$_{par}$SOM: A Parallel Implementation of the Self-Organizing Map Exploiting Cache Effects: Making the SOM Fit for Interactive High-Performance Data Analysis

Andreas Rauber, Philipp Tomsich and Dieter Merkl
Institute of Software Technology, Vienna University of Technology
Favoritenstraße 9-11/188, A–1040 Wien, Austria
{andi,phil,dieter}@ifs.tuwien.ac.at

Abstract

A large number of applications has shown, that the self-organizing map is a prominent unsupervised neural network model for high-dimensional data analysis. However, the high execution times required to train the map put a limit to its use in many application domains, where either very large datasets are encountered and/or interactive response times are required.

In order to provide interactive response times during data analysis we developed the $_{par}$SOM, a software-based parallel implementation of the self-organizing map Parallel execution reduces the training time to a large degree, with an even higher speedup obtained by using the resulting cache effects. We demonstrate the scalability of the $_{par}$SOM system and the speed-up obtained on different architectures using an example from high-dimensional text data classification.

1 Introduction

With the massive advance of huge, high-dimensional data collections, tools for exploring these collections to extract hidden information become increasingly important. The self-organizing map (SOM) [5] is a prominent unsupervised neural network model providing a topology-preserving mapping from a high-dimensional input space onto a two-dimensional map space. This capability allows an intuitive analysis and exploration of unknown data spaces, with its applications ranging from the analysis of financial data to scientific data analysis [3, 5].

Another prominent application arena are digital libraries, which are particularly challenging for the SOM, as very high dimensional feature spaces of thousands of attributes have to be dealt with, calling for high-performance computing solutions [1]. The goal is to provide a topical organization of documents on a SOM such that texts on similar topics are mapped close to each other similar to the arrangement of books in a conventional library. This organization facilitates intuitive exploration of unknown document archives. However, with the high execution times of training these very high-dimensional SOMs, interactive response times cannot be provided. This implies that documents cannot be clustered on the fly for, e.g., web-searches or for searching using iterative refinement. Thus, interactive training and mapping are highly desirable, as they open a huge field of applications in the context of data analysis in data warehouses or in the field of digital libraries, to name just two examples, where interactive response times are a must.

Existing hardware implementations [4, 5, 6] have not gained wide-spread acceptance because of the limitations they put both on the dimensionality of the data to be analyzed and on the size of the map, lacking the flexibility required by these applications. On the other hand, high-performance computing provides the technology necessary for the desired speedup by performing the computationally intensive training process in parallel on multiple computing nodes.

In this paper we present the $_{par}$SOM, a software-based parallel implementation of the SOM, using a simple asymmetric model of parallelization. Apart from the obvious speedup gained by having the training process executed in parallel for the various neural processing units we also show, that the use of cache effects allows a better-than-linear speedup for the resulting implementation. We implemented and evaluated the $_{par}$SOM on a number of platforms, ranging from off-the-shelf dual-processor Celeron systems, a 16-processor SGI Power

Challenge, to a 64-processor SGI Cray Origin 2000, using an application from the text classification domain as primary testbed.

The remainder of the paper is organized as follows. Section 2 provides a brief introduction to the architecture and the training algorithm of a self-organizing map, presenting a serial implementation focused on cache-awareness. The actual algorithm implemented in the $_{par}$SOM system is presented in Section 3, describing the master-slave model and the general effects on memory bandwidth. We furthermore provide experimental results from using the $_{par}$SOM to cluster news articles from the *Time Magazine*, with a brief evaluation of the topical clusters found in the collection outlined in Section 4. This is followed by a presentation of performance results on various architectures, discussing scalability issues and cache effects in Section 5. Conclusions and an outlook on further optimizations are presented in Section 6.

2 SOM Architecture and Training

The self-organizing map is an unsupervised neural network providing a mapping from a high-dimensional input space to a usually two-dimensional output space while preserving topological relations as faithfully as possible. The SOM consists of a set of i units arranged in a two-dimensional grid, with a weight vector $m_i \in \Re^n$ attached to each unit. Elements from the high dimensional input space, referred to as input vectors $x \in \Re^n$, are presented to the SOM and the activation of each unit for the presented input vector is calculated using an activation function. Commonly, the Euclidean distance between the weight vector of the unit and the input vector serves as the activation function.

In the next step the weight vector of the unit showing the highest activation (i.e. the smallest Euclidean distance) is selected as the 'winner' and is modified as to more closely resemble the presented input vector. Pragmatically speaking, the weight vector of the winner is moved towards the presented input signal by a certain fraction of the Euclidean distance as indicated by a time-decreasing learning rate α. Thus, this unit's activation will be even higher the next time the same input signal is presented. Furthermore, the weight vectors of units in the neighborhood of the winner as described by a time-decreasing neighborhood function h_{ci} are modified accordingly, yet to a less strong amount as compared to the winner. This learning procedure finally leads to a topologically ordered mapping of the presented input signals. Similar input data is mapped onto neighboring regions on the map [5].

A serial implementation of the SOM iterates through the training algorithm. This involves scanning all units to find the winner and then modifying all units to reflect the change. As the map will usually be implemented as an array of weight-vectors, this array is traversed twice. From this, it should be clear that a major bottleneck in a software-based implementation of a self-organizing map lies with the memory bandwidth. During every iteration of the training algorithm every unit within the map is touched twice. First, during the search for the best matching unit within the map and then again, when the weights are adjusted. Due to the fact, that we are using high-dimensional inputs with multiple thousands of inputs, it is not uncommon that a rather small map requires in excess of 5 MBs. A medium-sized map may easily occupy tens of megabytes. This is beyond the capacity of most cache memories and therefore incurs large overheads in its memory accesses. The cache thrashing can be lessened by using a technique known as *loop inversion*. Instead of traversing the unit-array from its beginning to its end when traversing it for the second time (i.e., when adjusting the weights), the array is traversed backwards to exploit the L2 cache: The last part of the map is contained in the cache, when the weight adjustment loop starts to execute. This loop modifies the map starting from the end, which ascertains that a minimum of cache replacement will occur.

Another major bottleneck results from the huge number of floating point operations required. Training a 10x15 map of 4000 features in 20000 iterations requires in excess of 72 billion arithmetic floating point instructions and 6 million floating point comparisons. In order to provide high performance in such applications, multiple processors need to be used in parallel. This becomes a viable alternative, as the distance calculation and the adaption of the weights can be executed independently for each unit. These independent operations constitute the major part of the runtime, calling for a parallel implementation.

Combining these two approaches offers the possibility to provide interactive classification for even very-high dimensional input data, with the added benefits of the topology preserving capability of the SOM, allowing intuitive data exploration.

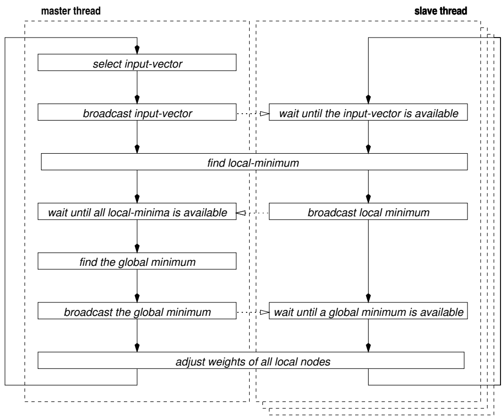

Figure 1: A flowchart of the computation using an asymmetric model of parallelization.

3 The $_{par}$SOM implementation

Two different strategies for the parallelization of SOMs are available:

- **Vectorization.** It is possible to use multiple processors or SIMD (single instruction multiple data) techniques to operate on the vector-components in parallel. Modern microprocessors are sometimes fitted with SIMD nodes for the benefit of multimedia applications. Although SIMD units cut down the pure processing time, the load on the memory bus is increased. Special compiler support is necessary for this optimization.

- **Partitioning.** The map is partitioned into multiple sections, which are maintained by different processors. One of the major benefit of this strategy is the distribution of required memory across multiple units. This allows both for the handling of far larger maps than serial implementations and for a faster execution, as multiple processors are used.

For our current implementation, we use partitioning, with one master controling multiple slaves. As depicted in Figure 1 three major synchronization points occur in this implementation. The approach chosen in our work is based on a communication-oriented computation and synchronization model. It uses a number of information and condition broadcasts to synchronize its operations. In addition, we view a multi-dimensional self-organizing map as a collection of units with associated coordinates. This allows us to distribute the units freely between the threads.

Any subsets of the units can be easily represented in a linear array. By segmenting the SOM into multiple regions, we can effectively distribute the memory usage across multiple processing nodes. Since training the SOM requires the entire map segment to be operated on, the distribution of the map across multiple processing nodes with independent memories—e.g., on computers using the non-uniform memory access (NUMA) architecture—reduces the pressure on the memory hierarchy evading bus contention, and makes it possible to fit the map segments entirely into L2 caches on the respective nodes, which provided an impressive performance boost in our experiments.

A master thread selects an input vector and broadcasts it to all its slave threads. All threads search for their local minima, i.e. best-matching units, concurrently. These best-matches are sent back to the master, which determines a global match. All slaves are notified of the global match and they modify their units relative to the location of it independently. The data exchanged at the respective synchronization points is as follows:

- **input vector:** Offset of the input vector within the input file.
- **local minimum:** 2-dimensional coordinate and a floating point number (error).
- **global minimum available.** 2-dimensional coordinate and a floating point number (error at the best matching unit).

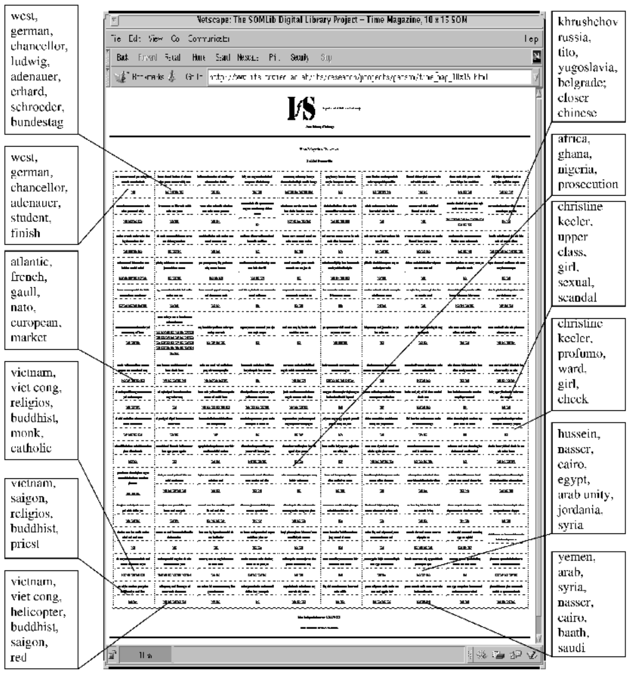

Figure 2: A map trained with 420 articles from the *TIME Magazine*.

These data packets exchanged at the synchronization points are very small, which reduces the necessary bandwidth. This algorithm should scale very well to any number of processors in theory, as the search for the local minima and the training process account for almost the entire execution time of the program. Remember, that both finding the local minima and training involves operations on arrays of large vectors of floating point numbers. For this reason, the additional work of the master thread is negligible and does not affect the overall performance. However, for small maps and very fast processors, scalability problems may arise due to the synchronization overheads.

4 SOM Clustering Evaluation

For the following experiments we use a collection of 420 articles from the *TIME Magazine* from the 1960's as a sample document repository. The articles are parsed to create a word histogram representation of the documents using the classical $tf \times idf$, i.e. term frequency times inverse document frequency weighting scheme according to the vector space model of information retrieval [9]. We thus obtain 4012-dimensional input vectors, which were further used for training SOMs of 10×15 units. The memory required to store this map exceeds 4.5MBs.

The resulting SOM is depicted in Figure 2. As a closer look at the resulting mapping reveals, the SOM succeeded in producing a topology preserving mapping from the 4012-dimensional input space onto the 2-dimensional map space, clustering the articles by topic. This can be easily demonstrated using the labels automatically extracted from the trained SOM by applying a variation of the *LabelSOM* technique [7].

For example, we find a number of articles covering the war in Vietnam in the lower left corner of the map. Another rather large cluster in the lower right corner of the map is devoted to articles on the situation in the Middle East, covering Nasser's attempts to create an arab union. Due to space considerations, we cannot present the complete map in full detail. However, the map is available at `http://www.ifs.tuwien.ac.at/ifs/research/projects/parsom/` for interactive exploration. For a closer analysis of the application of SOMs in the field of information retrieval, refer to [8].

The resulting representation of documents due to its topical organization offers itself to interactive exploration. However, to allow its application in an information retrieval or data mining setting, providing interactive response times is crucial, calling for a parallel processing of the SOM training process.

	SGI PowerChallenge			SGI Origin 2000			Dual Celeron PC		
	time			time			time		
threads	elapsed	relative	speedup	elapsed	relative	speedup	elapsed	relative	speedup
1	1555.591	1.0000	1.0000	461.940	1.0000	1.0000	1521.900	1.0000	1.0000
2	533.265	0.3428	2.9171	209.790	0.4541	2.2021	1037.850	0.6819	1.4664
3	231.964	0.1491	6.7069	150.223	0.3252	3.0750			
4	180.035	0.1157	8.6430	117.100	0.2534	3.9448			
5	128.025	0.0886	11.3122	114.930	0.2488	4.0178			
6	117.228	0.0753	13.2802	102.134	0.2211	4.5216			
7	96.632	0.0621	16.1030	90.817	0.1966	5.0854			
8	91.481	0.0588	17.0068	83.240	0.1802	5.5493			
9	83.333	0.0535	18.6915						
10	80.993	0.0520	19.2307						
11	76.701	0.0493	20.2839						
12	70.390	0.0452	22.1238						

Table 1: Execution times in seconds and speedups on multiprocessor systems

5 $_{par}$SOM Performance Evaluation

Table 1 summarizes the execution time for training the map using $_{par}$SOM. We used POSIX threads, a shared address space and user-level spinlocks in our implementation. The test were conducted on a number of multi-processor machines with dissimilar memory architectures:

- **Dual Celeron System.** Two Intel Celeron processors using a shared memory bus with a total throughput of 264MB/s. Each processor uses a 256KB level 2 cache.
- **SGI PowerChallenge XL.** A 16-processor system with a shared-bus. Each processor has a 2MB level 2 cache available and the total throughput to the main memory is 1.2GB/s.
- **SGI Cray Origin 2000.** A 64-processor system which adheres to a ccNUMA architecture, which implies that memory access times are dependent of the physical location of the memory and the accessing processor. The system uses 32 SMP node boards with 4MB of cache memory each.

The dual-processor Celeron system is a commodity personal computer. The small cache sizes of the processors require large amounts of memory transfers. Due to the limited memory bandwidth, not even a linear speedup is achievable in this configuration. The only apparent cause for the bad scalability can be attributed to bus contention. On this system, inverting the training loop does not cause a significant performance gain.

An entirely different performance characteristic was obtained on our principal development platform. On the SGI PowerChallenge, which uses a shared memory architecture, a 22-fold increase in performance could be achieved with only 12 processors. Although this may seem paradox, it should be expected. The performance increases very quickly for the first few threads, until the amount of cache memory exceeds the size of the map (compare Figure 3). After that, the performance increase continues in a linear fashion. We provided time measurements up to 12 processors only, as the machine was not available for dedicated testing. The scalability on this system remains unaffected by the available memory bandwidth: the entire SOM is distributed through the cache memories of the various processor, effectively avoiding bus contention. Interestingly, loop inversion leads to a large increase in performance for the uniprocessor version on this machine. This optimization resulted in a 22% increase in performance.

The current implementation scales to 8 processors and achieves a 5.5-fold performance increase, while still offering room for improvement due to our dependence on POSIX threads and how these are implemented in IRIX 6.5. We are currently working on a MPI-based implementation to create an optimal version for this architecture. Overall, our results compare favourably with the performance reported in [2], both in terms of spead-up and execution time.

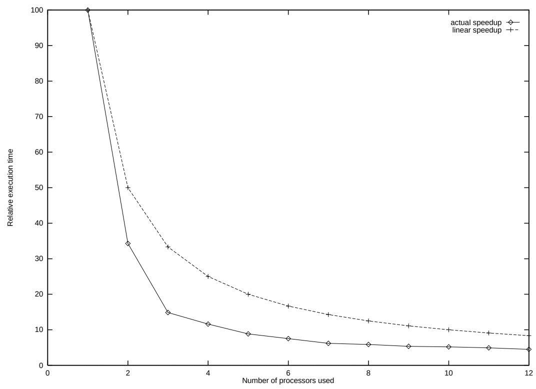

Figure 3: Performance on a SGI PowerChallenge providing a better than linear speedup.

6 Conclusions and Future Work

Our experiments with parallel, software-based implementations of self-organizing maps for text classification have shown that interactive response times—even for high dimensional feature spaces—can be achieved using supercomputing resources. The used algorithms do not only scale well, but offer an additional performance advantage due to cache effects. Parallelism effectively becomes a method to overcome the memory latencies in self-organizing neural networks.

We are currently adding MPI support. Dynamic load-balancing becomes a necessity, if non-dedicated computing resources are used, as the background load may severely affect the performance of the system. In such environments, a dynamic redistribution of the map between nodes becomes necessary to provide peak performance. Support for SIMD extensions to common microprocessors and vector computers will be also be added in future versions.

References

[1] P. Adam, H. Essafi, M.-P. Gayrad, and M. Pic. High-performance programming support for multimedia document database management. In *In: Proc. of the Europ. Conf. on High Performance Computing and Networking (HPCN99)*, pages 1211–1214, Amsterdam, Netherlands, 1999.

[2] Y. Boniface, F. Alexandre, and S. Vialle. A bridge between two paradigms for parallelism: Neural networks and general purpose mimd computers. In *Intl. Joint Conf. on Neural Networks (IJCNN99)*, Washington, DC, 1999.

[3] G. DeBoeck and T. Kohonen. *Visual Explorations in Finance.* Springer Verlag, Berlin, Germany, 1998.

[4] D. Hammerstrom. A VLSI architecture for high-performance, low-cost, on-chip learning. In *Intl. Joint Conf. on Neural Networks (IJCNN99)*, Washington, DC, 1999.

[5] T. Kohonen. *Self-Organizing Maps.* Springer Verlag, Berlin, Germany, 1995.

[6] J. Liu and M. Brooke. A fully parallel learning neural network chip for real-time control. In *Intl. Joint Conf. on Neural Networks (IJCNN99)*, Washington, DC, 1999.

[7] A. Rauber. LabelSOM: On the labeling of self-organizing maps. In *Intl. Joint Conf. on Neural Networks (IJCNN99)*, Washington, DC, 1999.

[8] A. Rauber and D. Merkl. Using self-organizing maps to organize document collections and to characterize subject matters: How to make a map tell the news of the world. In *Proc. 10th Int'l Conf. on Database and Expert Systems Applications (DEXA99)*, Florence, Italy, 1999.

[9] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, Reading, MA, 1989.

A Parallel Implementation of the Self-Organizing Map Exploiting Cache Effects:
Making the SOM Fit for Interactive High-Performance Data Analysis Andreas Rauber, Philipp Tomsich and Dieter Merkl
In: Proceedings of the International Joint Conference on Neural Networks 2000 (IJCNN'2000), 24. - 27. 7. 2000, Como, Italy.