NeuronFlow: a neuromorphic processor architecture for Live AI applications

Orlando Moreira[1], Amirreza Yousefzadeh[1], Fabian Chersi[2], Gokturk Cinserin[1], Rik-Jan Zwartenkot[1], Ajay Kapoor[1], Peng Qiao[1], Peter Kievits[1], Mina Khoei[2], Louis Rouillard[2], Aimee Ferouge[1], Jonathan Tapson[3], and Ashoka Visweswara[1]

[1]*GrAI Matter Labs (GML), Eindhoven, Netherlands*
[2]*GrAI Matter Labs (GML), Paris, France*
[3]*GrAI Matter Labs (GML), California, USA*

***Abstract*—Neuronflow is a neuromorphic, many core, data flow architecture that exploits brain-inspired concepts to deliver a scalable event-based processing engine for neuron networks in Live AI applications. Its design is inspired by brain biology, but not necessarily biologically plausible. The main design goal is the exploitation of sparsity to dramatically reduce latency and power consumption as required by sensor processing at the Edge.**

I. Introduction

Vision and audio sensor processing is currently dominated by neural networks. These are robust, more accurate and easier to develop than hand-crafted feature extraction/classification techniques. Their emergence enabled a plethora of applications and opened new markets and opportunities.

We focus on Edge of the cloud applications and in, particular, on **Live AI**, i.e. applications that receive live data from one or more sensors and quickly react to environment changes, making decisions in real-time. Live AI typically has stringent requirements in timing (especially latency) and power consumption, and thus cannot be off-loaded to the cloud.

We contend that popular strategies to process neural networks for Live AI suffer from a power bottleneck that cannot be addressed by existing compute architectures. NeuronFlow bridges this gap by employing both recent advances in neuromorphic engineering and older ideas from dataflow processors.

State-of-the-art vision and audio sensors generate large volumes of time-sampled data. Most commercially available vision sensors rely on the capture of complete images (frames) at equally-spaced (i.e. periodic) time intervals, regardless of whether the scene changes. Such sensors are called *frame-based*.

Algorithms processing vision sensors typically follow a frame-based structure both because this fits the dominant, frame-based sensor technology, and because it enables the reuse of single-frame algorithms to process frame sequences, ie, a single picture object recognition DNN can be applied to a video sequence frame by frame.

As a consequence, the same irrelevant background objects are repeatedly identified and analyzed across frames. The processing of all these superfluous data dramatically increases computational load, resulting in highly inefficient, power-hungry processing. This would not be a serious issue if the absolute computational requirements of image sensing algorithms were low, but neural networks are computationally intensive and come at a significant power consumption cost. For current processor technology, and for Live AI, these costs are highly prohibitive.

Our approach to these challenges is grounded on neuromorphic engineering. Biological neuron networks are more efficient in processing sensory data than computer systems, doing it with considerable less computation and at a much lower power consumption. The performance of state-of-the-art neural processors, however, is not even close to brain performance concerning the trade-off between power consumption, accuracy, and speed. The human brain contains approximately 86 billion neurons and 150 trillion synaptic connections while only consuming around 20W [1]. Visual and audio processing consumes around 30% and 3% of brain resources, respectively [2]. And even though the biological fabric of the brain is not as efficient as our modern silicon technologies, it achieves extremely efficient power consumption for streaming signal processing. State-of-the-art DNN processors can attain 5 TOPs/W [3]. To execute a neural network with 150T synapses in this technology with the assumption of 10Hz updates (1.5P operations per second), requires 300W of power (15x more than the human brain). Biological studies [4] [5] show that, most of the time, only about 1% to 10% of the neurons in the brain are active. This suggests that, by efficiently avoiding redundant processing, it should be possible to bridge the power consumption gap between DNN accelerators and the brain. SectionIII, which discusses **sparsity**, will elaborate on this.

The biological brain has other impressive characteristics. It scales from 10k neurons in a worm brain up to 86B neurons in a human brain, while using the same building blocks and architecture. This level of **scalability** is not found in any human-designed processor. It is achieved via asynchronous distributed processing and event-based communication [6].

In-memory processing In the brain, memory and processor are not separate entities as in typical von-Neumann architectures. And there is a cost to this separation. Measurements show [7] that, in a conventional processor, an external memory access consumes two orders of magnitude more power than a MAC (Multiply-ACcumulate) operation, while local memory accesses consume almost the same power. Beyond energy reduction, in-memory processing enables scalability, as it

978-3-9819263-4-7/DATE20/©2020 EDAA

mitigates the memory bandwidth bottleneck [8].

We propose to dramatically reduce power consumption and computational requirements by selectively applying neuromorphic concepts to processor design.

The next section reviews related work. Section III discusses sparsity and how SpArNet (Sparse Asynchronous Neural Network) can efficiently exploit sparse DNN inference. Section IV presents NeuronFlow.

II. The State-of-the-art and its limitations

A neuromorphic processor is a neural network execution machine where the neuron model is inspired by biology. Currently, the highest level of bio-plausibility can be found in analog neuromorphic processors like BrainScaleS [9], DYNAPs [10], and Neurogrid [11]. This is achieved by designing an analog electronic circuit to emulate the behavior of a biological neuron and replicating it thousands of times.

The big advantage of analog neuromorphic design is power consumption, attained by applying asynchronous technology and sub-threshold design. Its main disadvantages are a large silicon area per neuron, which severely constrains the neuron count in a chip, and the limitations of analog circuitry regarding fabrication variations and noise.

In the digital domain, neuromorphic platforms typically contain many processing cores, with each core simulating many neurons. Per neuron, a memory word stores the membrane potential, while a shared ALU performs operations. Cores communicate through packet-switched NoCs. Some examples are IBM TrueNorth[12], Intel Loihi[13] and SpiNNaker[14].

SpiNNaker[15] (2012) aims at partially emulating the human brain in real-time. It consists of a matrix of ARM processors connected by asynchronous routers and a shared HBM memory. SpiNNaker uses a scalable Globally-Asynchronous-Locally-Synchronous (GALS) NoC architecture[16] that can connect more than 1M cores in a single platform. The flexibility comes at the cost of higher power consumption. Because of small on-chip memories, execution frequently needs random off-chip memory access, resulting in high latency and a memory bottleneck [17].

TrueNorth[12] (2014) was proposed as a new type of processor architecture for natural signal processing. With 5.4B transistors, it was IBM's largest chip. It contains 4096 cores. Each core emulates 256 neurons and communicates through a GALS NoC. It achieves very low power consumption (around 100mW) by exploiting asynchronous, event-driven execution. It suffers from limited connectivity, as each neuron can only connect to 256 other neurons in one core with binary weights. This makes it inefficient to implement even moderately complex networks.

Loihi[13] (2017) is a many-core SoC, designed by Intel, with 128 cores, connected through two independent NoCs. Each core emulates 1024 neurons. A flexible connectivity scheme allows exploitation of structural sparsity. It supports flexible synaptic weight precision (from 1b to 9b), allowing a trade-off between the number of connections and the precision per connection. The neuron model is fixed and designed for bio-plausibility. Thus, applications are constrained to platform-specific algorithms. Loihi consumes almost half of its silicon area in on-chip learning. This learning mechanism has not shown competitive performance in accuracy.

In summary, known digital neuromorphic approaches aim at understanding the brain, thus focusing on mimicking its operation. To achieve competitive performance in power and latency, we believe one must instead find out which brain characteristics contribute to its desirable properties and exploit them in a way fitting modern silicon technology. NeuronFlow does this by exploiting sparsity, the subject of our next section.

III. Sparsity and Sparse Asynchronous Inference

A. The Importance of Sparsity

Frame-based sensors capture enormous amounts of redundant data. This means that relevant information is generally sparse. Sparsity is present within a frame (spatial sparsity), in-between frames (temporal sparsity) and across the neural network (structural sparsity).

Structural sparsity: The dominant operation in DNN is multiplication-accumulation of synaptic weights by neuron output. These can be skipped when either operand is zero. DNNs are generally built with regular topology, and this regularity is often exploited by the processor architecture. However, after training, some connections may become unnecessary. Weight pruning [18] will assign zero value to insignificant weights. To exploit this, the processor must skip zero-weight connections, but this creates topological irregularities that may cause execution overhead.

Spatial sparsity: In DNNs, a neuron accumulates its inputs and then applies a non-linear activation function to the result to produce its output. It is known that the function's non-linearity is more important than its shape, and a DNN can adapt to many activation functions. Using $ReLU$ ($y = max(0, X)$) became popular since it is partially linear and efficient for hardware implementation. The output of neurons with $ReLU$ is zero when the accumulation result is negative. This imposes high sparsity (around 50%) to each layer. Again, skipping zeros during computation will conflict with exploitation of the regular structure.

Temporal sparsity: data are said to be temporarily sparse if they rarely change over time. For example, the output of a security camera in a quiet place mostly contains the unchanging background scene. Exploiting temporal sparsity means skipping the update of a neuron when none of its inputs changes. The algorithm needs to remember the results of past calculations and reuse them efficiently, thus requiring a large memory. This is the case with biological brains, which trade-off power consumption for memory. Therefore, even though the volume of the brain is large, it is very low power, as only 1% to 10% of its neurons are active at a time [4] [5]. This contrasts with conventional ANN inference where all neurons are updated and communicate their output for every frame.

As Neural Networks are mostly used for natural, temporally-sparse signal processing, such as audio/video streams, enabling sparse processing in inference engines dramatically impacts

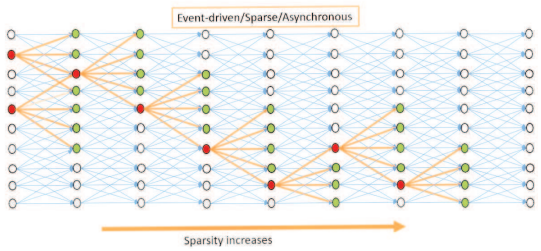

Fig. 1: A feed-forward neural network. Circles represent individual neurons. Green neurons received a spike and should be updated, red neurons fired after receiving a spike. Execution goes from left to right, from sensor input to classification output. Typically, sparsity increases with depth.

power consumption. The next section introduces SpArNet, an example of efficient DNN inference in NeuronFlow. SpArNet adapts bio-inspired Spiking Neural Networks (SNN) to perform sparse inference of non-spiking ANNs, without compromising accuracy. It adapts known schemes [19] [20] [21] [22] [23] to NeuronFlow.

B. SpArNet

Fig. 1 illustrates a simplified schematic of a Spiking Neural Network (SNN). In an SNN with Leaky Integrate&Fire (LIF) neurons, each neuron has a potential which is updated upon the integration of its inputs over time. If a neuron's potential reaches the predefined threshold, an output spike is fired and propagated through its output synapses. If not, the spiking neuron does not generate output. Similar to our brain, only a small number of neurons is active at a time.

SNNs have two advantages over synchronous processing. First, unlike a synchronous neuron, an SNN neuron only updates its potential when there is an input event (spike). Second, input events can be processed immediately and the neuron can fire anytime without waiting for an external trigger.

SpArNet shares characteristics of both synchronous ANN and LIF. It performs the inference of an ANN, but asynchronously, and with the number of operations optimized for spatio-temporal sparsity. Starting from an ANN, we obtain a SpArNet with the same topology by quantizing the neurons' activation functions. In SpArNet, events are valued and represent quantized changes in a neuron output value. To perform an inference, each neuron updates its potential upon receiving an event. If the change in output is higher than a defined threshold, the neuron will fire an event to its consumers. This is known as change-based or delta inference.

We use hysteresis quantization, not direct quantization, to set thresholds [19]. This avoids excessive firing activity. Fig. 2 compares the spiking behavior for these quantization schemes. In direct quantization, when the input (X) is near the transition point of two quantization levels, small variations/oscillations in X may result in several big changes in the quantized X which is not desirable. Choosing the threshold is a trade-off,

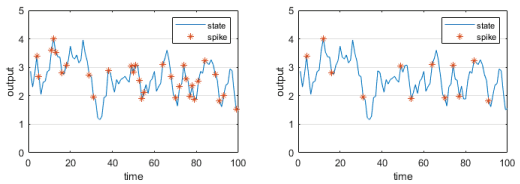

Fig. 2: Comparison between direct (left) and hysteresis (right) quantization when the quantization level (threshold) is one.

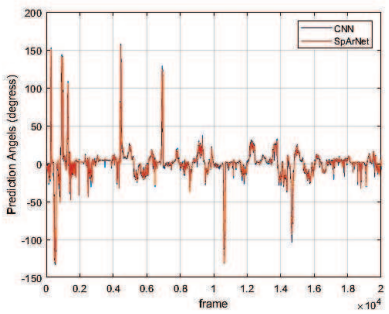

Fig. 3: Prediction angles for the first 20,000 frames of the original CNN inference and the proposed SNN inference.

as coarser quantization results in higher quantization error but generates fewer spikes, resulting in lower computational load.

As an example, our results show a reduction of one to two orders of magnitude in the number of updates required when executing PilotNet in GrAI One. PilotNet is a neural network introduced by NVIDIA [24] along with the dataset[1]. The dataset contains a video recording (10 fps) by a camera placed in front of the car as well as the corresponding steering angles

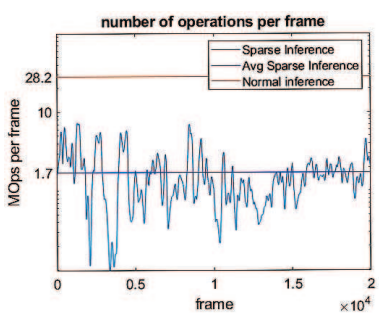

Fig. 4: Number of MACs for the first 20,000 frames in CNN versus SparNet inference (original 10fps recording).

[1]https://github.com/lhzlhz/PilotNet

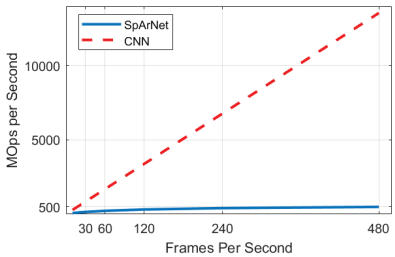

Fig. 5: PilotNet: the average number of operations per second increases linearly with fps for CNN. For SpArNet, it saturates.

for each frame of the video[2]. The output of the network is the predicted steering angle for the input frame. This network contains 108K neurons and 1.6M parameters.

Fig. 3 shows the output of PilotNet for the original CNN inference versus the proposed SNN inference for part of the dataset. Fig. 4 shows the number of required MAC operations during inference of PilotNet in the original CNN and our SNN network for the first 20,000 frames. As the original PilotNet dataset has a very low frame rate (10 fps), we tested the network with higher frame-rate recordings[3] (480fps). As it is shown in Fig. 5, savings in number of operations increase with frame rate, due to increased inter-frame redundancy[4].

SpArNet enables sparse asynchronous execution, but exploiting this in a conventional processor is difficult, and that may be the reason why regular inference models are typically preferred. Most bio-inspired models run very slowly on a conventional computer. We designed NeuronFlow to be flexible to program and efficient in exploiting all types of sparsity, as we shall see in the next section.

IV. The NeuronFlow Architecture

A. Consequences of sparsity for computer architecture

To exploit temporal sparsity, the computing machine must:

- keep a **resilient neuron state** across time, to track variation in output and avoid full re-computation of state every time a change and/or a time-trigger happens.
- process updates to resilient neuron states as synapse input arrives in a **event-based**, data-dependent order, since the neurons to be updated are not known at compile-time; furthermore, since there are few updates (due to spatial and temporal sparsity), avoid pulling values from memory to decide what neurons to update; this suggests a **data-flow execution model**, where produced results trigger consuming computations (instead of relying on a pre-defined execution order).

[2]https://youtu.be/_N7nC-8YxzE shows the original PilotNet inference.
[3]https://youtu.be/LyPTj3Parp0
[4]https://youtu.be/OySJKJWGrjw shows number of operations for the PilotNet dataset per frame with a dynamic illustration.

Conventional processor architectures for signal processing, however, rely heavily on architectural features that are counter-productive in processing sparse sensor input, as follows.

Data-value-independent execution order: a typical implementation will cycle through all neurons to update them in a pre-determined order, working in a feed-forward manner through the layers. Per layer, it accesses the memory-stored values of output synapses one by one. A high-degree of parallelism and reuse can be achieved, but each neuron and each input synapse are visited once. This makes sense when evaluating the full network; less so when computing sparse updates. In a sufficiently sparse system, an optimized static-order of evaluation will not compensate the overhead of so many redundant updates.

Locality of reference: memory accesses to large, cheap memory arrays incur high latencies. Communication devices (busses and NoCs) and DMA accelerators try to alleviate this by reading large chunks of contiguous data items in bursts; memory hierarchy techniques like caching work by creating local copies of frequently accessed, contiguous data in faster memory. These techniques - burst access, DMA and caching - make sense when a) the same data are accessed frequently in succession and b) access to a data item is a good predictor for a subsequent access to contiguous data (such as when neuron updates are processed in-order, as described above). Such techniques make less sense when computing sparse updates. This is not a fully black and white picture: in video processing, there is some locality of reference as, for instance, when an object moves, many contiguous pixels will change. Nonetheless, employing burst accesses and cache mechanisms to applications with little locality of reference may result in costly hardware overhead and in performance degradation [21], through failed speculative data access and unnecessarily complex control.

Pull memory model: in conventional computer architectures, the processor must obtain data by requesting it from memory; it must then wait for data to be sent, with varying latency depending on where the requested data resides in the memory hierarchy; this means that the effectiveness of the processing chain is conditioned by the memory architecture, and that considerable communication bandwidth and power must be spent pulling data from memory; moreover, for a sparse application, this may result in spending resources and power just to obtain data that do not need to be (re)processed.

B. The NeuronFlow Architecture

NeuronFlow (see Figure 6) is a scalable many-core array of event-processing cores with distributed execution control. In a possible realization, NeuronFlow I, each Neuron Core is comprised of 1024 neurons states organized in Neuron-State Memory, 1024 Synapses to keep the neural network parameters, allowing for the implementation of hundreds of thousands of synapses per cluster through our proprietary connection sharing schemes (explained below). The number of cores in a chip will vary (up to several thousands) based on the performance target, and the architecture can be scaled post-

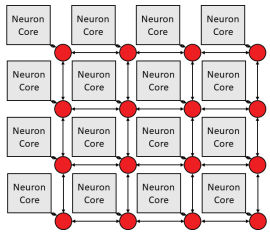

Fig. 6: The NeuronFlow many-core grid.

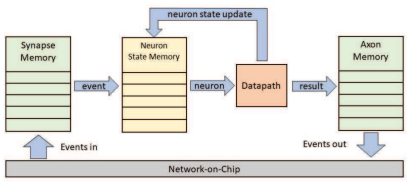

Fig. 7: The event-processing pipeline of a Neuron Core

silicon through chip tiling. The NoC is optimized for small packet transactions (to support sparse communication).

Each Neuron Core implements an event-processing pipeline (Figure 7), designed to handle one incoming event per clock cycle. In the basic execution model, each event corresponds to the output of a source neuron which is driven through a weighted synapse to a target neuron. Moreover, if our connection sharing scheme is used, events to many weighted synapses can be packed in a single output event that will address multiple target neurons in the destination core.

The processing of an event is as follows. The Neuron Core receives events from the network. These events are processed in order of arrival. Each event triggers one or more synapses. For each synapse thus triggered, the Neuron Core will fetch weight and other parameters from the parameter memory, and multiply the event value by the weight of the synapse. It will then read the state of the target neuron from the Neuron State Memory and accumulate the synaptic results onto it, writing it back to the Neuron State Memory.

For a threshold-triggered neuron (such as a SpArNet neuron), once the state is updated, and if the activation condition is met, all the output axons are activated, resulting in the injection of new event(s) into the network. The Axon Memory contains the destination core addresses for the events.

The **data-flow transport of data and control** does not follow the pull-push (read-write) memory access model of conventional von Neumann architectures, thus avoiding the memory accesses that would be required to check for non-occurring updates in a sparse application. As shown in Fig 7, every access to memory occurs in the core where the item is located (**in-place data processing**). Note that this also avoids unnecessary data transport, and only occurs as the result of an event trigger, meaning that it is a change at the producer that triggers update of the consumer.

Our **connection sharing scheme** allows a single neuron output event to fan-out to multiple weighted synapses in the next layer, while only using the bandwidth of one event. Since a connection pattern can be shared by many neurons, the method also reuses Synapses and Axons and is thus particularly efficient for convolution layers. This is done without imposing any constraints on irregular connections, which are still fully supported. Additionally, multi-casting imposes no constraints on the NoC. Connection sharing also supports zero-skipping and multiple weight resolutions.

By using **relative core addressing**, we limit each neuron to address all neurons in the same core or in the neighboring cores. Long-distance communication is still possible by using dummy neurons as intermediate hops. This will reduce the amount of memory we need to store the destination addresses while keeping the architecture scalable to any size. Relative addressing also allows NeuronFlow to be tiled to make a bigger mesh of cores.

Since AI algorithms are evolving fast, flexibility is important. We support both dedicated neuron models and general DFG (Data-Flow Graph) nodes that allow full programmability. Neuron output triggers can use a configurable value threshold, a configurable time-trigger or a programmable trigger. For precision, we employ dynamic fixed-point (similar to [25]), with a configurable scale factor and zero-point.

Some neuron models for SNN need state leakage. NeuronFlow provides a configurable, event-driven leak process, controlled either by a programmable trigger or by a timer.

Neuronflow events are valued, i.e. not simple spikes as usual in neuromorphic systems. In a digital realization, the main transport cost is not the value but the target neuron address, so the relative cost of a valued payload is minor. This is less biologically-accurate, but more efficient for digital circuits.

Out of the box, NeuronFlow supports *Basic arithmetic and logic operations* and *Switch* DataFlow nodes as well as *Leaky Integrate and Fire (LIF)*, *ANN* (for conventional inference) and *SpAr* neurons with *Linear, ReLU, Leaky-ReLU, Clipped ReLU, Hard Sigmoid* and *Hard Tanh* activation functions.

For inter-core synchronization, it supports both autonomous and time-step-driven execution. Time-step driven execution requires all cores to finish one time instant before proceeding to the next, assuming a fixed delay per synapse of one time unit. This synchronization mechanism can be split hierarchically, allowing for either multiple autonomous islands of synchronicity, or for heterogeneous synaptic delay, i.e. the latency in time-steps of an event depends on the mapping of neurons to the fabric. In de-synchronized execution, every core executes events continuously in the order of arrival to the input queue and sleeps if the input queue is empty.

Table I compares two possible hardware realizations of the architecture for different application domains and using different technology nodes, NeuronFlow I and NeuronFlow II, with other architectures. Power numbers are averaged for typical applications. Both realizations have 196 cores, but different memory configurations per core. Larger processors

TABLE I: Comparing two possible Neuronflow configurations with similar architectures

	Neurogrid	TrueNorth	Loihi	NeuronFlow I	NeuronFlow II
Technology	Analog (180nm)	Digital (28nm)	Digital (14mn)	Digital (28nm)	Digital (14nm)
Size of a neuron (μm^2)	2600	430	460	100	25
Memory(bits)/neuron	—	426	2000	120	152
Neurons/chip	65k	1M	131k	200k	4M
Synapses/neuron	1.5k	256 (binary)	1k (shared, 1b)	1k (shared,8b)	4k (shared,8b)
Axons/neuron	—	1	4k (shared)	1k (shared)	16k (shared)
Energy/synaptic-operation	100pJ	25pJ	80pJ	20pJ	10pJ
On-chip Learning	No	No	Yes	No	No

can easily be built through tiling. The power consumption numbers are reported by Spyglass tools from Synopsys.

GrAI Matter Labs also provides an SDK to program the chip and directly map onto it neural networks trained in TensorFlow. GrAIFlow provides performance numbers (usage of cores, number of events per core during run time, estimated power consumption, debugging tools, etc.) by performing mapping and simulating hardware behavior.

REFERENCES

[1] J. W. Mink, R. J. Blumenschine, and D. B. Adams, "Ratio of central nervous system to body metabolism in vertebrates: its constancy and functional basis," *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, vol. 241, no. 3, pp. R203–R212, 1981.

[2] D. Grady, "The vision thing: Mainly in the brain," *Discover*, Jun 1993.

[3] D. Fick, "Mythic @ hot chips 2018," https://medium.com/mythic-ai/mythic-hot-chips-2018-637dfb9e38b7, 2018.

[4] G. K. R Quian Quiroga, "Measuring sparseness in the brain: comment on bowers (2009)," *Psychol Rev*, 2010.

[5] R. S. Shy Shoham, Daniel H. OConnor, "How silent is the brain: is there a dark matter problem in neuroscience?" *Journal of Comparative Physiology A*, 2006.

[6] A. Yousefzadeh, M. Jaboski, T. Iakymchuk, A. Linares-Barranco, A. Rosado, L. A. Plana, S. Temple, T. Serrano-Gotarredona, S. B. Furber, and B. Linares-Barranco, "On multiple aer handshaking channels over high-speed bit-serial bidirectional lvds links with flow-control and clock-correction on commercial fpgas for scalable neuromorphic systems," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 5, pp. 1133–1147, Oct 2017.

[7] B. Moons, K. Goetschalckx, N. V. Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, Oct 2017, pp. 1921–1925.

[8] K. H. Choi, D. Jung, S. W. Kim, T. h. Hwang, J. s. Kwon, and D. S. Kim, "Study of gpu scalability," in *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, 2014, pp. 1–2.

[9] J. Schemmel, A. Grbl, S. Hartmann, A. Kononov, C. Mayr, K. Meier, S. Millner, J. Partzsch, S. Schiefer, S. Scholze, R. Schffny, and M. Schwartz, "Live demonstration: A scaled-down version of the brainscales wafer-scale neuromorphic system," in *2012 IEEE International Symposium on Circuits and Systems*, May 2012, pp. 702–702.

[10] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, Feb 2018.

[11] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[12] P. A. Merolla and et. al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[13] M. Davies and et. al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[15] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor," pp. 2849–2856, June 2008.

[16] L. A. P. et al., "An on-chip and inter-chip communications network for the spinnaker massively-parallel neural net simulator," in *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, April 2008, pp. 215–216.

[17] A. Yousefzadeh, M. Soto, T. Serrano-Gotarredona, F. Galluppi, L. Plana, S. Furber, and B. Linares-Barranco, "Performance comparison of time-step-driven versus event-driven neural state update approaches in spinnaker," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–4.

[18] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019.

[19] A. Yousefzadeh, M. A. Khoei, S. Hosseini, P. Holanda, S. Leroux, O. Moreira, J. Tapson, B. Dhoedt, P. Simoens, T. Serrano-Gotarredona, and B. L. Barranco, "Asynchronous spiking neurons, the natural key to exploit temporal sparsity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2019.

[20] A. Yousefzadeh, S. Hosseini, P. Holanda, S. Leroux, T. Werner, T. Serrano-Gotarredona, B. L. Barranco, B. Dhoedt, and P. Simoens, "Conversion of synchronous artificial neural network to asynchronous spiking neural network using sigma-delta quantization," *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019.

[21] L. Cavigelli, P. Degen, and L. Benini, "Cbinfer: Change-based inference for convolutional neural networks on video data," *CoRR*, vol. abs/1704.04313, 2017.

[22] P. O'Connor and M. Welling, "Sigma delta quantized networks," *ICLR*, 2017. [Online]. Available: https://openreview.net/forum?id=HkNRsU5ge

[23] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "Deltarnn: A power-efficient recurrent neural network accelerator," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: ACM, 2018, pp. 21–30.

[24] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[25] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *CoRR*, vol. abs/1712.05877, 2017.