# Optimalization of Parallel GNG by Neurons Assigned to Processes

Lukáš Vojáček[1(✉)], Pavla Dráždilová[2], and Jiří Dvorský[2]

[1] IT4Innovations, VŠB - Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic
lukas.vojacek@vsb.cz

[2] Department of Computer Science, FEECS, VŠB - Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava-Poruba, Czech Republic
{pavla.drazdilova,jiri.dvorsky}@vsb.cz

**Abstract.** The size, complexity and dimensionality of data collections are ever increasing from the beginning of the computer era. Clustering is used to reveal structures and to reduce large amounts of raw data. There are two main issues when clustering based on unsupervised learning, such as Growing Neural Gas (GNG) [9], is performed on vast high dimensional data collection – the fast growth of computational complexity with respect to growing data dimensionality, and the specific similarity measurement in a high-dimensional space. These two factors reduce the effectiveness of clustering algorithms in many real applications. The growth of computational complexity can be partially solved using the parallel computation facilities, such as High Performance Computing (HPC) cluster with MPI. An effective parallel implementation of GNG is discussed in this paper, while the main focus is on minimizing of interprocess communication. The achieved speed-up was better than previous approach and the results from the standard and parallel version of GNG are same.

**Keywords:** Growing neural gas · High-dimensional dataset · High performance computing · MPI

## 1 Introduction

The size and complexity of data collections are ever increasing from the beginning of the computer era, while the dimensionality of the data sets is rapidly increasing in recent years. Contemporary and especially future technologies allow us to acquire, store and process large high dimensional data collections. High dimensional data collections are commonly available in areas like medicine, biology, information retrieval, web analysis, social network analysis, image processing, financial transaction analysis and many others.

Clustering, considered the most important unsupervised learning problem, is used to reveal structures, to identify "natural" groupings of the data collections

DOI: 10.1007/978-3-319-59105-6_6

and to reduce large amounts of raw data by categorizing in smaller sets of similar items.

There are two main issues when clustering based on unsupervised learning, such as *Growing Neural Gas* (GNG) [9], is performed on vast high dimensional data collection:

1. The fast growth of computational complexity with respect to growing data dimensionality, and
2. The specific similarity measurement in a high-dimensional space, where the expected distance, computed by Euclidean metrics to the closest and to the farthest point of any given point, shrinks with growing dimensionality [2].

These two factors reduce the effectiveness of clustering algorithms on the above-mentioned high-dimensional data collections in many real applications.

The growth of computational complexity can be partially solved using the parallel computation facilities, such as *High Performance Computing* (HPC) cluster with MPI. Obviously, it is necessary to resolve technical and implementation issues specific to this computing platform. An effective parallel implementation of GNG is discussed in this paper, while the main focus is on minimizing of interprocess communication.

## 2 Artificial Neural Networks

### 2.1 Related Works

The methods based on Artificial Neural Networks (ANN) are highly computationally expensive. There are different approaches on how to improve effectivity of these methods. The one possibility is to improve computation. The authors of this paper [3] propose two optimization techniques that are aimed at an efficient implementation of the GNG algorithm internal structure. Their optimizations preserve all properties of the GNG algorithm. The first technique enhances the nearest neighbor search using a space partitioning by a grid of rectangular cells and the second technique speeds up the handling of node errors using the lazy evaluation approach.

The next possibility for how to improve effectivity methods based on ANN is parallelization. In the paper [4] the authors combine the batch variant of the GNG algorithm with the MapReduce paradigm resulting in a GNG variant suitable for processing large data sets in scalable, general cluster environments. The paper [16] is focused on the actualizations of neurons' weights in the learning phase of parallel implementation of SOM. There are two extremal update strategies. Using the first strategy, all necessary updates are done immediately after processing one input vector. The other extremal choice is used in Batch SOM – updates which are processed at the end of whole epoch and authors study update strategies between these two extremal strategies.

For parallelization is often use Graphics Processing Units (GPU). In the paper [1] authors present the results of different parallelization approaches to the

GNG clustering algorithm. They especially investigated the GPU and multi-core CPU architectures. Authors in the paper [12] explore an alternative approach for the parallelization of growing self-organizing networks, based on an algorithm variant designed to match the features of the large-scale, ne-grained parallelism of GPUs, in which multiple input signals are processed simultaneously. The paper [15] describes the implementation and analysis of a network-agnostic and convergence-invariant coarse-grain parallelization of the deep neural network (DNN) training algorithm. The coarse-grain parallelization is achieved through the exploitation of the batch-level parallelism. This strategy is independent from the support of specialized and optimized libraries. Therefore, the optimization is immediately available for accelerating the DNN training. The proposal is compatible with multi-GPU execution without altering the algorithm convergence rate.

### 2.2 Growing Neural Gas

The principle of this neural network is an undirected graph which need not be connected. Generally, there are no restrictions on the topology. The graph is generated and continuously updated by competitive Hebbian Learning [8,13]. According to the pre-set conditions, new neurons are automatically added and connections between neurons are subject to time and can be removed. GNG can be used for vector quantization by finding the code-vectors in clusters [7], clustering data streams [6], biologically influenced [14] and 3D model reconstruction [10]. GNG works by modifying the graph, where the operations are the addition and removal of neurons and edges between neurons.

To understand the functioning of GNG, it is necessary to define the algorithm. The algorithm described in our previous article [17] is based on the original algorithm [5,7], but it is modified for better continuity in the SOM algorithm. The description of the algorithm has been divided for convenience into two parts. Here is the Algorithm 1 which describes one iteration.

*Remark.* The notation used in the paper is briefly listed in Table 1.

## 3 Parallelization

In the paper [17] we have dealt with the parallelization of GNG. The following is a brief description of our parallelization algorithm.

After analysing the GNG learning algorithm we identified the one most time-consuming processor area. This part was selected as a candidate for the possible parallelization. The selected area are:

**Finding BMU** – this part of GNG learning can be significantly accelerated by dividing the GNG output layer into smaller pieces – distribution of neurons for effective parallelization. Each piece is then assigned to an individual computation process. The calculation of the Euclidean distance among the individual

**Table 1.** Notation used in the paper

| Symbol | Description |
|---|---|
| $M$ | Number of input vectors |
| $n$ | Dimension of input vectors, number of input neurons, dimension of weight vectors in GNG output layer neurons |
| $N$ | Current number of neurons in GNG output layer |
| $N_{max}$ | Maximum allowed number of neurons in GNG output layer |
| $\mathsf{n}_i$ | $i$-th input neuron, $i = 1, 2, \ldots, n$ |
| $\mathsf{N}_i$ | $i$-th output neuron, $i = 1, 2, \ldots, N$ |
| $\mathsf{e}_{ij}$ | edge between neurons $\mathsf{N}_i$ and $\mathsf{N}_j$ for some $i, j = 1, \ldots, N$, where $i \neq j$. |
| $\mathsf{E}$ | set of all edges in GNG |
| $G$ | undirected graph describing topology of GNG, $G(\{\mathsf{N}_1, \ldots, \mathsf{N}_N\}, \mathsf{E})$ |
| $t$ | Current epoch, $t = 1, 2, \ldots, T$ |
| $X$ | Set of input vectors, $X \subset \mathbb{R}^n$ |
| $\boldsymbol{x}(t)$ | Current input vector in epoch $t$, arbitrarily selected vector from set $X \boldsymbol{x}(t) \in X$, $\boldsymbol{x}(t) = (x_1, x_2, \ldots, x_n)$ |
| $\boldsymbol{w_k}(t)$ | Weight vector of neuron $\mathsf{N}_k$, $k = 1, 2, \ldots, N \boldsymbol{w_k}(t) \in \mathbb{R}^n$, $\boldsymbol{w_k}(t) = (w_{1k}, w_{2k}, \ldots, w_{nk})$ |
| $\mathsf{N}_{c_1}$ | The first Best Matching Unit ($BMU_1$), winner of learning competition |
| $\mathsf{N}_{c_2}$ | The second Best Matching Unit ($BMU_2$), the second best matching neuron in learning competition |
| $\boldsymbol{w}_{c_1}(t)$ | Weight vector of $BMU_1$ |
| $\boldsymbol{w}_{c_1}(t)$ | Weight vector of $BMU_2$ |
| $l_{c_1}$ | Learning factor of $BMU_1$ |
| $l_{nc_1}$ | Learning factor of $BMU_1$ neighbours |
| $e_i$ | Local error of output neuron $\mathsf{N}_i$, $i = 1, 2, \ldots, N$ |
| $\alpha$ | Error $e_i$ reduction factor |
| $\beta$ | Neuron error reduction factor |
| $\gamma$ | Interval of input patterns to add a new neuron |
| $a_{max}$ | Maximum edges age |
| $a_{ij}$ | Age of edge $\mathsf{e}_{ij}$ |
| $p$ | Number of processes |
| $V_i$ | Set of neurons assigned to $Process_i$ |
| $V$ | Set of in the GNG |

input vector and all the weight vectors to find BMU in a given part of the GNG output layer is the crucial point of this part of GNG learning. Each process finds its own partial BMU in its part of the GNG output layer. Each partial BMU is then compared with other BMUs obtained by other processes.

**Algorithm 1.** One iteration of the Growing Neural Gas algorithm

```
1. Find neurons BMUs neurons $\mathsf{N}_{c_1}$ and $\mathsf{N}_{c_2}$.
2. Update the local error $e_{c_1}$ of neuron $\mathsf{N}_{c_1}$
       $$e_{c_1} = e_{c_1} + \|\boldsymbol{w}_{c_1} - \boldsymbol{x}\|^2 \quad (1)$$
3. Update the weight vector $\boldsymbol{w}_{c_1}$ of neuron $\mathsf{N}_{c_1}$
       $$\boldsymbol{w}_{c_1} = \boldsymbol{w}_{c_1} + \boldsymbol{l}_{c_1}(\boldsymbol{x} - \boldsymbol{w}_{c_1}) \quad (2)$$
4. For all neurons $\mathsf{N}_k$ where exists edge $\mathsf{e}_{c_1 k}$ ($\mathsf{N}_{c_1}$ neighbourhood)
   (a) Update the weights $\boldsymbol{w}_k$ using $l_{nc_1}$ learning factor
       $$\boldsymbol{w}_k = \boldsymbol{w}_k + l_{nc_1}(\boldsymbol{x} - \boldsymbol{w_k}) \quad (3)$$
   (b) Increase age $a_{kc_1}$ of edge $\mathsf{e}_{c_1 k}$
       $$a_{kc_1} = a_{kc_1} + 1 \quad (4)$$
5. If there is no edge between neurons $\mathsf{N}_{c_1}$ and $\mathsf{N}_{c_2}$, then create such edge. If the edge
   exists, the age is set to 0.
6. If any edge has reached the age of $a_{max}$, it is removed.
7. If there is a neuron without connection to any edge, the neuron is then removed.
8. If the number of processed input vectors in the current iteration has reached the
   whole multiple of the value $\gamma$ and the maximum allowed number of output neurons
   is not reached, add a new neuron $\mathsf{N}_{N+1}$. The location and error of the new neuron
   is determined by the following rules:
   (a) Found neuron $\mathsf{N}_b$(NBE) which has the biggest error $e_b$.
   (b) Found neuron $\mathsf{N}_c$(NSE) among neighbours of neuron $\mathsf{N}_b$ and has the biggest
       error $e_c$ among these neighbours.
   (c) Create a new neuron $\mathsf{N}_{N+1}$ and the value of $w_n$ is set as:
       $$\boldsymbol{w}_{N+1} = \frac{1}{2}(\boldsymbol{w}_b + \boldsymbol{w}_c) \quad (5)$$
   (d) Creating edges between neurons $\mathsf{N}_b$ and $\mathsf{N}_{N+1}$, and also between neurons $\mathsf{N}_c$
       and $\mathsf{N}_{N+1}$.
   (e) Removed edge between neurons $\mathsf{N}_b$ and $\mathsf{N}_c$.
   (f) Reduction of error value in neurons $\mathsf{N}_b$ and $\mathsf{N}_c$ using the multiplying factor $\alpha$.
       Error for neuron $\mathsf{N}_{N+1}$ is equal to the new error of neuron $\mathsf{N}_b$.
```

Information about the BMU of the whole network is then transmitted to all the processes to perform the updates of the BMU neighbourhood.

**Updates of weights** – update weights of edges incident with $N_{c1}$ it is quickly in the event that the neighboring nodes to $N_{c1}$ are on a same process. This part can theoretically accelerate if we move adjacent nodes on a single process. Unfortunately, the transfer node for multidimensional data is very time consuming (test data have a dimension of 8000+).

A detailed description of our parallelization process is described in Fig. 1.

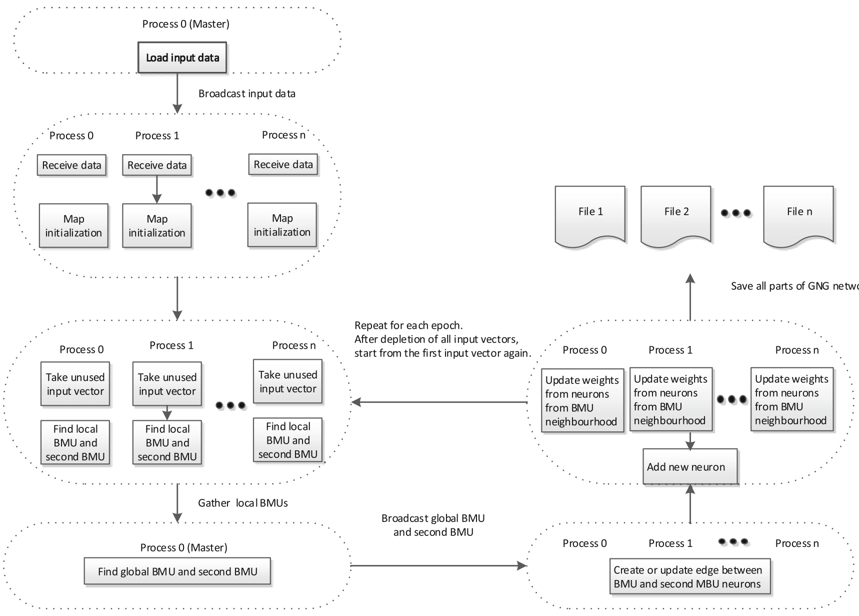

Process 0 (Master)
Load input data
Broadcast input data
Process 0
Process 1
Process n
Receive data
Map initialization
Take unused input vector
Find local BMU and second BMU
Gather local BMUs
Find global BMU and second BMU
Broadcast global BMU and second BMU
Create or update edge between BMU and second MBU neurons
Add new neuron
Update weights from neurons from BMU neighbourhood
Repeat for each epoch. After depletion of all input vectors, start from the first input vector again.
Save all parts of GNG network
File 1
File 2
File n

**Fig. 1.** Parallel algorithm

The parallelization of GNG learning was performed on an HPC cluster, using *Message Passing Interface* (MPI) technology. MPI technology is based on effective communication between processes. That means that one application can run on many cores. The application uses MPI processes which run on individual cores. The processes are able to send and receive messages and data, communicate etc. Detailed information about HPC and MPI technology is provided, for example, in [11].[1]

## 3.1 Distribution Neurons for Effective Parallelization

In the paper [17] we used Method 1 (the equal distribution of neurons), where new neurons are allocated to the process with the lowest number of neurons (see Fig. 2). The advantage of this distribution is constant workload processes. The disadvantage is increased communication between processes.

Our goal is to focus on reducing interprocessor communication by using the following methods for adding new neuron:

Method 2 The neurons are gradually added to the process, which currently does not contain a predetermined number of neurons ($\frac{N_{max}}{p}$). If one process is filled up so a new neuron is added to the next process (see Fig. 3).

[1] A specification of MPI is available on the web: http://www.mpi-forum.org/.

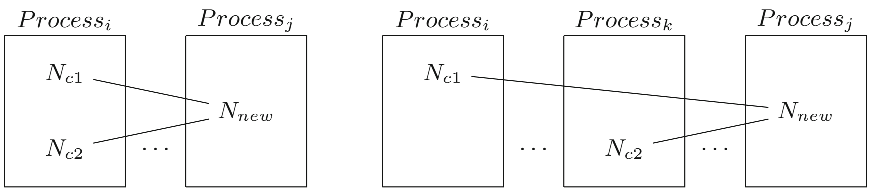


**Fig. 2.** Add the new neuron to the next free $Process_j$ by a cyclic way - Method 1.

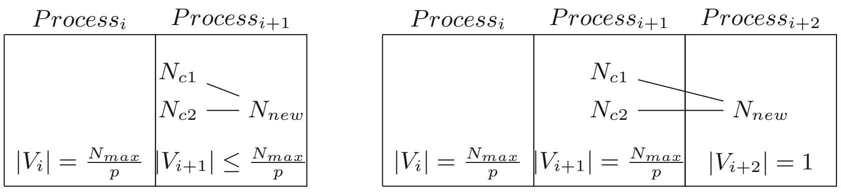
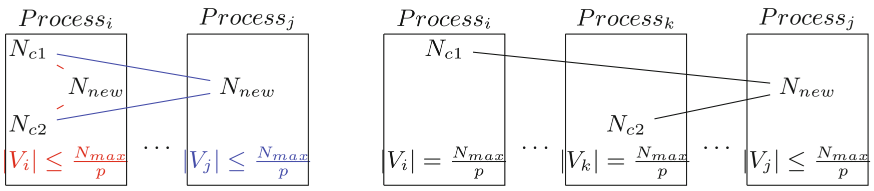


**Fig. 3.** Add the new neuron to the $Process_x$ by gradual way - Method 2.



$Process_i$
$Process_j$
$N_{c1}$
$N_{new}$
$N_{c2}$
$N_{new}$
$|V_i| \leq \frac{N_{max}}{p}$
$|V_j| \leq \frac{N_{max}}{p}$
$Process_i$
$Process_k$
$Process_j$
$N_{c1}$
$N_{c2}$
$N_{new}$
$|V_i| = \frac{N_{max}}{p}$
$|V_k| = \frac{N_{max}}{p}$
$|V_j| \leq \frac{N_{max}}{p}$

**Fig. 4.** Add the new neuron to the $Process_x$ where is $N_{c1}$ or $N_{c2}$ - Method 3.

Method 3 The start is similar to Method 1. If $|V_i| \geq 2\ \forall i \leq p$ then add $N_{new}$ to $Process_i$ where ($N_{c1} \in V_i$ and $|V_i| \leq \frac{N_{max}}{p}$) or add $N_{new}$ to $Process_k$ where ($N_{c2} \in V_k$ and $|V_k| \leq \frac{N_{max}}{p}$) or add $N_{new}$ to the next free $Process_j$ (see Fig. 4).

## 4 Experiments

### 4.1 Experimental Datasets and Hardware

One dataset was used in the experiments. The dataset was commonly used in Information Retrieval – *Medlars*.

**Medlars Dataset.** The Medlars dataset consisted of 1,033 English abstracts from a medical science[2]. The 8,567 distinct terms were extracted from the Medlars dataset. Each term represents a potential dimension in the input vector

[2] The collection can be downloaded from ftp://ftp.cs.cornell.edu/pub/smart. The total size of the dataset is approximately 1.03 MB.

space. The term's level of significance (weight) in a particular document represents a value of the component of the input vector. Finally, the input vector space has a dimension of 8,707, and 1,033 input vectors were extracted from the dataset.

**Experimental Hardware.** The experiments were performed on a Linux HPC cluster, named Anselm, with 209 computing nodes, where each node had 16 processors with 64 GB of memory. Processors in the nodes were Intel Sandy Bridge E5-2665. Compute network is InfiniBand QDR, fully non-blocking, fat-tree. Detailed information about hardware is possible to find on the web site of Anselm HPC cluster[3].

## 4.2 The Experiment

The experiment was oriented towards a comparison of the parallel GNG algorithm and parallel by modification by assignment to processes. The Medlars dataset was used for the experiment. A parallel version of the learning algorithm was run using 2, 8, 16, 24, 32 and 64 MPI processes. The records with an asterisk (*) represents the results for only one process i.e. this is the original serial learning algorithm and there is no network communication.

GNG parameters are the same for all experiments and are as follows $\gamma = 200$, $e_w = 0.05$, $e_n = 0.006$, $\alpha = 0.5$, $\beta = 0.0005$, $a_{max} = 88$, M = 2021, $\delta = 1500$. The achieved computing time is presented in Table 2.

**Table 2.** Computing time with respect to number of cores, standard GNG algorithm, dataset medlars

| Cores | Computing time [mm:ss] | | |
|---|---|---|---|
| | Method 1 | Method 2 | Method 3 |
| 1* | 35:41 | | |
| 4 | 09:36 | 10:28 | 10:41 |
| 8 | 06:59 | 06:25 | 06:52 |
| 16 | 05:53 | 06:18 | 06:37 |
| 24 | 07:29 | 05:33 | 05:35 |
| 32 | 07:42 | 07:13 | 07:17 |

As we can see from Table 2 and Fig. 5, the computing time depends on the number of used cores as well. With a growing number of processors, the computation effectiveness increases, and the computational time is sufficiently reduced (in the paper [17] we used Method 1).

[3] https://support.it4i.cz/docs/anselm-cluster-documentation/hardware-overview.

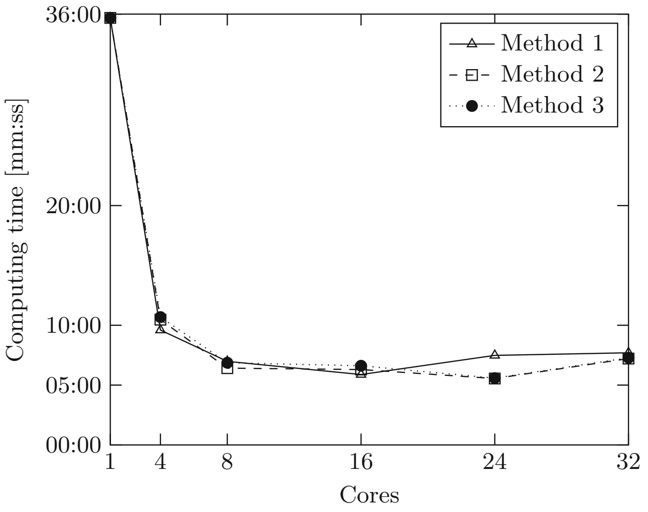


**Fig. 5.** Graph of computing time with respect to number of cores, standard GNG algorithm, dataset medlars

## 5 Conclusion

In this paper the parallel implementation of the GNG neural network algorithm is presented. The achieved speed-up was better than our previous approach. However, the effectiveness of a parallel solution is dependent on the division of the output layer. The authors introduced three different methods of neuron assignment to processes, where better acceleration for the new approaches was achieved. These approaches reached the best time but speed up is not too significant for the selected data set. Our methods are focusing on the different ways of assigning a new neuron in the GNG to processes for parallel computation.

In future work we intend to focus on the sparse date, use combinations of neural networks for improved result and improved acceleration.

**Acknowledgement.** This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project “IT4Innovations National Supercomputing Center LM2015070” and by Grant of SGS No. SP2017/100 “Parallel processing of Big Data 4”, VŠB-Technical University of Ostrava, Czech Republic.

## References

1. Adam, A., Leuoth, S., Dienelt, S., Benn, W.: Performance gain for clustering with growing neural gas using parallelization methods. In: ICEIS, vol. 2, pp. 264–269 (2010)
2. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is “nearest neighbor” meaningful?. In: Database Theory 1999, vol. 1540, pp. 217–235 (1999)

3. FišEr, D., Faigl, J., Kulich, M.: Growing neural gas efficiently. Neurocomputing **104**, 72–82 (2013)
4. Fliege, J., Benn, W.: MapReduce-based growing neural gas for scalable cluster environments. In: Perner, P. (ed.) MLDM 2016. LNCS(LNAI), vol. 9729, pp. 545–559. Springer, Cham (2016). doi:10.1007/978-3-319-41920-6_43
5. Fritzke, B.: A growing neural gas network learns topologies. In: Advances in Neural Information Processing Systems 7, pp. 625–632. MIT Press (1995)
6. Ghesmoune, M., Lebbah, M., Azzag, H.: A new growing neural gas for clustering data streams. Neural Netw. **78**, 36–50 (2016)
7. Holmström, J.: Growing neural gas experiments with GNG, GNG with utility and supervised GNG. Master's thesis, Uppsala University, 30 August 2002
8. Martinetz, T.: Competitive hebbian learning rule forms perfectly topology preserving maps. In: Gielen, S., Kappen, B. (eds.) ICANN 1993, pp. 427–434. Springer, London (1993). doi:10.1007/978-1-4471-2063-6_104
9. Martinetz, T., Schulten, K.: A "neural-gas" network learns topologies. Artif. Neural Netw. **1**, 397–402 (1991)
10. Orts-Escolano, S., Garcia-Rodriguez, J., Serra-Perez, J.A., Jimeno-Morenilla, A., Garcia-Garcia, A., Morell, V., Cazorla, M.: 3D model reconstruction using neural gas accelerated on GPU. Appl. Soft Comput. **32**, 87–100 (2015)
11. Pacheco, P.: Parallel Programming with MPI, 1st edn. Morgan Kaufmann, Burlington (1996)
12. Parigi, G., Stramieri, A., Pau, D., Piastra, M.: A Multi-signal variant for the GPU-based parallelization of growing self-organizing networks. In: Ferrier, J.-L., Bernard, A., Gusikhin, O., Madani, K. (eds.) Informatics in Control, Automation and Robotics. LNEE, vol. 283, pp. 83–100. Springer, Cham (2014). doi:10.1007/978-3-319-03500-0_6
13. Prudent, Y., Ennaji, A.: An incremental growing neural gas learns topologies. In: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, IJCNN 2005, vol. 2, pp. 1211–1216 (2005)
14. Sledge, I., Keller, J.: Growing neural gas for temporal clustering. In: Proceedings of the 19th International Conference on Pattern Recognition, ICPR 2008, pp. 1–4 (2008)
15. Tallada, M.G.: Coarse grain parallelization of deep neural networks. SIGPLAN Not. **51**(8), 1:1–1:12 (2016)
16. Vojáček, L., Dráždilová, P., Dvorský, J.: Self organizing maps with delay actualization. In: Saeed, K., Homenda, W. (eds.) CISIM 2015. LNCS, vol. 9339, pp. 154–165. Springer, Cham (2015). doi:10.1007/978-3-319-24369-6_13
17. Vojáček, L., Dvorský, J.: Growing neural gas – a parallel approach. In: Saeed, K., Chaki, R., Cortesi, A., Wierzchoń, S. (eds.) CISIM 2013. LNCS, vol. 8104, pp. 408–419. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40925-7_38