A Multi-threading Kernel for Enabling Neuromorphic Edge Applications

Lars Niedermeier*‡, Vyom Shah†, and Jeffrey L. Krichmar†‡
*Niedermeier Consulting, Zurich, ZH, Switzerland
†Department of Computer Science, University of California, Irvine, CA, USA
‡Department of Cognitive Sciences, University of California, Irvine, CA, USA
Correspondence Email: lars@niedermeier-consulting.ch

***Abstract*—Spiking Neural Networks (SNNs) have sparse, event-driven processing that can leverage neuromorphic applications. In this work, we introduce a multi-threading kernel that enables neuromorphic applications running at the edge, meaning they process sensory input directly and without any up-link to or dependency on a cloud service. The kernel shows speed-up gains over single thread processing by a factor of four on moderately sized SNNs and $1.7X$ on a Synfire network. Furthermore, it load-balances all cores available on multi-core processors, such as ARM, which run today's mobile devices and is up to 70% more energy efficient compared to statical core assignment. The present work can enable the development of edge applications that have low Size, Weight, and Power (SWaP), and can prototype the integration of neuromorphic chips.**

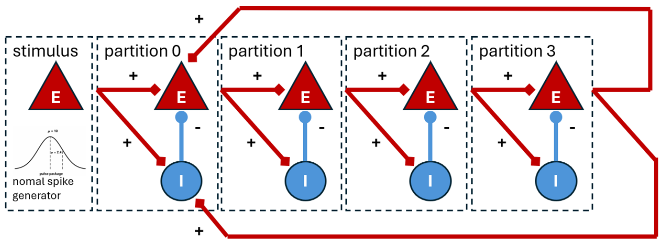
***Index Terms*—Edge Computing, Neuromorphic Applications, Spiking Neural Networks**

I. Introduction

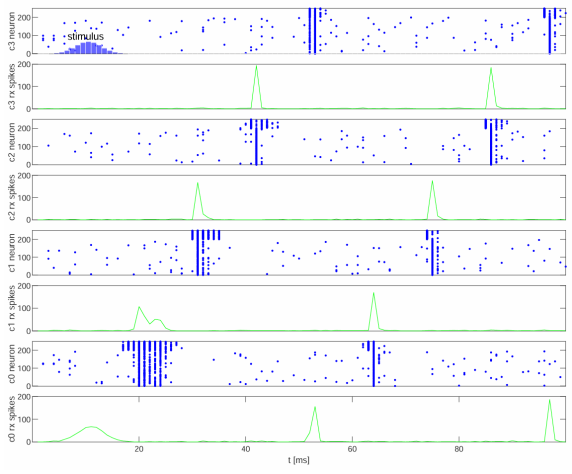
Spiking Neural Networks (SNNs) mimic natural nervous systems with elements that replicate the sparse, all-or-none neural activity. This makes them a good fit to take advantage of low Size, Weight, and Power (SWaP) computing systems. The downside of SNNs are the higher computational costs for the numerical solution of the differential equations that determine the neuron's state. Software packages such as CARLsim have evolved over the years to provide a mature framework for computational neuroscientists, embedded systems engineers, and roboticists [1].

To measure, the potential of SNNs for neuromorphic edge applications, methods are needed to quantitatively measure performance. Recently, a multilayered recurrent SNN benchmark, called a Synfire chain, was used to measure the energy efficiency of the SpiNNaker2 neuromorphic chip [2]. The Synfire network, which consisted of leaky-integrated-and-fire (LIF) spiking neurons, is used to measure the dynamic voltage and frequency scaling (DVFS) developed for SpiNNaker [3].

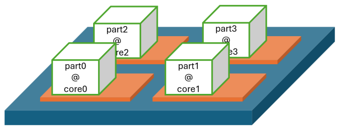
In the present work, we investigate the Synfire benchmark with CARLsim and the Izhikevich neuron model [4], [5]. Compared to LIF neurons, the Izhikevich neuron has a wider range of dynamics and can mimic a variety of neuron types found in the brain. Fig. 1 presents the Synfire network developed to benchmark CARLsim. The excitatory groups E consist of regular spiking (RS) neurons found as pyramidal cells in the cortex, the inhibitory groups I of fast spiking (FS) interneurons. The network is segmented in four partitions that are assigned to a dedicated core. We follow [6] in connecting the last segment $partition3$ to the first $paritition0$ in contrast to the original referenced Synfire network is a strict feed-forward-inhibition (FFI) network [7].



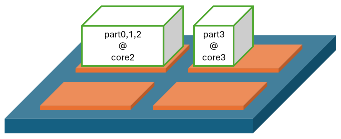
(a) Synfire network as CARLsim kernel benchmark.



(b) Spikewave propagation with correlated inhibition.



(c) PThreads kernel (CARLsim 4).



(d) New multi-threading kernel.

Fig. 1. The CARLsim *Synfire* network with normal spike generator for the stimulus. (a) Izhikevich neurons in four partitions recurrently linked. (b) Neural activity visualized in CARLsim spike monitor. (c) Paritions with fixed core affinity in pthreads kernel of CARLsim4. (d) The new kernel is more energy efficient by assigning cores dynamically to free up SoC compute resources.

Depending on the network size, efficient operation may require parallel processing on GPUs, multi-core CPUs, or neuromorphic hardware. For instance, *small* networks up to a few hundred neurons run most efficiently in a single thread on a CPU. The efficient simulation of *large-scale* networks with millions of neurons, as typically used in computational neuroscience [8], are the core feature of CARLsim as its PThreads based kernel scales over multiple GPUs [9]. In the case of *mid-size* networks (e.g., 10^3 neurons), which are often used for edge applications, the overhead for PThreads has poor performance compared to an execution on a single thread.

In the present work, we introduce a multi-threading kernel that scales efficiently over the available cores of modern CPUs used in SoCs such as ARM Cortex-76 in Raspberry Pi 5. This addresses the performance limitation of PThreads in CARLsim and other SNN simulators. All code and models are open-source and available on [10]. The main contributions of this work are:

1) ***Multi-threading for SNNs.*** A multi-threading kernel that scales independently of the partitioning of the network.
2) ***Dynamic load balancing.*** A load-balancing algorithm that dynamically allocates computation to cores and avoids synchronization bottlenecks of idle threads.
3) ***Performance monitoring.*** An SNN performance monitor with ms precision.
4) ***Synthetic load network.*** A synthetic load network named *Chainfire* that produces and measures neural and synaptic activity.
5) ***SNN Benchmarks.*** Concrete benchmark results on Intel and ARM processors.
6) ***Edge processing.*** SNNs that fulfill real-time requirements on off-the shelf mobile processors, without the need of specialized neuromorphic hardware.

II. Methods

A. Spiking neuron model

CARLsim efficiently implements spiking neural networks such as LIF and the Izhikevich neuron model with 4 and 9 parameters [1]. In contrast to [2], we implemented the Synfire network utilizing Izhikevich 4-parameter model described by the following equations [4].

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I \quad (1)$$

$$\dot{u} = a(bv - u) \quad (2)$$

$$\text{if } v \geq 30 \begin{cases} v = c \\ u = u + d \end{cases} \quad (3)$$

The present simulations use Forward Euler for numerical integration. More complicated neuron models with multiple compartments may require other numerical methods, such as Runge-Kutta, to handle instabilities. The CARLsim kernel is designed to handle these cases.

B. Synfire network architecture

We utilize the Synfire chain network as a benchmark to measure and compare performance of CARLsim and potentially other neuromorphic chips. We follow Höppner et. al. in their approach for SpiNNaker [2] and built a network with the same structure and sizing, see Fig. 1. The excitatory groups E has 200 regular spiking (RS) neurons $(a = 0.02, b = 0.2, c = -65, d = 8)$, and the inhibitory groups consist of 50 fast spiking (FS) neurons $(a = 0.1, b = 0.2, c = -65, d = 2)$. The code and configuration is Open Source and available in the CARLsim GitHub repository [10]. Using the CARLsim multi-thread kernel, we replicated Synfire results for current based (CUBA) and conductances based (COBA) synapses. See Fig. 3db and GitHub repository [11] for the benchmark runs and Subsection II-B.

C. Chainfire network architecture

In addition to using the Synfire chain SNN for benchmarking, we created a Chainfire network, which could more readily control parameters that affect CPU loads. Fig. 2 shows the synthetic load network *Chainfire* with the minimal amount of synapses that allows to produce and measure arbitrary loads, specifically targeted at the compute-intensive status updates of neurons, defined by equations 1 - 3. The network is designed to generate loads similar to the *Synfire* network and therefore has four clusters, that are marked in blue. This allows validation, tuning, scaling of the parallelization, and provides the maximum of performance improvement possible by this kernel.

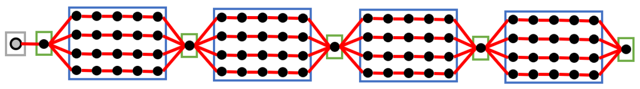


Fig. 2. *Chainfire* SNN for validation, tuning, and scaling of the parallelization. The blue rectangles contain groups of excitatory neurons. It can induce specific loads on the multi-threading kernel. The green squares depict synchronization neurons that dictate the fan-in and fan-out between neuron groups.

The parallel rows are chains of excitatory RS neurons with specific delays and weights (e.g. d_{exc} = 5 ms, w_{exc} = 0.432), configured to propagate the spike wave with minimal delay and without noise. The synchronization neurons, indicated by a green frames, are the also excitatory RS neurons and consolidate the fan-in for predecessor group or integrate the fan-out.

The feed-forward load generator SNN is configurable by the following parameters: N total number of neurons per cluster, d ms of delay for pre-synaptic to post-synaptic neuron in the parallel chains that are sequential connected, and the *span* in ms that determine the length of the chain, e.g. $N = 100$, $d = 20$ms, $span = 100$ms results in the shown network groups with 100 neurons, structured in 4 rows (parallel chains) and five columns.

The fan-in from synchronization neurons are configured, so that a spike activates the input layer (first column) at the same time. The spike-wave travels through the group and activates each neuron of the adjacent column. The output layer (last column) transmits the spikes to the fan-out which has reduced weights, so that the synchronization neurons produce a single spike. This in turn feeds back to the first column of the Chainfire.

The frequency of the input spike generator is 1 Hz. The delays enforce a travel time of the spike-wave of around 500 ms (400 ms for the cluster plus inter-neurons, and some processing time around 2 ms for the RS neurons). The spike-wave travels through the network without interference to the last synchronization neuron, before the next stimuli is induced.

D. *Benchmark processors*

SNNs running at the edge can take advantage of off-the-shelf multi-core processors. The CARLsim multi-threaded kernel utilizes available cores of modern energy efficient CPUs, such as the ARM Cortex-76 in Raspberry Pi 5. These processors are typically built in System-on-chips (SoCs), and are used to operate devices at the edge. For a moderate sized SNN to run efficiently at the edge, care must be taken to ensure all cores have fully balanced loads. Algorithm 1 outlines parallelization used in the multi-threaded kernel.

Algorithm 1: Multi-threading kernel for neuron state updates.

```
Input: simulation time in ms, network partition $net_{id}$, numerical integration
       steps per ms $S$, core threads $T$
Output: spikes
1 while $step \in S$ do                              /* numerical integration */
2     while $group \in partion$ do                   /* initialize shared */
3         while $neuron \in group$ do                /* in parallel $T$ */
4             $I_{sum}$ = CUBA/COBA;                 /* synaptic fan-in */
5             $v, u$ = EULER/RK4$(dudt, dvdt, a, b, c, d, I_{sum})$
6             if $v > 30$ then                       /* Izhikevich */
7                 $spike = true$;
8             $runtimeData.recovery[lNId] = u$;      /* update */
```

E. *Load balancing by dynamic core assignment (DCA)*

One reason SNNs are energy efficient is that their activity is sparse with a firing rate of a few Hz with occasional spike bursts. CARLsim utilizes the intrinsic Dynamic Voltage and Frequency Scaling (DVFS) of modern CPUs, which provide advanced energy policies. Intel no longer recommends direct manipulation of the P-states to manipulate DVFS and recommends only to use power profiles for the sake of system stability.

The overhead for thread synchronization cannot be neglected. Consequently, our load-balancer allocates only the minimum cores necessary to fulfill the performance criteria, for instance real-time, meaning 1 ms in the SNN corresponds to 1 ms wall clock time.

III. Results

We provide benchmark results for the Intel i9 with 8 cores and several ARM Cortex processors with 4 cores. All results can be reproduced with the open source of the CARLsim repository. Furthermore, we provide supplemental material such as videos and log of the run on GitHub [11].

A. *Performance gain by multi-threading on Chainfire*

We measure Chainfire performance on a release build and a version with debugging enabled. Fig. 3a and Tables I and II present the performance of a Chainfire network with 2000 neurons run on a Intel i9 desktop processor. The simulation model time was 10s. The overall spike count is $20,040$ and applied as a measure to determine that the neural activity is same for all (parallel) runs. The $t_{execution}$ (t_{real}) is the wall clock time, which is implemented by the C++ standard library (STL) *std::chrono::steady_clock*. Performance is measured by a *speed_factor*, defined as $t_{model}/t_{execution}$. For example, if the simulation of the SNN with eight threads runs in 2.28s, it has a $speed\_factor = 4.4$. If the single threaded run takes 8.88s, it a the $speed\_factor = 1.1$. The resulting performance gain is then $4.4/1.1 = 4.0$.

TABLE I
PERFORMANCE IMPROVEMENTS CHAINFIRE (RELEASE BUILD)

Threads	Execution Time	Speed Factor	Performance Gain
1	8.88 s	1.1 x	
2	5.20 s	1.9 x	1.7 x
4	3.09 s	3.2 x	2.9 x
8	2.38 s	4.2 x	3.8 x
16	**2.28 s**	**4.4 x**	**4.0 x**
32	5.87 s	1.7 x	1.5 x

TABLE II
PERFORMANCE IMPROVEMENTS CHAINFIRE (DEBUG BUILD)

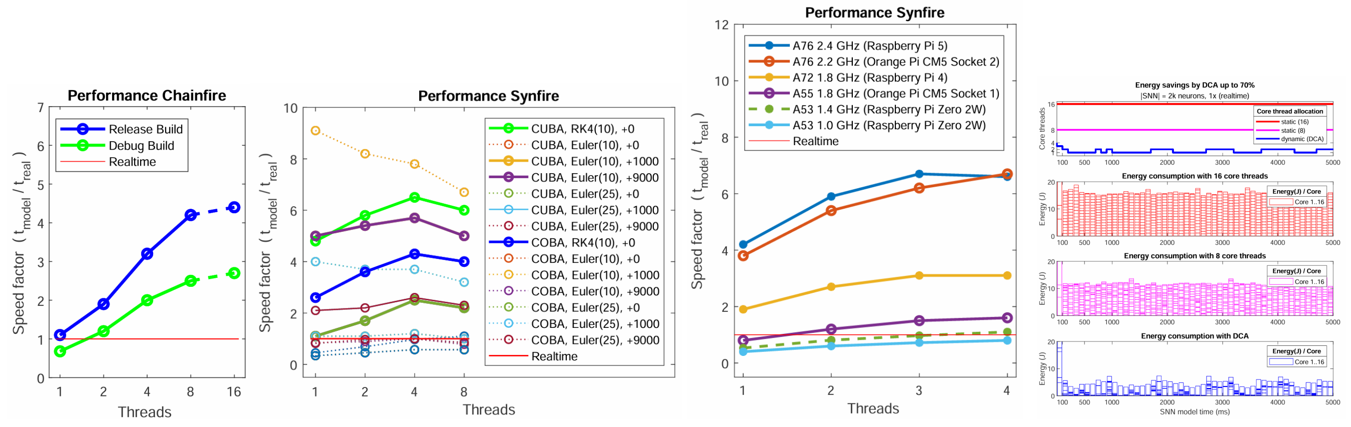
Threads	Execution Time	Speed Factor	Performance Gain
1	14.82 s	67.5%	
2	8.49 s	1.2 x	1.8 x
4	4.93 s	2.0 x	3.0 x
8	3.96 s	2.5 x	3.7 x
16	**3.72 s**	**2.7 x**	**4.0 x**
32	5.87 s	1.7 x	1.9 x

B. *Performance gain by multi-threading on Synfire*

Fig. 3b and Table III present the performance improvement on the Synfire chain network with 1,200 neurons and 77K synapses run on a Intel i9 desktop processor. As expected, the performance gain is lower than on the synthetic SNN as the optimization for the multi-threading kernel aimed primarily on the compute intense numerical integration of the differential equations for the neuron state. On the other hand, the network is rather small and 70% performance gain is actually above expectations. Future work will aim to parallelize the synaptic processing.

TABLE III
PERFORMANCE IMPROVEMENTS SYNFIRE (RELEASE BUILD)

Threads	Execution Time	Speed Factor	Performance Gain
1	0.43 s	7 x	
2	0.31 s	9.7 x	1.4 x
4	**0.26 s**	**12.0 x**	**1.7 x**
8	0.26 s	12.0 x	1.7 x
16	0.29 s	10.0 x	1.4 x
32	0.62 s	4.8 x	0.7 x



(a) Chainfire on Intel i9 (b) Synfire on Intel i9 (c) Synfire on ARM Cortex (d) DCA

Fig. 3. Performance improvements by multi-threading kernel over allocated cores. (a) Chainfire (2k neurons) on 11th Gen Intel Core i9-11900K @ 3.5 GHz (8 cores, 16 logical). (b) Synfire (1.2k neurons, 77k synapses) on Intel i9. (c) Synfire on several ARM Cortex processors. (d) Dynamic core allocation saves up to 70% energy as overhead for over-allocated cores thread synchornization is avoided. The performance monitor Intel PCM shows this at ms precision at timeline of the SNN.

Fig. 3c presents the the performance improvements of the same Synfire network on several ARM processors that can be used at the edge. The kernel has a similar structural performance scaling on ARM as on Intel, as long as the cores are used on Intel. Because two Intel *logical processors* share a core, the performance degrades, which is indicated by the dashed line in Fig. 3b.

C. *Energy savings by DCA*

Fig. 3d shows that DCA reduces the core threads as long as the real-time criteria are met (first 250 ms). When the system load demands, DCA allocates additional core threads (e.g. at 750 ms and 1800 ms), respectively frees them, when no longer needed (e.g. at 750 ms and 1800 ms).

IV. Conclusion

The present work introduces a multi-threading kernel for SNNs. It especially optimizes moderately sized SNNs deployed on the multicore processors used for mobile devices. The kernel supports dynamic core allocation to ensure efficient processing across SoC devices. We show impressive performance gains on a network architecture, Synfire chain, which is commonly used to test SNNs. We also introduce a Chainfire network to further evaluate performance.

The synthetic Chainfire network is instrumental to understand the performance relevant internal workings of the former multi-threading kernel. It also enabled to validate and quantitative measure the optimization, see Fig. 3a. Without it, optimization information is hidden behind noise. There are great tools available for performance profiling. However, to identify bottlenecks, it was essential to produce specific loads, for instance the status update only without interference of the synaptic processing.

With the multi-threading kernel extensions, CARLsim now supports mid-size SNNs on modern multicore processors such as the ARM Cortex family. Compared to neuromorphic chips such as Intel's Loihi or Brainchip's Akida, which are highly specialized on neural processing and usually require a host system to run the base application, CARLsim utilizes the existing compute capacity of the mobile processor. Depending on the use case, similar energy efficiency on these specialized neuromorphic chips might be achieved with CARLsim on mobile processors. The new power saving policies in CARLsim may make the intrinsic energy efficiency of SNNs even more attractive for edge applications. It makes neuromorphic applications possible without additional hardware costs. Furthermore, an SNN executed by software is much more flexible in regards of changes and use case adaptation.

The present work opens the way for a new generation of neuromorphic applications that can be deployed on millions of mobile devices, such as wearables like the Samsung Watch Ultra 2025 or embedded devices based on SoCs such as the Raspberry Pi 5.

References

[1] L. Niedermeier, K. Chen, J. Xing, A. Das, J. Kopsick, E. Scott, N. Sutton, K. Weber, N. Dutt, and J. L. Krichmar, "Carlsim 6: An open source library for large-scale, biologically detailed spiking neural network simulation," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–10.

[2] S. Höppner, Y. Yan, A. Dixius, S. Scholze, J. Partzsch, M. Stolba, F. Kelber, B. Vogginger, F. Neumärker, G. Ellguth, S. Hartmann, S. Schiefer, T. Hocker, D. Walter, G. Liu, J. Garside, S. Furber, and C. Mayr, "The spinnaker 2 processing element architecture for hybrid digital neuromorphic computing," 2022. [Online]. Available: https://arxiv.org/abs/2103.08392

[3] S. Höppner, Y. Yan, B. Vogginger, A. Dixius, J. Partzsch, F. Neumärker, S. Hartmann, S. Schiefer, S. Scholze, G. Ellguth *et al.*, "Dynamic voltage and frequency scaling for neuromorphic many-core systems," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[4] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, 2003.

[5] ——, "Which model to use for cortical spiking neurons?" *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1063–1070, Sep. 2004.