Cortical-inspired placement and routing: minimizing the memory resources in multi-core neuromorphic processors

Leite, Vanessa R C ; Su, Zhe ; Whatley, Adrian M ; Indiveri, Giacomo

DOI: https://doi.org/10.1109/biocas54905.2022.9948653

Cortical-inspired placement and routing: minimizing the memory resources in multi-core neuromorphic processors

Vanessa R. C. Leite, Zhe Su, Adrian M. Whatley, Giacomo Indiveri
Institute of Neuroinformatics, University of Zurich and ETH Zurich
Email: vanessa@ini.uzh.ch

***Abstract*—Brain-inspired event-based neuromorphic processing systems have been emerging as a promising technology in particular for bio-medical circuits and systems. However, both neuromorphic and biological implementations of neural networks have critical energy and memory constraints. To minimize the use of memory resources in multi-core neuromorphic processors, we propose a network design approach that takes inspiration from biological neural networks. We use this approach to design a new routing scheme optimized for small-world networks and, at the same time, to present a hardware-aware placement algorithm that optimizes the allocation of resources for small-world network models. We validate the algorithm with a canonical small-world network and present preliminary results for other networks derived from it.**

***Index Terms*—compiler, neuromorphic processors, hierarchical routing, small-world networks, multi-core, scaling up, cortical networks**

I. Introduction

The large energy costs of Deep Neural Network (DNN) and Artificial Intelligence (AI) algorithms are pushing the development of domain-specific hardware accelerators [1]. Neuromorphic processors are a class of AI hardware accelerators that implement computational models of Spiking Neural Networks (SNNs) adopting in-memory computing strategies and brain-inspired principles of computation [2]–[4]. They represent a very promising approach, especially for edge-computing and bio-signal processing applications, as they have the potential to reduce power consumption to ultra-low (e.g., sub-milliwatt) figures. However, the requirement of SNN hardware accelerators to store the state of each neuron, combined with their in-memory computing circuit design techniques leads to very large area consumption figures, which limits the sizes and numbers of parameters of the networks that they can implement.

The current strategy used to support the integration of large SNN models in these accelerators is to use multi-core architectures [5]–[9]. In these architectures, each core either *emulates* with analog circuits [5] or *simulates* with time-multiplexed digital circuits [7]–[9] neuro-synaptic arrays in which both the synaptic weight matrix and the network connectivity routing memory blocks occupy a significant proportion of the total layout area. Although the advent of nano-scale memristive devices can mitigate this problem by enabling the construction of dense cross-bar array structures for storing the weight matrices [4], the problem of allocating routing and connectivity resources to allow arbitrary networks at scale is of a fundamental nature that even memristors or 3D-VLSI technologies cannot solve [10].

Finding trade-offs to optimize both weight-matrix and connectivity/routing memory structures in multi-core neuromorphic processors can therefore have a significant impact on their total chip die area and on the size of the networks they can implement. Following the original neuromorphic engineering approach [11], in this paper, we look at animal brains for inspiration and propose brain-inspired architectures and strategies to reduce the memory needed to place and route networks, thus reducing the total chip die area.

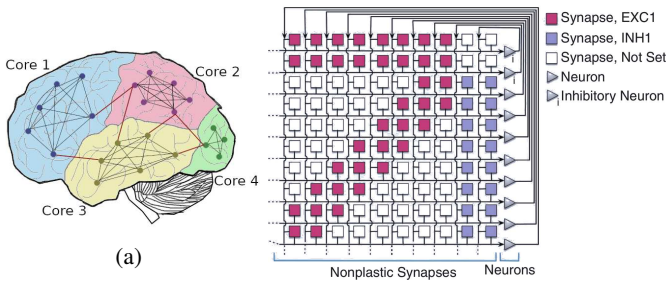
Specifically, we show that, by focusing on small-world network connectivity, we can implement trade-offs that minimize memory consumption requirements while still enabling the design of SNN architectures that can solve a wide range of relevant "edge-computing" problems, i.e., the types of sensory-motor processing problems that animals must solve in the real world.

II. Neural network connectivity schemes

A. In biological systems

In animal brains, computation and other functions emerge from the interaction of neural areas. Brain networks have short path length, high clustering, and a modular community structure [12]. They express modular, *small-world*, heavy-tailed characteristics. In small-world networks, most edges form small, densely connected clusters and the others maintain connections between these clusters (Fig. 1a). This mixture of local clusters and global interaction generates a structure that provides function and integration in the brain that can support a wide range of complex computation, cognition and behavior [12]. By restricting the types of SNNs that can be implemented in neuromorphic processors to small-world networks, we can dramatically reduce the memory required to specify the routing/connectivity schemes while still supporting

This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Program Grant Agreement No. 724295 (NeuroAgents), and by the Electronic Component Systems for European Leadership (ECSEL) joint undertaking Grant Agreement No. 876925 (ANDANTE)



(a) (b)

Fig. 1: Small-world networks in brains and neuromorphic chips. (a) Schematic example of the patterns of connections found within and across different brain areas, which are highly correlated with brain functions [18]. Biological neural networks are often highly recurrent and have dense connections among nearby neurons and sparse connections to specific/far-away neurons, following an exponential decay in the number of connections with increasing distance. (b) Example of a "Winner-Take-All" (WTA) network implemented on a neuromorphic processor (from [19]). Blue squares represent inhibitory synapses with negative weights, and red squares represent excitatory synapses with positive weights.

a wide range of computations for solving pattern recognition and signal processing tasks, e.g. [13]–[17].

B. Routing schemes in neuromorphic hardware

Multi-core neuromorphic processors usually use Network-on-Chip (NoC) designs for managing the communication of neurons between cores. Different neuromorphic chips adopt different NoC architectures, according to the application. Mesh architectures [8] represent an easy way to build large-scale systems, however, when the NoC size increases, the required hardware area increases considerably which reduces the system scalability. In flattened butterfly architectures [20], neuron cores belonging to the same row and column can communicate directly, with lower routing latency, but this architecture also brings the disadvantage of large area cost and poor multi-casting support. In [21], the authors proposed a hierarchical architecture that overcomes some of these disadvantages, by using off-chip Dynamic Random Access Memory (DRAM) to store the routing lookup table, which significantly increases power consumption.

Current methods for saving power adopt in- or near-memory computing strategies. However, when on-chip memory is used to store configurable neuron connections, the required hardware area increases proportionally with the number of neurons and synapses. For example, [5] uses on-chip hierarchical routing with a combination of point-to-point source-address routing and multicast destination-addresses to reduce memory usage, and still, the memory used takes around 80% of the chip area.

C. Network placement on neuromorphic hardware

The NoC and routing scheme define the source and target memory structures, thus setting constraints and restrictions on placing a network on it. Placing an SNN onto neuromorphic hardware is a mandatory step, needed to exploit the advantages of the hardware [22], [23], and each type of hardware has its own set of tools to make it appealing to SNN developers. Two main approaches are used to offer such a set of tools: platform-based design and hardware-software co-design. The approaches proposed in [24]–[26] are platform-based designs, where the development of the hardware is independent of its software, allowing exploration of alternative solutions, in a more general setting. In this work we use the second approach, where the memory minimization strategies validated in software lead to routing circuit specifications for new neuromorphic chip designs. The contributions of this paper are two-fold: i) a novel architecture designed to support small-world networks, and ii) a new placement algorithm to provide specifications for new hardware designs. When developing the placement algorithm we take into account requirements derived from hardware design choices proposed by the chip designers, which define constraints on the algorithm. In this way, software and hardware are optimized together.

III. Hardware-software co-design strategy

By limiting the topology of the networks to be small-world networks, we can minimize memory requirements by reducing the address space (i.e., the number of bits and hence chip area) required to map the (many) connections between nearby neurons, and allocate more bits for larger address space domains used by the sparse long-range connections. We took as "canonical" examples of Winner-Take-All (WTA) networks, as shown in Fig. 1b, networks that have small-world connectivity matrices. And, to generalize to other types of small-world networks, while minimizing memory usage, we propose a new heuristic for a hardware-aware *neuromorphic compiler*.

A. Hierarchical routing model

Figure 2 shows our hierarchical routing scheme designed to support small-world network connectivity with on-chip memory. Since we have densely connected clusters, there is no specificity within the cores in our architecture: independently of which neuron is sending a spike, all other neurons inside the same core will receive it. Following the exponential decay of connections with distance observed in biology, we assume that the number of connections required between cores is dependent on the distance between them, and we associate physical distance with the levels in the router hierarchy: e.g., all cores that can be reached via an R1 router level are at distance 1. Each neuron in a core reached through an R1 router can receive spikes from half of the neurons belonging to the source cores. Similarly, neurons reached through an R2 router will accept spikes from a fourth of the neurons of the source cores. As the distance between cores increases, fewer connections are made, thus, there is no need to allow

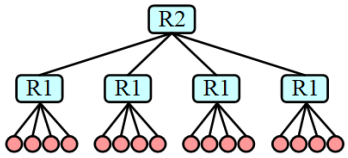


Fig. 2: Hierarchical routing scheme: example with three levels of routers. R0 (not shown) connects neurons inside the same core (each core is shown as a red circle), R1 manages connections between n local cores ($n = 4$ in this example), and R2 connects n R1 routers. With this routing scheme we can reduce dramatically the number of bits (memory) necessary to specify a small-world network, even with a large fan-in.

connectivity between all neurons in different cores. This allows us to reduce the connectivity address space and thus reduce the overall memory required to specify the source population address for each neuron.

These connections have a constrained address space, i.e., since there is an upper bound on the router to which a spike can be sent, it is not necessary to take into account all of the neurons on the whole chip. In a model going up to the R2 level, each neuron needs memory to store 10 bits in total, allowing a fan-in from up to half of the neurons in the cores that can be reached through the R1 level plus a fourth of the neurons in each of the cores that can be reached through R2. To support this reduction in address space we compute the routing distance by combining the use of computing logic and memory in each router module and update the distance information in the spike packet as it traverses the router, thus reducing the address space to the bare minimum needed by the local cluster.

B. Placement algorithm

We present a placement algorithm (Alg. 1) that optimally maps SNN models that follow a small-world structure onto neuromorphic hardware architectures that implement the specificity and distance-based connectivity constraints described in Section III-A.

To place neurons in cores, we first find *cliques* in the network graph. Each clique is a group of densely connected neurons, and they will be placed in different cores. (If there are cliques larger than the core size, they can be subdivided.) This first step gives us the number of cores needed to place the network. Then we calculate the distance between two cores i and j, as a function of the number of connections that they share. We define the distance between the core and itself ($i = j$) as zero, and between two cores that do not share connections as –1. Note that our distance definition is a *quasi-metric* that can be non-symmetric, i.e., the distance from core i to j might not be the same as from j to i. The maximum distance in the network indicates the maximum router level we need to map the network.

Having determined the number of cores and distances between cores, we can finally place connections. We start the connection placement from the closest pair of cores. The closer

Alg. 1 High-level description of the placement algorithm

Require: $G = (V, E)$, where V is the set of neurons and E is the set of paired neurons.
find cliques on G
for each clique **do**
  Add clique to a new core
end for
$n =$ number of neurons per core
$e(i, j) =$ number of connections core i receives from core j.
initialize matrix $dist[i][j]$ with zeros
for each pair of cores i and j **do**:
  if $e(i, j) == 0$ **then** $dist[i][j] = -1$
  else $dist[i][j] = floor((n/e(i, j))) + 1$
  end if
end for
sort $dist[i][j]$
for each pair of cores i and j in $dist$ **do**
  if neuron in i has synapse available **then**
    connection is placed
  else
    connection is flagged
  end if
end for

cores are to each other, the more connections they share. We allocate the nearby connections first because they are more numerous than the further-away ones. If a connection can not be added, it is flagged as unplaced. Unplaced neurons and connections can be added in a second loop, given the availability of extra cores.

IV. Results

To validate our placement algorithm we tested it with canonical networks generated to match a hypothetical neuromorphic processor comprising cores of 16 neurons. The canonical network thus have populations of 16 neurons, where each population is all-to-all connected, and the number of connections between populations drops off with the distance between the cores, as depicted in Fig. 3.

By using a canonical network that fully matches the hardware structure considered, we define a *ground truth* (GT) placement. As our algorithm maps the canonical network to this hardware perfectly, we can verify that the proposed heuristic works as expected.

To evaluate how the algorithm performs in non-optimal conditions, we performed two sets of tests. Two canonical WTA networks were considered, based on hardware with 16 neurons per core, and either 7 or 70 cores (112 or 1120 neurons respectively). First, we tested it by using perturbed networks that deviate from the canonical one by removing a percentage of nodes (1%, 10% and 25%). With this experiment we evaluate how much deviations from the canonical network affect the placement algorithm. Our algorithm still finds solutions that are very close to the GT, for small deviations (1% and 10%).

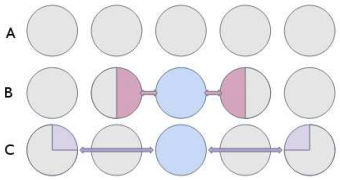
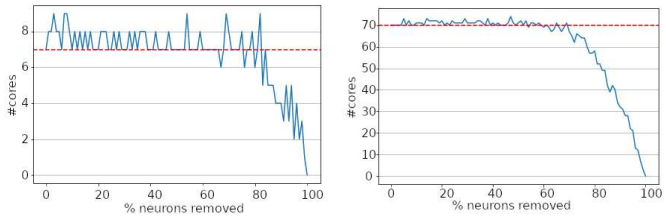


Fig. 3: Canonical networks generated to match a hypothetical neuromorphic processor. Row A shows five populations. We create populations with the same number of neurons as in a core, all-to-all connected. For simplicity, we place our populations on a 1D-line defining distances (and thus the connections between populations). In B we assign connections between cores at distance 1. These cores will share connections with half of the number of neurons in a nearby core. In C we assign connections between cores at distance 2. They share connections with a fourth of the number of neurons in a core at that distance. This process is repeated until there are no more cores or they are so far away that no connection is created. Only the procedure for the central core is shown in the figure, but this process is applied to all cores.



(a) (b)

Fig. 4: Placement of SNN network models onto the resource constrained neuromorphic hardware. Two canonical WTA networks were considered, based on hardware with 16 neurons per core, and either 7 or 70 cores. The red dashed line marks the canonical network placement. For each one of the canonical networks, we remove an increasing number of neurons and place the perturbed network. (a) shows the result for the ∼100-neuron network, and (b) for the ∼1k-neuron network. Note that for the perturbed networks, there is some fluctuation in the number of cores needed, however it stays close to the ideal case.

Larger deviations (25%) produce solutions that are close to the GT only in large networks (∼1k neurons).

Secondly, we tested the placement algorithm by removing an increasing number of nodes (from 1 to all the nodes in the network). The results of this test are shown in Figure 4.

For all perturbed networks and network sizes considered, the performance of the placement algorithm is close to the GT performance, indicating an optimal use of the limited resources on the neuromorphic hardware.

A. Memory comparison

In Fig. 5 we compare our work with TrueNorth [6] and Dynamic Neuromorphic Asynchronous Processor

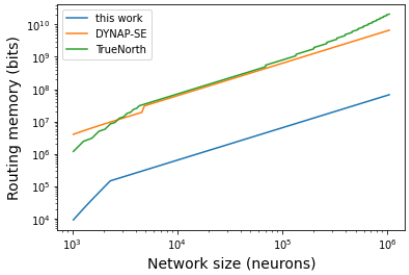


Fig. 5: Memory scaling comparison between TrueNorth [6] and DYNAP [5] architectures and this work. For comparison, we considered a core with 256 neurons in all architectures. The number of bits used by the three architectures is plotted as a function of the network size (log scale). For TrueNorth and DYNAP, the memory increases linearly with increasing fan-in/fan-out, with TrueNorth performing better than DYNAP while the fan-in fits the cores. In our proposed architecture, the increase in fan-in/fan-out demands more cores but the memory required increases at a slowly, since all the cores at the same level need a fixed number of bits to be addressed.

(DYNAP) [5] architectures. In our design, the number of bits required to fit a network does not increase with the neuron fan-in/fan-out. We can increase the fan-in by adding more cores, so the increase in memory is linear in the number of cores. TrueNorth architecture has a fixed fan-in per neuron, and with an increase in fan-in/fan-out we need to recruit relay neurons from other cores. This starts to be costly for large networks in which neurons have a large fan-in. Indeed, any architecture with a fixed fan-in per neuron will not scale well due to the requirement to resort to relay neurons [27]. Also in the DYNAP architecture the fan-in is fixed. But its mixed source/destination addressing scheme mitigates the number of intermediate nodes required. Our analysis shows that our canonical network with a million neurons requires ∼67 Mbit if implemented with our scheme, about 98× more on DYNAP (∼6591 Mbit) and about 307× more on TrueNorth (∼20649 Mbit).

V. Conclusion

The development of domain-specific neuromorphic hardware can help to advance AI for edge-computing tasks, and the optimization of memory resource allocation paves the way to building large-scale neuromorphic computing systems. In this work, we present a hardware-software co-design approach, where brain-like small-world networks are used to inspire simultaneously our routing scheme and placement algorithm.

Our co-design approach reduces the memory necessary to place and route networks that follow a small-world structure while not limiting the possible applications. Additionally, our placement algorithm can find optimal solutions for networks that follow our canonical design and can place deviations from them without diverging too much from the ideal case.

The simultaneous design of a place and route scheme is allowing us to design a new multi-core SNN chip able to handle

larger networks with a minimum of memory consumption, and thus smaller area.

REFERENCES

[1] M. Ibtesam, U. S. Solangi, J. Kim, M. A. Ansari, and S. Park, "Highly efficient test architecture for low power AI accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[2] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[3] E. Chicca and G. Indiveri, "A recipe for creating ideal hybrid memristive-CMOS neuromorphic processing systems," *Applied Physics Letters*, vol. 116, no. 12, p. 120501, 2020.

[4] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[5] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 12, pp. 106–122, Feb. 2018.

[6] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, Aug. 2014.

[7] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[8] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[9] S. Furber and P. Bogdan, eds., *SpiNNaker: A Spiking Neural Network Architecture*. Boston-Delft: now publishers, 2020.

[10] S. B. Laughlin and T. J. Sejnowski, "Communication in neuronal networks," *Science*, vol. 301, no. 5641, pp. 1870–1874, 2003.

[11] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–36, 1990.

[12] E. Bullmore and O. Sporns, "Complex brain networks: graph theoretical analysis of structural and functional systems," *Nature Reviews Neuroscience*, vol. 10, no. 3, pp. 186–198, 2009.

[13] E. Donati, M. Payvand, N. Risi, R. Krause, K. Burelo, T. Dalgaty, E. Vianello, and G. Indiveri, "Processing EMG signals using reservoir computing on an event-based neuromorphic system," in *Biomedical Circuits and Systems Conference, (BioCAS)*, pp. 1–4, IEEE, Oct. 2018.

[14] E. Donati, M. Payvand, N. Risi, R. Krause, and G. Indiveri, "Discrimination of EMG signals using a neuromorphic implementation of a spiking neural network," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 13, no. 5, pp. 795–803, 2019.

[15] N. Risi, E. Calabrese, and G. Indiveri, "Instantaneous stereo depth estimation of real-world stimuli with a neuromorphic stereo-vision setup," in *International Symposium on Circuits and Systems, (ISCAS)*, pp. 1–5, IEEE, May 2021.

[16] R. Krause, J. J. A. van Bavel, C. Wu, M. A. Vos, A. Nogaret, and G. Indiveri, "Robust neuromorphic coupled oscillators for adaptive pacemakers," *Scientific Reports*, vol. 11, no. 1, 2021.

[17] R. Kreiser, A. Renner, V. R. C. Leite, B. Serhan, C. Bartolozzi, A. Glover, and Y. Sandamirskaya, "An on-chip spiking neural network for estimation of the head pose of the icub robot," *Frontiers in Neuroscience*, vol. 14, 2020.

[18] C. W. Lynn and D. S. Bassett, "The physics of brain network structure, function and control," *Nature Reviews Physics*, vol. 1, no. 5, pp. 318–332, 2019.

[19] G. Indiveri and Y. Sandamirskaya, "The importance of space and time for signal processing in neuromorphic agents," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 16–28, 2019.

[20] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2019.

[21] J. Park, T. Yu, S. Joshi, C. Maier, and G. Cauwenberghs, "Hierarchical address event routing for reconfigurable large-scale neuromorphic systems," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2016.

[22] N. Mysore, G. Hota, S. R. Deiss, B. U. Pedroni, and G. Cauwenberghs, "Hierarchical network connectivity and partitioning for reconfigurable large-scale neuromorphic systems," *Frontiers in Neuroscience*, vol. 15, 2022.

[23] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2020.

[24] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha, "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, IEEE, 2013.

[25] F. Galluppi, S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber, "A hierachical configuration system for a massively parallel neural hardware platform," in *Proceedings of the 9th conference on Computing Frontiers*, pp. 183–192, ACM, 2012.

[26] C.-K. Lin, A. Wild, G. N. Chinya, T.-H. Lin, M. Davies, and H. Wang, "Mapping spiking neural networks onto a manycore neuromorphic architecture," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation PLDI*, pp. 78–89, ACM, 2018.

[27] A. Rao, P. Plank, A. Wild, and W. Maass, "A long short-term memory for AI applications in spike-based neuromorphic hardware," *Nature Machine Intelligence*, vol. 4, no. 5, pp. 467–479, 2022.