*Parallel Computing: Technology Trends*
*I. Foster et al. (Eds.)*


*doi:10.3233/APC200077*

# Towards High-End Scalability on Biologically-Inspired Computational Models

Dario DEMATTIES [a], George K. THIRUVATHUKAL [b,c] Silvio RIZZI [c]
Alejandro WAINSELBOIM [e] B. Silvano ZANUTTO [a,d]

[a] *Universidad de Buenos Aires, Facultad de Ingeniería, Instituto de Ingeniería Biomédica, Ciudad Autónoma de Buenos Aires, Argentina*
[b] *Computer Science Department, Loyola University Chicago, Chicago, Illinois, United States*
[c] *Argonne National Laboratory, Lemont, Illinois, United States*
[d] *Instituto de Biología y Medicina Experimental-CONICET, Ciudad Autónoma de Buenos Aires, Argentina*
[e] *Instituto de Ciencias Humanas, Sociales y Ambientales, Centro Científico Tecnológico-CONICET, Ciudad de Mendoza, Mendoza, Argentina*

**Abstract.** The interdisciplinary field of neuroscience has made significant progress in recent decades, providing the scientific community in general with a new level of understanding on how the brain works beyond the store-and-fire model found in traditional neural networks. Meanwhile, Machine Learning (ML) based on established models has seen a surge of interest in the High Performance Computing (HPC) community, especially through the use of high-end accelerators, such as Graphical Processing Units(GPUs), including HPC clusters of same. In our work, we are motivated to exploit these high-performance computing developments and understand the scaling challenges for new–biologically inspired–learning models on leadership-class HPC resources. These emerging models feature sparse and random connectivity profiles that map to more loosely-coupled parallel architectures with a large number of CPU cores per node. Contrasted with traditional ML codes, these methods exploit loosely-coupled sparse data structures as opposed to tightly-coupled dense matrix computations, which benefit from SIMD-style parallelism found on GPUs. In this paper we introduce a hybrid Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) parallelization scheme to accelerate and scale our computational model based on the dynamics of cortical tissue. We ran computational tests on a leadership class visualization and analysis cluster at Argonne National Laboratory. We include a study of strong and weak scaling, where we obtained parallel efficiency measures with a minimum above 87% and a maximum above 97% for simulations of our biologically inspired neural network on up to 64 computing nodes running 8 threads each. This study shows promise of the MPI+OpenMP hybrid approach to support flexible and biologically-inspired computational experimental scenarios. In addition, we present the viability in the application of these strategies in high-end leadership computers in the future.

**Keywords.** MPI, OpenMP, Central Processing Units(CPUs), biologically-inspired computational models, neuroscience, irregular computation, sparse computation.

## Introduction

Neuroscience has undoubtedly provided a more in-depth understanding of brain organization in the last decades. Nevertheless, mainstream Artificial Intelligence (AI) research is yet to incorporate these advancements in their models. This fact could be attributed–at least in part–to the success accomplished by some AI approaches–such as Deep Convolutional Neural Networks–which have achieved classification accuracy levels without precedent in the last years. Despite this, some researchers from the AI community recognize that in order to overcome current AI limitations and to create intelligent machines it will be necessary to understand and mimic the brain [1,2]. In such sense, to better understand and explore more deeply how the brain may process information it is essential to use more complex and biophysically accurate neuron and network models than the ones that are prevalent today.

The model of Hodgkin-Huxley (HH) [3]–for example–simulates synaptic receptors and ion channels explicitly. Nevertheless, the more interesting the biological mechanisms, the more limited they are by the size and complexity of the networks. There are some alternative models such as spiking model [4] and the integrate-and-fire model [5] which have been proposed as a simplification of the HH model. Such models demand less computational power, but are not able to directly simulate the biological dynamics present in ion channels. On the other side, we find deep learning (DL) applications [6] that are partially inspired by the biology of the visual ventral pathway, which have dramatically improved the state-of-the-art in many AI domains while ignoring–at the same time–important biological facts and giving priority to computational efficiency and classification accuracy.

Finding the appropriate level of detail in modeling the brain seems to be the holy grail to disentangle the mysteries of animal behavior. In [7] we introduced a biologically inspired and fully unsupervised neurocomputational approach following sequence learning mechanisms applied in [1], and gathering what are–under our point of view–only relevant neuro-anatomical and neuro-physiological facts in order to process information in cortical tissue. In such work we simulated columnar organization, spontaneous micro-columnar formation, adaptation to contextual activations and Sparse Distributed Representations(SDRs) produced by means of partial N-Methyl-D-aspartic acid (NMDA) depolarization. Our pyramidal neuron model dissociated proximal from distal dendritic branches. Proximal dendrites acted as a homogeneous set receiving only afferent information. Information in proximal dendrites determined a bunch of neural units in a Cortical Column (CC) which could be activated depending on the previous activations in the same as well as in neighboring CCs. Distal dendrites–on the other hand–received only lateral and apical information acting as independent detectors. Distal dendritic information pre-activated neural units putting them in a

predictive state in order to receive future afferent information. Some important remarks in reference to the neurocomputational approach are: (i) proximal afferent dendrites do not determine a neuron to fire, instead, they bias its probability of doing so, (ii) distal dendritic branches are independent computing elements that contribute to somatic firing by means of dendritic spikes, and (iii) prediction failures in the network produce a phenomenon called Massive Firing Event (MFE) which manifests with the activation of many neurons in a CC impairing SDRs formation. The model's feature abstraction capabilities showed promising phonetic invariance and generalization attributes, improving the performance of a Support Vector Machine (SVM) classifier for monosyllabic, disyllabic and trisyllabic word classification tasks in the presence of environmental disturbances such as white noise, reverberation, and pitch and voice variations. The work aimed to gather only biologically relevant aspects avoiding loading simulations with excessive computational burden and–at the same time–capturing the essence of the information processing properties of the cortex.

With these points in mind, certain aspects were taken into account in order to pursue the implementation of our computational model. Firstly, the biological plausibility of our model freed us from the need to compute gradients. Even though there are important works supporting the idea that *credit assignament*–the ultimate goal of backpropagation–could be a phenomenon happening in cortical tissue [8], we pondered that it is unknown whether *teaching signals* exist in the brain. Furthermore, there is not enough evidence to include a too complex mechanism in our model yet. Instead, we decided to be conservative in this respect. Secondly, prevalent DL frameworks are mainly biased towards GPU parallelization on CUDA cores. Albeit those frameworks have been extremely optimized to take the maximum advantage especially from NVIDIA cards, too many conditions have to be satisfied in order to obtain the best performance. Moreover, there exists an acute specialization of such technologies towards the precise development of certain DL frameworks with little room for innovative and specifically biologically plausible implementations. In that sense, one of the biggest problems in such approaches arises when trying to implement neural populations with sparse or random connectivity structures. Those implementations–strongly demanded in biologically plausible modelling–compromise coalescence in GPU cards and seriously impair performance [9].

Following this line, we implemented our model in C++14 using Object-oriented programing (OOP) paradigm and parallelized it by means of a hybrid strategy using MPI and OpenMP (Fig. 1). The OOP paradigm gave us a powerful tool to compose modular structures allowing the management of complex computational graphs. MPI enabled our model to run on distributed memory systems in a coherent and stable way. Finally, OpenMP provided a fine grained distribution of workload inside each computing node with the option to schedule the OpenMP threads dynamically. This allowed to manage different options of thread affinity and to vary the number of threads in each computing node, among other options.

The measurements of scaling efficiency returned by our tests allow us to claim that this parallelization strategy is a promising procedure to approach new computational implementations, with more biological plausibility and with more irregular and unstructured data-sets in high-end leadership computer resources.

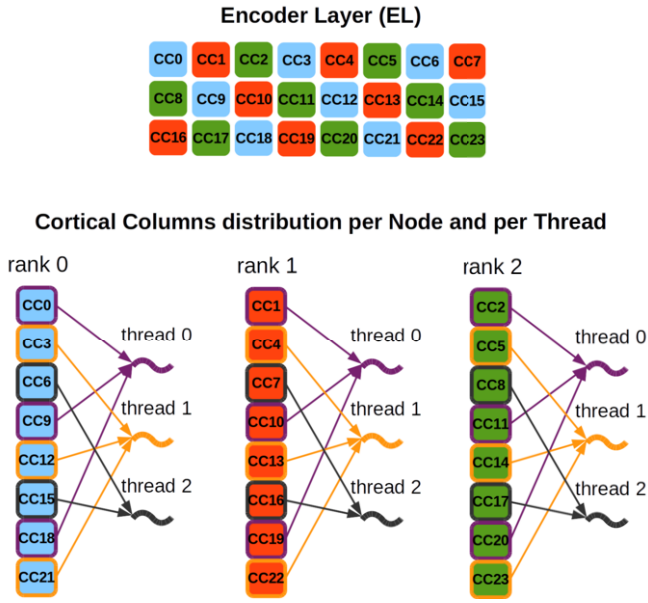



**Figure 1.** Encoder Layer (EL) MPI+OpenMP parallelization.

## Related Work

The computational effort demanded by the brain's specifications force us to consider only those physiological and anatomical features which are key for information processing avoiding loading computational simulations with unnecessary biological detail. In the same direction, parallelization strategies have to be as highly qualified as to face the challenges presented by the implementation of new–biologically accurate–computational approaches on HPC resources.

*Brain-Inspired Artificial Neural Network (ANN) Computational Approaches*

The development of ANNs is classified in three generations regarding their computational units [10,11]. In 1943, the first generation of ANNs came from McCulloch and Pitts [12]. The authors introduced neurons as computational units which received binary inputs through associated weights and produced threshold dependent binary outputs. Important derivations from such ANNs are multilayer perceptrons, Hopfield nets and Boltzman Machines.

In the second generation, neural units are computational elements whose outputs represent a continuous set of possible values obtained by means of activation functions applied to the weighted sum of the inputs. Common activation functions are sigmoid, polynomial or exponential functions. Examples of these networks are feedforward and recurrent sigmoidal neural nets. An extremely important feature of these networks is that they support learning algorithms based on gradient descend–such as the popular backpropagation. Finally, the real-valued outputs of networks of this generation are interpreted as firing rates in real neurons.

Important behavioral and physiological evidence though made *firing rate interpretation* questionable, and gave rise to the third generation of ANNs which employ *spiking neurons*–or *integrate and fire neurons*–as computational units [3,4,5]. These models–unlike the second generation models–use timing of single action potential–or spikes–to encode information. Including the concept of time in their processing model, Spiking Neural Networks(SNNs) could capture neural behavior more accurately than traditional neural networks. Unlike traditional ANNs, the main idea is that neurons in a SNN do not fire at each cycle, and rather they fire only if a membrane potential reaches its threshold.

In spite of such compelling modeling approach, threshold circuits like the ones introduced by the first generation could be seen as abstract models for digital computation on networks of spiking neurons. In such sense, one bit in active state could be interpreted as a neuron firing within certain short time window and the same bit in inactive state could be interpreted as the same neuron non-firing within such time window [13]. This coding strategy provides a good model for a network of spiking neurons as long as firing times among pre and postsynaptic neurons are synchronized within a few msec time window. There is evidence supporting the fact that this strongly synchronized activity does really occur within the nervous system [14,15].

In such sense, there are new algorithmic developments [1,7] which instead of modeling precise timing activations, prioritize the different roles of proximal and distal dendritic configurations incorporating important physiological and anatomical phenomena, such as the consideration of dendritic branches as active processing elements, the microcolumnar organization in cortical tissue and the sparse patterns of activation in the neocortex–among others. Almost all ANNs, such as those used in deep learning [6] and spiking neural networks [10], use artificial neurons without considering active dendrites and with an unrealistic low number of synapses. These facts suggest that these ANNs are missing fundamental functional properties present in the brain.

*CPUs and GPUs for ANN Large Simulations*

In the realm of biologically plausible computational models the CPU/GPU dichotomy is not clearly defined. In [9] for example, the authors analyzed the advantages and drawbacks of the CPU and GPU parallelization in different shared memory parallel paradigms, such as OpenMP, Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) of mean-firing rate neurons. The authors inspected different speed limiters such as floating point precision, thread configuration, data organization and connectivity structure of the networks. Parallel CPU implementations greatly benefited smaller networks, mostly because of cache effects. Large networks–on the other hand–benefited from the GPU only if they demanded memory beyond the available on CPU caches, otherwise an OpenMP implementation was highly preferred. The authors compared several structure representations on the different parallel frameworks showing that on CPUs, these representations reached almost the same computation time. On GPUs instead, the performance was significantly affected by violations of coalescence induced by heterogeneous data structures. Finally, the most serious

problem appeared when the network had a sparse or random connectivity structure, i.e. neurons received connections randomly from other neurons, and not in an organized or ascending order. As the authors pointed out, this totally broke down the performance of GPU implementations, while CPUs were only slightly affected. This was perhaps the strongest argument against GPU implementations of mean-firing rate neural networks, since this sparse connectivity is a repeated pattern in biological networks as well as it is in the computational model presented in this paper.

## Materials and Methods

In this paper we introduce a parallelization strategy with great independence on data coalescence and show how it scales efficiently on distributed memory systems while running our biologically-inspired computational model which simulates cortical dynamics with highly sparse and random connectivity profiles [7].

Our group pursued the implementation of a completely unsupervised and biologically inspired computational model–the Encoder Layer (EL) in [7]–which incorporated key properties from the mammalian cortex and returned phonetic features that improved the classification accuracy levels of the SVM algorithm in word classification tasks. This happened in the presence of noise, reverberation and pitch and voice variations not present during training [7]. In this paper we introduce the parallelization strategy applied to the Encoder Layer (EL) code which is approached by means of a hybrid MPI and OpenMP paradigm and through the use of MPI I/O parallel file system with Checkpoint and Restart capacity in the training stage where there is total flexibility in terms of the number of ranks with which the execution is restarted (Fig. 1).

We performed all computational experiments on Cooley [16], a visualization and analysis cluster at Argonne National Laboratory in which we executed scaling tests on the EL–the central algorithm in our model–using up to 64 nodes (one MPI rank per node) and up to 8 OpenMP threads per node/rank. We performed strong and weak scaling tests and measured scaling efficiency.

We parallelized the Encoder Layer (EL) in a way that each MPI rank ends up with one or more CCs and the CCs in each rank are distributed among different OpenMP threads. Fig. 1 shows a hypothetical distribution of CCs in an EL with 3 by 8 (24) CCs among three MPI ranks with three OpenMP threads per rank. Certain ranks could take care of a different number of CCs depending on the number of MPI ranks as well as the number of CCs in the EL. Each MPI rank distributes its CCs among different threads in the same fashion.

Information among MPI ranks must be transferred in each time step. We gather all the information corresponding to the CCs in each rank and then use MPI Bcast function to transmit such information using a special communication protocol by means of which we specify the boundaries in the information corresponding to each CC. By means of this strategy, each MPI rank has to call MPI Bcast just once in order to transmit its data.

The EL uses MPI I/O parallel file system to save its status. Each MPI rank gathers all the data corresponding to its CCs in the EL and puts such data in a

`std::stringstream` class. Then such MPI rank communicates the part of the file it will use to the other MPI ranks in order to store the data without interfering with the other ranks in the MPI environment. Finally, each MPI rank saves all its data with a unique call to MPI Write using the complete `std::stringstream`. An EL with a different number of ranks can load the same file without affecting the final results.

In this work, each MPI rank runs in a different node and keeps all the data that corresponds to the EL object. Although this general EL data is replicated in each MPI rank, this is not significant compared to the data corresponding to the CC objects. Each MPI rank keeps only the data for those CCs which are under its charge.

## Results

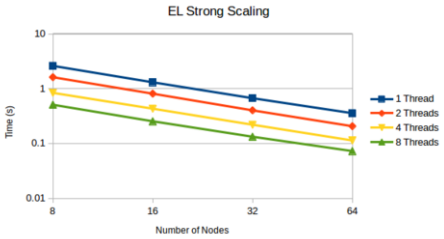
In this paper we tested the scaling efficiency of the parallelization strategies used in the EL by means of *strong* and *weak* scaling tests (Fig. 2). We conducted our tests on Cooley, a cluster to analyze and visualize data produced on high-end supercomputers at Argonne National Laboratory. Cooley has 126 compute nodes; each node has 12 CPU cores. The entire system has a total of 47 terabytes of system RAM. The Cooley node configuration has an Intel Haswell architecture with two 2.4 GHz Intel Haswell E5-2620 v3 processors (6 cores per CPU, 12 cores total), 384GB RAM, FDR Infiniband interconnect and 345GB local scratch space.

Figs. 2a and 2b show the strong scaling capacity of our code in terms of number of processing elements used for the task. In these tests we constrained the code to run one MPI rank per node. Each MPI rank spreads a specific number of threads through the different CPUs in its corresponding node as shown in Fig. 1. The problem size stayed fixed and the number of processing elements was increased. Straight lines in Fig. 2a show–at first–a good scaling capacity. Such fact is confirmed by Fig. 2b which shows the strong scaling efficiency [1].

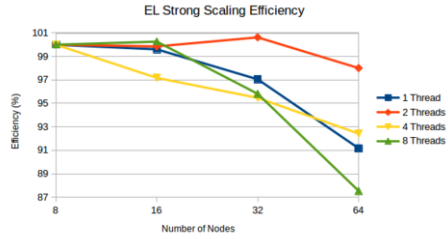
In order to avoid scaling efficiency degradation, the EL has to keep certain number of CCs per OpenMP thread. We tested the scaling running on up to 64 nodes with 8 OpenMP threads each, since the more nodes you incorporate, the more Inter-Process Communication (IPC) load you have. In order to keep a considerable number of CCs per OpenMP thread, we generated an EL with 16384 CCs. In the worst scenario there were 64 computing nodes with 8 OpenMP threads each (512 threads), the model ended up distributing 32 CCs per OpenMP thread. Each CC in this model had 225 neural units to reach a total of 3686400 neural units and 1706803200 synapses in the EL.

As can be seen in Fig. 2b, the larger amount of computing nodes (MPI ranks) with the consequent growth of MPI IPC load did not affect the strong scaling efficiency of the model which was above 87% when running 8 threads per node, but was above 97% when running two threads per node for 64 nodes.

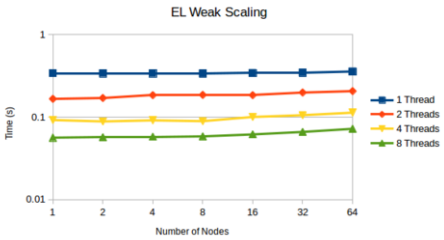
[1] Let $t_1$ be the amount of time to complete a work unit with 1 processing element, and $t_N$ the amount of time to complete the same unit of work with $N$ processing elements, the Strong Scaling Efficiency is: $t_1/(N * t_N) * 100$



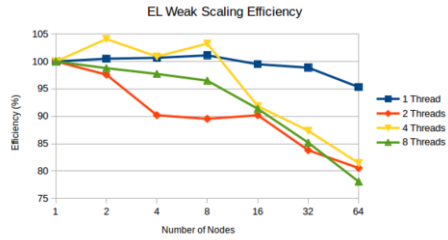

(a) Strong Scaling. Run time vs. the number of nodes for different number of threads per node.




(b) Strong Scaling. Efficiency vs. the number of nodes for different numbers of threads per node. Race line (reference) is taken at 8 computing nodes.




(c) Weak Scaling. Run time vs. the number of nodes for different number of threads per node.




(d) Weak Scaling. Efficiency vs. the number of nodes for different numbers of threads per node.

**Figure 2.** Strong and Weak scaling tests of the EL algorithm on Cooley nodes. Each MPI rank runs in a different node with 1, 2, 4 and 8 OpenMP threads running in each rank.

In reference to Weak Scaling, in order to keep the ratio of CCs per OpenMP thread constant, we used increasing EL sizes and kept a ratio of 32 CCs per OpenMP thread. Figs. 2c and 2d show the weak scaling performance of the model. In this case the problem workload assigned to each processing element stayed constant and additional elements were used to solve a larger total problem. The horizontal lines in Fig. 2c show–at first–a good scenario. As can be seen in Fig. 2d, the scaling efficiency was always above 75% [2]. These measures show that the model parallel execution was not affected by MPI IPC load as the number of computing nodes increased. This scenario was specially evident for the case of one OpenMP thread in whose case the worst efficiency was above 95%.

## Discussion and Conclusion

In this paper we show how parallelization strategies with great independence on data coalescence, scale efficiently on distributed memory systems while running

[2]Let $t_1$ be the amount of time to complete a work unit with 1 processing element, and $t_N$ the amount of time to complete $N$ times the same unit of work with $N$ processing elements, the Weak Scaling Efficiency is: $t_1/t_N * 100$

a biologically inspired computational model with highly sparse and random connectivity profile.

Algorithmic implementations strongly based on Single Instruction, Multiple Data (SIMD) parallel computing architectures, impose important restrictions on memory data alignment. OpenMP threads in shared memory systems are instead highly independent and powerful processing abstractions which can perform complex tasks with eventual vectorization optimizations when possible.

Our findings show that the parallelization strategies used in this work present good and robust scaling efficiency even in the face of intensive IPC loads. Such behavior can be kept in a sustained manner. Achieving load balance in distributed memory systems is extremely expensive, given the amount of IPC overload needed. In this manner, we claim that the best way to balance computational load among computing elements is to try to confine as much computational load as possible in a shared memory unit without far exceeding the concurrent threading or hyperthreading capacity and/or cache memory capacity provided by a node. Once such conditions are satisfied, it is relatively straightforward to spawn a number of threads in which the work could be distributed. Once in a shared memory system, OpenMP threads are much lighter than MPI processes, and they do not need complex communication methods to share data. Furthermore, OpenMP threads can manage load balancing efficiently and automatically since OpenMP manages dynamic parallel schedule on its own. This is highly desirable, specially in a simulation environment in which individual modules–such as CCs in our cortical model–are not uniformly analogous in terms of size and or connectivity. In MPI instead, the programmer has to deal with load balancing using intensive IPC which is highly expensive especially when the communication is carried out among processes in different nodes. On the other side, OpenMP threads suffer from false sharing in the CPUs caches, but with a highly flexible parallelization scheme as the one used in section Materials and Methods, the user can flexibly vary the parallelization granularity as to achieve the best performance, avoiding that each thread exceeds the quota of cache memory available in each CPU.

In Fig. 2 the phenomenon of *super-linear* speedup is present for several cases. In [17] the authors pointed out that: The superlinear speedup performance in persistent algorithms occurs mainly due to the increased cache resources in the parallel computer architectures, the prefetching of shared variables in shared memory organization, or better scheduling in heterogeneous environments. The effects of the shared memory architectures also impact the performance behavior of the granular and scalable algorithms. We endorse such statement and consider it sustainable as a general explanation for our case. Nevertheless, we also consider that more in depth analysis of memory utilization using profiling tools will be needed in the future.

The scenario in which the computational burden assigned to each shared memory system is distributed among a set of highly lightweight, flexible and dynamic OpenMP threads, is really favorable in a context in which the number of CPUs sharing memory increases specially in high-end supercomputers. In such respect and in the face of the good results returned by our experiments, we evaluate as viable the implementation of these parallelization strategies in high end supercomputers in the future.

We thus claim that this work introduces parallelization strategies whose flexibility and robustness are particularly useful in overly variable and biologically inspired computational scientific scenarios whose modelization approaches can vary dramatically in different biologically accurate implementations strategies in which there are erratic network structures with highly sparse and random connectivity profiles.

## References


[1] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10:23, 2016.

[2] Adam H. Marblestone, Greg Wayne, and Konrad P. Kording. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10:94, 2016.

[3] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bulletin of Mathematical Biology*, 52(1):25 – 71, 1990.

[4] Eugene M. Izhikevich, Joseph A. Gally, and Gerald M. Edelman. Spike-timing dynamics of neuronal groups. *Cerebral cortex*, 14 8:933–44, 2004.

[5] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sept 2004.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[7] Dario Dematties, Silvio Rizzi, George K. Thiruvathukal, Alejandro Wainselboim, and B. Silvano Zanutto. Phonetic acquisition in cortical dynamics, a computational approach. *PLOS ONE*, 14(6):1–28, 06 2019.

[8] Jordan Guerguiev, Timothy P. Lillicrap, and Blake A. Richards. Towards deep learning with segregated dendrites. In *eLife*, 2017.

[9] Helge lo Dinkelbach, Julien Vitay, Frederik Beuth, and Fred H Hamker. Comparison of gpu- and cpu-implementations of mean-firing rate neural networks on parallel hardware. *Network: Computation in Neural Systems*, 23(4):212–236, 2012. PMID: 23140422.

[10] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997.

[11] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Third generation neural networks: Spiking neural networks. In Wen Yu and Edgar N. Sanchez, editors, *Advances in Computational Intelligence*, pages 167–178, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[12] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, 52 1-2:99–115; discussion 73–97, 1990.

[13] Leslie G. Valiant. *Circuits of the Mind*. Oxford University Press, Inc., New York, NY, USA, 1994.

[14] M. Abeles, Hagai Bergman, E Margalit, and Eilon Vaadia. Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. *Journal of neurophysiology*, 70 4:1629–38, 1993.

[15] W. Bair, C. Koch, W. T. Newsoine, and K. H. Britten. Reliable temporal modulation in cortical spike trains in the awake monkey. *Proceeding of Dynamics of Neural Processing*, pages 84–88, 1994.

[16] Cooley | Argonne Leadership Computing Facility. https://www.alcf.anl.gov/user-guides/cooley. Accessed: 2018-11-24.

[17] S. Ristov, R. Prodan, M. Gusev, and K. Skala. Superlinear speedup in hpc systems: Why and when? In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 889–898, Sept 2016.