implementations,

TABLE I
Execution times of both the Secuential and Parallel SOM Implementations using differents map sizes

NUMBER OF NODES	SECUENTIAL EXECUTION TIME	PARALLEL EXECUTION TIME
100	15’14s	13’06s
256	37’46s	16’51s
400	59’19s	21’42s
900	2h1’0s	41’32s
2500	6h9’51s	1h50’07s

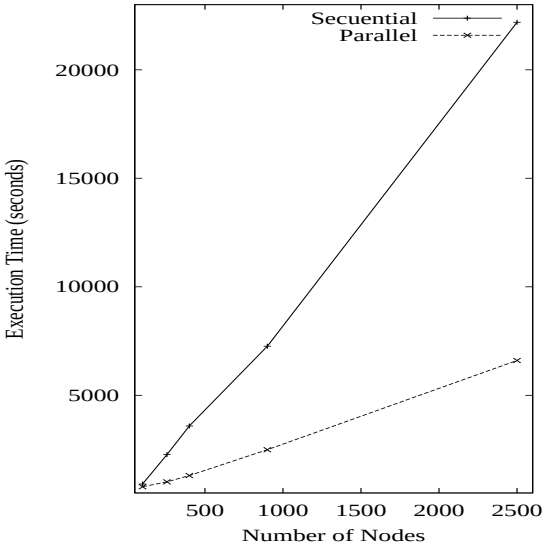

Fig. 3. Performance of both Secuential and Parallel SOM implementatios. The speed-up gained is far larger when the number of nodes is large

Secuential and Parallel. The number of machines used in the Beowulf Cluster was the same for training the differents settings of the map. The main reason of this is the increase in computation time of the SOM algorithm resulting from increasing the number of nodes. Table I shows us that using a parallel implementation of the SOM algorithm is not advisible when the size of the map is small. This is because the speed-up gained using the Beowulf Cluster is not significant. Also, if there are many computers in the cluster, the execution time could be affected due to the increase in communication packets among master and slaves machines. On the other hand, figure 3 suggest that the cluster could be used in a propper manner for training very large maps which are not unusual in practice. Text organization is an example of applications where the required number of nodes could be too high since the number of documents to organize can exceed several thousands.

VIII. Conclusions

We have described how a parallel SOM implementation, which make a partition of the map and distribute the map segments among the slaves, can improve the execution time for

training large maps. The Beowulf Cluster we use was conformed by homogeneous computers, i.e. with identical hardware specifications. The leading reason we decided to do this is that we avoid the bottleneck due to the lower computer. Using the cluster with homogeneous computers limits the potential of the Beowulf Cluster. For avoiding such a constrain we can modify the way the map segments are distributed across multiple slaves. In this proposal, the map is no longer managed as a matrix of nodes, instead the map is represented as a linear array [2]. In this way, the map is divided in segments of different sizes which are distributed between all machines in the cluster. Machines having a better performance will maintain a larger map than those machines with lower performance. This new distribution scheme for map segments requires more time for analizing the hardware specifications for all machines in the cluster. This is why this strategy should be used when dealing with large maps. Another prominent work is to determine the optimal number of machines required by the cluster for training some given maps.

REFERENCES

[1] Teuvo Kohonen, *Self-Organizing Maps*, Springer series in information sciences; 30, 3th edition, 2001.

[2] Philipp Tomsich, Andreas Rauber, and Dieter Merkl, "Optimizing the par som neural network implementation for data mining wiht distributed memory systems and cluster computing," 2001.

[3] Teuvo Kohonen, Samuel Kaski, Krista Lagus, Jarkko Salojarvi, Jukka Honkela, Vesa Paatero, and Antti Saarela, "Self organization of a massive document collection," 2000.

[4] Paul Frantz, "Massively parallel self-organizing feature maps: Progress report," 1995.

[5] Lai-Wan CHAN, Man-Wai CHAU, and Wing-Chung CHUNG, "Globalor: A parallel implementation of the self-organizing map," 1995.

[6] Tomas Nordström, "Designing parallel computers for self-organizing maps," 1992.

[7] Jacek Radajewski and Douglas Eadline, "Beowulf-howto," 1998.