Efficient Computation of SOM for Outage Database*

Petr Gajdoš[1], Michal Krátký[1], David Bednář[1], Radim Bača[1], Radomír Goňo[2], and Jiří Walder[1]

[1] Department of Computer Science
[2] Department of Electrical Power Engineering
FEECS, VŠB – Technical University of Ostrava,
17. Listopadu 15/2172,
708 33 Ostrava-Poruba, Czech Republic
{petr.gajdos, michal.kratky, david.bednar, radim.baca, radomir.gono, jiri.walder}@vsb.cz

Abstract. This paper describes a utilization of the Self Organizing Map (SOM) method for the analysis of power outage data. SOM, to be already used in many fields, is based on the Kohonen self-organizing neural network and it is known to capture underlying concepts. We apply this method for a unified database of power outages to be collected for several years in the Czech Republic. The most significant attributes are selected from the database and records are used for the training of the SOM. We utilize our previously introduced application EDAS (Electrical Data Analysis using SOM) for the visualization, understanding, and analysis of the trained SOM. Because of performance issues in the previous introduced approaches, we implement our SOM on GPU environment and compare this method with previous solutions in this article.

Key words: power networks, outage data, Self Organizing Map (SOM), GPU

1 Introduction

Databases of power outages have been collected in the Czech Republic for several years [7, 8]. The common framework which enables us to combine these heterogeneous databases into the one unified database has been introduced in [15, 14]. Next we analyze this data and present the underlying knowledge in such a way that can be easily understood by domain experts.

Clustering is a common technique for data analysis that can be used to reveal structures and to identify groupings on different attributes of the input data.

Based on a similarity or dissimilarity measurement, clustering consists of partitioning a data set into subsets or clusters, so that the data in each cluster have high similarity in comparison to other, however they are very dissimilar to data in other clusters.

* This work was supported by the Czech Science Foundation (No. GA102/09/1842) and by the Ministry of Education, Youth and Sports of the Czech Republic (SGS, No. SP2011/193).

Competitive neural networks also have been applied to data clustering. Self Organizing Map (SOM) [11] is a neural network algorithm based on unsupervised learning. It has already been successfully applied in various engineering applications like pattern recognition, image analysis process monitoring, and fault diagnosis [13, 1].

In this paper, we describe an application of the SOM method [12] for the power outage database. We select the most significant attributes of the database and use SOM for the clustering. After the neural network is trained we identify clusters of similar records in the map by means of the SOM density U-matrix calculation [25]. This approach has been depicted in [5, 4]. The proposed application is called EDAS (Electrical Data Analysis using SOM). In this articles, we show that it is possible to improve the computation time using GPU (Graphics Processing Unit). In this way we reach high parallelism capability and improve total time for delivering of results.

In Section 2, we describe the Kohonen maps utilized in this article. In Section 3, outage database is depicted. In Section 4, we briefly describe some preliminaries of using SOM to analysis the outage data. Section 5 includes a description of CUDA techniques for the SOM computation and results in comparison with common methods for the computation. We conclude with summarization of our work and the suggestions for the future.

2 Neural Networks

For the insight on how the method works, we will state the basic facts of the (Kohonen) neural network [11] which has several beneficial features useful for data mining. It implements a dimensionality-reduction mapping of trained data.

In the cerebral cortex, the basic element is a *neuron*, with the number of inputs (dendrites) and one output (axon). The output can be divided and connected to other dendrites through synaptic connections. Based on this biological description, a formal neuron – a structural member of the neural network – was defined. The formal neuron is shown in Figure 1.

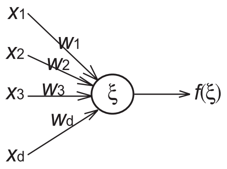


Fig. 1. Formal neuron

Where, $\mathbf{x}$ =[$x_1, \ldots, x_d$] is the input vector of dimension d with $x_1, \ldots, x_d$ scalar inputs which are mapped to the set or subset of input data attributes. The weight vector $\mathbf{w} = [w_1, \ldots, w_d]$ is assigned to the connection between the input and neuron. The ξ is the potential and $y = f(\xi)$ is the neuron's output obtained from the potential by an activation function f. The potential ξ is usually calculated from $\mathbf{w}$ and $\mathbf{x}$ vectors as the linear combination:

$$\xi = \sum_{i=1}^{d} w_i x_i.$$

The basic idea of SOM is based on the human brain which uses an internal 2D or 3D representation of information. We can imagine the input data to be transformed to vectors recorded in a neural network and most neurons in the cortex are organized in 2D. Only the adjacent neurons are interconnected. This is reflected in the SOM structure where the neurons are usually organized into a 2D rectangular or hexagonal grid. We see an example in Figure 2(a).

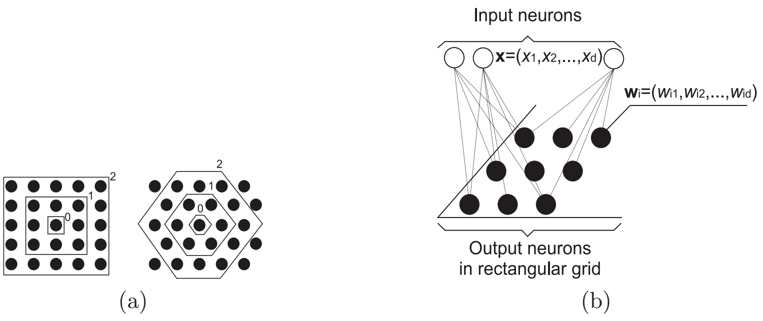


Fig. 2. (a) Rectangular and hexagonal grid latices with depicted neighborhoods (b) Kohonen network structure

The neural network is trained by a training input set (set of input vectors). Each vector of the size d is equal to the number of inputs in an input space (dimension of the input space). Every input is connected with each neuron in the grid (see Figure 2(b)). The increasing number of neurons the improving coverage of the input space. However, the computation time is increased as well.

2.1 SOM Learning Algorithm

The basic SOM algorithm is iterative. Each neuron n_i has a d-dimensional weight vector $\mathbf{w}_i = [w_{i1}, \ldots, w_{id}]$. The network weights are preset to a random or pre-calculated value. In each training step, a sample data vector $\mathbf{x}$ is randomly chosen from the training set. Distances between $\mathbf{x}$ and all the weights are computed. The best-matching unit (BMU), denoted by b, is the map neuron with the weights closest to $\mathbf{x}$:

$$\|\mathbf{x} - \mathbf{w}_b\| = \min_i \{\|\mathbf{x} - \mathbf{w}_i\|\}$$

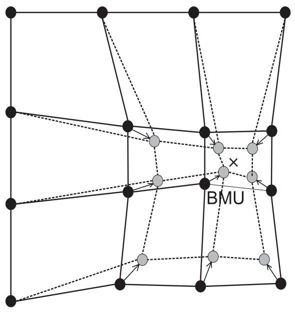


Fig. 3. Updating the BMU and its neighbors towards the input vector marked by $\times$

Next, the weight vectors are updated. The BMU and its topological neighbors are moved closer to the input vector in the input space (see Figure 3). The rule for update of the weight vector of i-th neuron is:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)h_{bi}(t)\left[\mathbf{x} - \mathbf{w}_i(t)\right] \tag{1}$$

where t denotes time, $\alpha(t)$ is the learning rate and $h_{bi}(t)$ is the neighborhood kernel centered of the i-th neuron. The kernel can be for example Gaussian:

$$h_{bi}(t) = e^{-\frac{\|\mathbf{r}_b - \mathbf{r}_i\|^2}{2\sigma^2(t)}}$$

where $\mathbf{r}_b$ and $\mathbf{r}_i$ are the positions of b and i neurons of the SOM grid and $\sigma(t)$ is the neighborhood radius. Both $\alpha(t)$ and $\sigma(t)$ decrease monotonically with the time t. These parameters determine the speed of the weight adaptation. When the weights are high the moves of the weights are also high and they are applied to a larger area. With increasing time the moves are smaller and are applied to a smaller area (we refer to Equation (1)). Because of the neighborhood relation, the similar weights are pulled to the same direction in the flexible SOM network. The similar weight regions form areas with the similar or very close values. We call these areas as clusters.

So far a simple principle of the incremental and iterative algorithms has been taken into account. Although the current evolution of modern hardware significantly increases a performance of the algorithm, a truly acceleration of learning algorithms consists in batch and parallel approaches.

Several different implementations of parallel SOM have already been presented in [18], [17], [21] and [29] and there are also studies on how the computer architecture could be modified in order to support highly parallel calculations especially for SOM training [19]. These approaches focus mainly on time efficiency issues and how the mathematical operations can be distributed on different machines to speed up the self-organization process. Different approach was presented by I. Valova [27], [26], where the amount of parallelism is not determined by the number of available hardware resources but rather by the size of the input pattern. More convenient approach for a completely parallel SOM has been presented by Weigang [28]. Here in fact, the network processes the whole input space at once, but the algorithm still uses competitive weight updating (the best-matching criterion) and thus only modifies one single node at a given time. Another approach, known as batch training, consists of updating parameters for the neurons only after all training patterns have been presented to the network, allowing the use of advanced optimization algorithms, i.e. gradient descent.

Finally, all benefits of parallel approaches can be multiplied due to utilization of modern hardware such as GPUs (Graphics Processor Units). In Section 5, we show an implementation utilizing the nVIDIA CUDA technology [20].

3 Reliability Calculations and Outage Database

Research on power system reliability first appeared in the 1940s in the U.S.A., and later in the USSR and Great Britain. Since the 1950s, reliability research has carried on in all developed countries.

Component failure rates tend to vary with a component work life [16, 3]. Many parameters in the field of reliability vary for a specific component and the condition in which a component works. These random variables are represented by probability distribution functions [6].

Since, in this paper, we apply SOM to the analysis of outage data in the Czech and Slovak Republics, we should introduce a background of outage monitoring in these countries. In the former Czechoslovakia, discussions on power system reliability dated back to the 1960s. The turning point for reliability monitoring was in 1974: Regulations 2/74 for electric power systems CEZ[1] and SEP[2] were released [23]. These regulations unified failures, outages and damaged equipment monitoring options for all distribution companies in Czechoslovakia. Since 1975, exclusive outage databases have been on the rise.

This database is a very valuable baseline for reliability computation. Unfortunately, database building has ceased since 1990 because of political and social changes. Separate distribution companies have introduced their own systems for reliability monitoring since the 1990s. A complete database has not been built henceforth.

The expert group, CIRED Czech[3], has introduced a discussion on reliability issues. The first calls for integration of particular outage databases were already claimed at the first meeting of this group in 1997. In 1999, distributors opted for unified monitoring of global reliability indices and the reliability of selected pieces of equipment. Data for the reliability computation is centrally processed and analysed at the Technical University of Ostrava[4]. This data has been handled and processed since the year 2000.

[1] Czech energy company
[2] Slovak energy company
[3] `http://www.ckcired.cz/`
[4] `http://www.vsb.cz/`

The reliability computation of the whole system is executed on the basis of components reliability that are included in the system [7]. In [15, 14], we have introduced a framework for storage and querying outage data [7, 8]. Databases of various distributors are transformed into the common relation scheme (see Table 1). We see that some attributes are foreign keys of codebooks: these codebooks are labeled with an order number.

Table 1. The Outage relation scheme

Order	**Attribute**	**Data Type**	**Foreign Key/ Codebook Order**
1	distributor	`NUMBER`	yes/01
2	event_order	`CHAR`	
3	event_type	`NUMBER`	yes/02
4	distribution_point	`NUMBER`	yes/03
5	area	`CHAR`	
6	network_type	`NUMBER`	yes/05
7	network_voltage	`NUMBER`	yes/04
8	equipment_voltage	`NUMBER`	yes/04
9	original_event_order	`CHAR`	
10	event_cause	`NUMBER`	yes/06
11	equipment_type	`NUMBER`	yes/07
12	damaged_equipment	`NUMBER`	yes/08
13	damaged equipment_type	`NUMBER`	yes/10
14	amount	`NUMBER`	
15	short_type	`NUMBER`	yes/09
16	producer	`NUMBER`	yes/11
17	production_date	`DATE`	
18	T0	`DATE`	
19	T1	`DATE`	
20	T2	`DATE`	
21	T3	`DATE`	
22	T4	`DATE`	
23	TZ	`DATE`	
24	P1	`NUMBER`	
25	P2	`NUMBER`	
26	D1	`NUMBER`	
27	D2	`NUMBER`	
28	Z1	`NUMBER`	
29	Z2	`NUMBER`	
30	LxT	`NUMBER`	
31	failure_type	`NUMBER`	yes/13

4 SOM Analysis of the Outage Database

In [5, 4], we have introduced a utilization of SOM for analysis of the outage database. In these articles, we have selected attributes which are used in the reliability computation. (see Table 2).

For analyses `equipment_voltage` and one from T_3 or T_4 are skipped since `network_voltage` depends on `equipment_voltage` and values of T_3 or T_4 are not sometimes defined, therefore, we select the defined value of these two attributes. Only records with defined values for all these attributes are considered in our analysis. Consequently, 190,561 from 309,000 records are considered in this case.

Table 2. Attributes selected for the analysis

Order	Attribute	Data Type	Foreign Key/ Codebook Order
1	distributor	NUMBER	yes/01
3	event_type	NUMBER	yes/02
7	network_voltage	NUMBER	yes/04
8	equipment_voltage	NUMBER	yes/04
10	event_cause	NUMBER	yes/06
11	equipment_type	NUMBER	yes/07
12	damaged_equipment	NUMBER	yes/08
18	T_0	DATE	–
21	T_3	DATE	–
22	T_4	DATE	–
31	failure_type	NUMBER	yes/13

Values of these attributes are different integer values. A vector distance is calculated during the SOM computation. Obviously, these different values are not appropriate for this distance computation. We need to normalize these values, therefore, each value is normalized into the $\langle 0, 1\rangle$ interval using a knowledge about the attribute domain.

We trained the SOM with the normalized outage database. Training the SOM map of the 100×100 size with 500 epochs takes approximately 11 hours[5] (only one thread is used).

Once we created the result SOM map from the input data, we used our application EDAS (Electrical Data Analysis using SOM) to display the result in a comprehensible form. We can display the following maps: U-matrix describing the network density, number of records mapped to the SOM node, component plane maps, attribute correlation scatter plot and so on (see Figures 4 and 5).

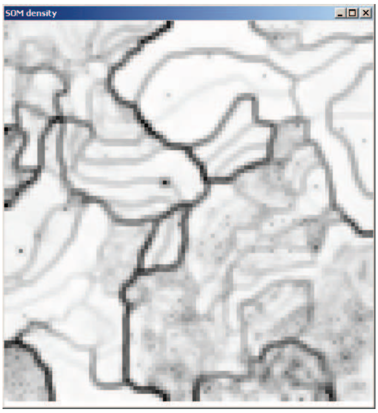
5 SOM computation using GPU

As it was already mentioned, the SOM can be updated in two ways; sequential (incremental) or batch. In the case of sequential training, the SOM is updated just after every input vector has been processed. Such training is more time-consuming in comparison with batch training where the SOM is updated after a set of input vectors (batch) has been processed and the final weight vector has been computed. Following text describes the parallel approach based on GPU.

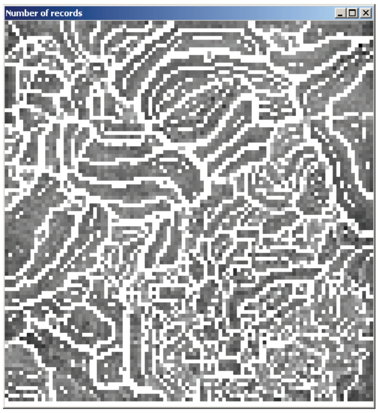
5.1 Platform Description

Modern graphics hardware plays an important role in the area of parallel computing. Graphics cards have been used to accelerate gaming and 3D graph-

[5] The experiments were executed on an AMD Opteron 865 1.8Ghz, 2.0 MB L2 cache; 2GB of DDR333; Windows 2003 Server.



(a)



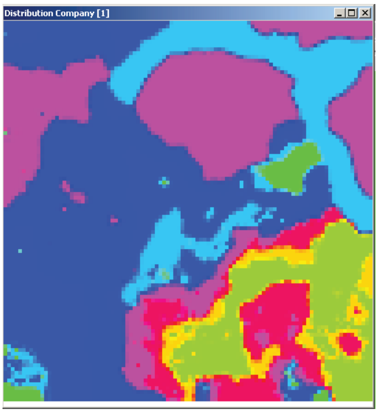
(b)

Fig. 4. (a) Density of SOM weights (b) Number of records per a SOM node

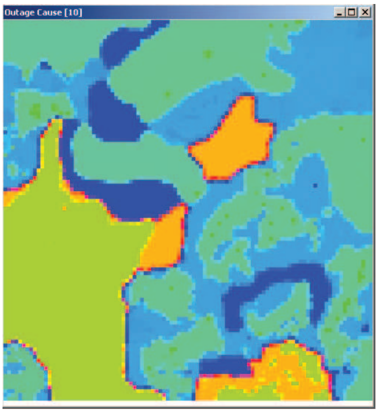
ics applications, and now, they are also used to accelerate computations with relatively distant topics, e.g. remote sensing, environmental monitoring, business forecasting, medical applications or physical simulations etc. Architecture of GPUs (Graphics Processing Unit) is suitable for vector and matrix algebra operations. That leads to the wide usage of GPUs in the area of information retrieval, data mining, image processing, data compression, etc. [20].

There are two graphics hardware leaders, vendors (ATI and nVIDIA) that prefer their solution before any other. ATI developed technology called ATI Stream and nVIDIA presented nVIDIA CUDA. Comparison of these two APIs is not a goal of this article, we refer to [20] for more information. CUDA is an acronym for Compute Unified Device Architecture and it was used in our experiments because of hardware availability. It is a general purpose parallel computing architecture that leverages the parallel compute engine in nVIDIA graphics processing units.

Each SIMD (Single Instruction Multiple Data) multiprocessor drives eight arithmetic logic units (ALU) which process the data, thus each ALU of a multiprocessor executes the same operations on different data, lying in the registers. In contrast to standard CPUs which can reschedule operations (out-of-order execution), the selected GPU is an in-order architecture. This drawback is overcome by using multiple threads as described by Hager et al. in [9]. Current general purpose CPUs with clock rates of 3 GHz outrun a single ALU of the multiprocessors with its rather slow 1.3 GHz. In case of nVidia Tesla C1060 card, the huge number of 30 multiprocessors each with 240 cores compensates this drawback. The processing is optimized for floating point calculations and a FMA (Fused Multiply Add) is four step pipelined, thus its latency is four clock cycles. Additional operations have different specifications and therefore require different numbers of clock cycles to complete. GPU producers prefer double precision in later GPUs because of their wide utilization in scientific computations. CUDA



(a)



(b)

Fig. 5. (a) SOM map for the Distribution company and (b) Outage cause attributes

Capability 1.3 and above is necessary to have double precision support. For completion, we refer to [2], [24], [22], [20] for more nVidia CUDA examples.

5.2 Parallel Learning Algorithm on GPU

From the theoretical point of view, the process of SOM update can be transformed into a sequence of matrix operations (multiplications) and thus it can be easily implemented on GPU. Of course, one have to think about dimensions of all used matrices to prevent memory leaks and hardware restrictions. The nVIDIA CUDA was used in our experiments, moreover, some of the matrix operations were performed with the usage of CUBLAS library[6], which is a well designed library for basic linear algebra. Individual critical parts which can not be interpreted as a matrix operations were implemented as separate kernel functions, e.g. finding the BMU (Best Matching Unit). Even in this cases, the principles of parallel programing and data processing were used, e.g. the principle of PR (Parallel Reduction) brings a significant time savings.

From the programming point of view, some fix data is precomputed in our application. Such data is stored in the global memory and then partially copied into the shared memory by all thread blocks to reduce the computational time during SOM update [10]. First, it is a distance matrix which holds the distances among neurons in a SOM. In Figure 6 on the left, there is an example of the SOM with rectangular topology of dimension 7×7. The maximum distance in this topology is $maxDist = (dimX - 1) + (dimY - 1) = (7 - 1) + (7 - 1) = 12$. Just for illustration, the black node in the middle represents BMU and all nodes with the same gray scale have the same distance to the BMU with respect to the rectangular topology.

[6] `http://developer.nvidia.com/cuBLAS`

A vector of learning factors F represents another fix data. In Figure 6 on the right, there are curves of learning functions for all epochs ($e = 4$). The vector F has a dimension equal to $(maxDist + 1) * e = (12 + 1) * 4 = 48$ in our example and represents a set of values $f_i(d)$, where $f_i(d)$ is the evaluated learning function (learning factors) for a given epoch i, and a distance d, where $d \in < 0, maxDist >$ and $i \in < 0, e >$.

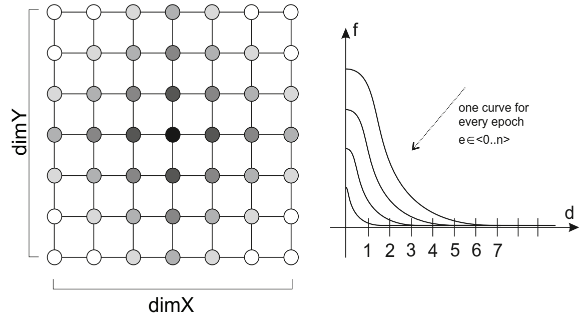


Fig. 6. Neuron distances in the SOM for a given BMU in the center of the SOM (on the left) and curves of learning functions for all epochs

5.3 GPU Performance

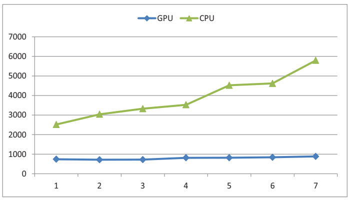
Following two experiments include a comparison between GPU and CPU implementation of SOM. The Table 3 shows detailed hardware specifications of GPU and CPU. The number of epochs for SOM was always set to 1000.

	GPU	CPU
CPU	Intel Core 2 Duo 2,2Ghz	4 x AMD Opteron 1,8 GHz
RAM	4 GB	32 GB
GPU	nVidia Tesla C1060, 4 GB	-
Threads	depends of GPU	8(CPU)

Table 3. Hardware specification

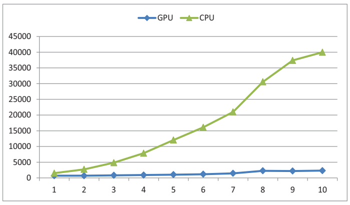
The first experiment and its results in Figure 7 shows the power of parallelism on neurons. The contribution of GPU implementation is perceptible in higher dimension of input vectors. As it can be seen in the graph, higher dimension does not have such influence on time consumption in case of the GPU utilization.

Next experiment (see Figure 8) illustrates the power of the GPU utilization in the case of different dimensions of SOM grid. The parallelism brings a significant improvement.



Experiment ID	Number of inputs	Vector dimension	SOM grid dimension X	SOM grid dimension Y	Computation time (s) GPU	Computation time (s) CPU
1	250000	40	10	10	741	2519
2	250000	50	10	10	716	3039
3	250000	60	10	10	723	3326
4	250000	70	10	10	815	3526
5	250000	80	10	10	820	4521
6	250000	90	10	10	842	4618
7	250000	100	10	10	885	5794

Fig. 7. Variable dimensions of input vectors



Experiment ID	Number of inputs	Vector dimension	SOM grid dimension X	SOM grid dimension Y	Computation time (s) GPU	Computation time (s) CPU
1	250000	4	10	10	692	1513
2	250000	4	20	20	698	2724
3	250000	4	30	30	819	4847
4	250000	4	40	40	906	7858
5	250000	4	50	50	1043	12048
6	250000	4	60	60	1176	16105
7	250000	4	70	70	1460	20991
8	250000	4	80	80	2268	30565
9	250000	4	90	90	2200	37387
10	250000	4	100	100	2308	39985

Fig. 8. Variable dimensions of SOM grid

One can predict the computation time for the real data collection. In the case of the outage data analysis, several experiments have been done. Figure 9 shows the final computation times.

Experiment ID	Number of inputs	Vector dimension	SOM grid X	SOM grid Y	SOM epochs	GPU time (s)
1	192036	9	32	32	500	741
2	192036	9	32	32	1000	716
3	192036	9	64	64	500	723
4	192036	9	64	64	1000	815

Fig. 9. Experiments on outage data collection

Current evolution of High Performance Computing (HPC) shows, that the usage of GPU can shift the limits of mathematical computation. Thus the earlier time-consuming computations can be performed on computers with common hardware configuration which is important for practical use.

6 Conclusion

In this paper, we described the SOM analysis of the outage database. We described how to prepare the data for a SOM analysis and how to create the SOM map from them. After analysis we visualized the data in the EDAS tool for a convenient work with a SOM map. Due to the performance issue, we introduced a parallel implementation using GPU in this article. The parallel implementation is up-to 10× faster then the CPU implementation. In future work, we will compare the results of SOM with other methods.

References

1. E. Alhoniemi, J. Hollmén, O. Simula, and J. Vesanto. Process monitoring and modeling using the self-organizing map. *Integr. Comput.-Aided Eng.*, 6(1):3–14, 1999.
2. M. Andrecut. Parallel gpu implementation of iterative pca algorithms. *Journal of Computational Biology*, 16(11):1593–1599, November 2009.
3. L. Bain, M. Englehardt, and M. Engelhardt. *Statistical Analysis of Reliability and Life-testing Models: Theory and Methods*. Marcel Dekker Ltd., 1991.
4. R. Bača, R. Goňo, M. Krátký, and V. Snášel. Using Kohonen Maps for a Power Outage Data Analysis. In *Proceedings of the 10th International Scientific Conference Electric Power Engineering 2009*, pages 367–372, Kouty nad Desnou, Czech Republic, 2009.
5. R. Bača, M. Krátký, and V. Snášel. Power Outage Data Analysis via SOM Neural Networks. In *Proceedings of the 5th ELNET Conference*, Ostrava, 2008.
6. W. H. Beyer. *Crc Standard Mathematical Tables*. CRC Press, Boca Raton, USA, 27th edition, 1984.
7. R. Goňo and S. Rusek. Analysis of Power Outages in the Distribution Networks. In *Proceedings of the 8th International Conference on ElectricalPower Quality and Utilisation, EPQU 2003. Cracow, Poland*. IEEE Press, 2003.
8. R. Goňo, S. Rusek, and M. Krátký. Reliability analysis of distribution networks. In *Proceedings of the 9th International Conference on Electrical Power Quality and Utilisation, EPQU 2007. Barcelona, Spain*. IEEE Press, 2007.
9. G. Hager, T. Zeiser, and G. Wellein. Data access optimizations for highly threaded multi-core cpus with multiple memory controllers. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –7, apr. 2008.
10. D. Kirk and W. mei Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. MORGAN KAUFMANN, Januar 2010.
11. T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.
12. T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, J. Honkela, V. Paatero, and A. Saarela. Self Organization of a Massive Document Collection. *IEEE Transactions on Neural Networks, Special Issue on Neural Networks for Data Mining and Knowledge Discovery*, 11(3):574–585, May 2000.
13. T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas. Engineering applications of the self-organizing map. In *Proceedings of the IEEE, Volume 84, Issue 10*, pages 1358–1384, October 1996.
14. M. Krátký, R. Goňo, and S. Rusek. A Framework for Querying and Indexing Electrical Failure Data. In *Proceedings of ELNET 2006. Ostrava, Czech Republic*, 2006.
15. M. Krátký, R. Goňo, S. Rusek, and J. Dvorský. A Framework for an Analysis of Failures Data in Electrical Power Networks. In *Proceedings of ELNET/PEA 2006. Gaborone, Botswana*. ACTA Press/IASTED, 2006.

16. N. R. Mann, R. E. Schafer, and N. D. Singpurwalla. *Methods for Statistical Analysis of Reliability and Life Data.* John Wiley & Sons, Inc., NewYork, 1974.
17. R. Mann and S. Haykin. A parallel implementation of Kohonen's feature maps on the warp systolic computer. In *Proc. IJCNN-90-WASH-DC, Int. Joint Conf. on Neural Networks*, volume II, pages 84–87, Hillsdale, NJ, 1990. Lawrence Erlbaum.
18. G. Myklebust, J. G. Solheim, and E. Steen. Wavefront implementation of self organizing maps on renns. In *International Conference on Digital Signal Processing, (Limassol, Cyprus)*, pages 268–273, 1995.
19. T. Nordström. Designing parallel computers for self organizing maps. In *Fourth Swedish Workshop on Computer System Architecture*, 1992.
20. nVIDIA. Cuda zone, August`http://www.nvidia.com/object/cuda_home_new.html`, 2010.
21. S. Openshaw and I. Turton. A parallel kohonen algorithm for the classification of large spatial datasets. *Comput. Geosci.*, 22(9):1019–1026, 1996.
22. D. Patnaik, S. P. Ponce, Y. Cao, and N. Ramakrishnan. Accelerator-oriented algorithm transformation for temporal data mining. *Network and Parallel Computing Workshops, IFIP International Conference on*, 0:93–100, 2009.
23. J. Piskač and J. Marko. Regulations for electric power system No.2 Failure statistics at electricity distribution, 1974.
24. T. Preis, P. Virnau, W. Paul, and J. J. Schneider. Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets. *New Journal of Physics*, 11(9):093024, 2009.
25. A. Ultsch and H. P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In *Proceedings Intern. Neural Networks*, pages 305–308, Paris, 1990. Kluwer Academic Press.
26. I. Valova, D. MacLean, and D. Beaton. Identification of patterns via region-growing parallel som neural network. *Machine Learning and Applications, Fourth International Conference on*, 0:853–858, 2008.
27. I. Valova, D. Szer, N. Gueorguieva, and A. Buer. A parallel growing architecture for self-organizing maps with unsupervised learning. *Neurocomput.*, 68:177–195, 2005.
28. L. Weigang and N. da Silva. A study of parallel neural networks. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 2, pages 1113 –1116, jul. 1999.
29. C.-H. Wu, R. E. Hodges, and C.-J. Wang. Parallelizing the self-organizing feature map on multiprocessor systems. *Parallel Computing*, 17(6-7):821 – 832, 1991.