Harnessing Manycore Processors with Distributed Memory for Accelerated Training of Sparse and Recurrent Models

Jan Finkbeiner[1, 2], **Thomas Gmeinder**[3], **Mark Pupilli**[3], **Alexander Titterton**[3], **Emre Neftci**[1, 2]

[1]Research Center Juelich
[2]RWTH Aachen University
[3]Graphcore

j.finkbeiner@fz-juelich.de, thomasg@graphcore.ai, markp@graphcore.ai, alexandert@graphcore.ai, e.neftci@fz-juelich.de

Abstract

Current AI training infrastructure is dominated by single instruction multiple data (SIMD) and systolic array architectures, such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), that excel at accelerating parallel workloads and dense vector matrix multiplications. Potentially more efficient neural network models utilizing sparsity and recurrence cannot leverage the full power of SIMD processor and are thus at a severe disadvantage compared to today's prominent parallel architectures like Transformers and CNNs, thereby hindering the path towards more sustainable AI. To overcome this limitation, we explore sparse and recurrent model training on a massively parallel multiple instruction multiple data (MIMD) architecture with distributed local memory. We implement a training routine based on backpropagation through time (BPTT) for the brain-inspired class of Spiking Neural Networks (SNNs) that feature binary sparse activations. We observe a massive advantage in using sparse activation tensors with a MIMD processor, the Intelligence Processing Unit (IPU) compared to GPUs. On training workloads, our results demonstrate 5-10$\times$ throughput gains compared to A100 GPUs and up to 38$\times$ gains for higher levels of activation sparsity, without a significant slowdown in training convergence or reduction in final model performance. Furthermore, our results show highly promising trends for both single and multi IPU configurations as we scale up to larger model sizes. Our work paves the way towards more efficient, non-standard models via AI training hardware beyond GPUs, and competitive large scale SNN models.

Introduction

The hardware lottery hypothesis (Hooker 2021) contends that the current dominance of Graphics Processing Units (GPUs) in the AI hardware market constrains the innovation of model architectures. As a result, contemporary research in Artificial Intelligence (AI) and Machine Learning (ML) is predominantly driven by parallel, dense architectures that fully leverage the processing power of GPUs. However, this situation restricts the exploration of alternative approaches with great potential, such as neuromorphic computing, due to the inherent inefficiency of such architectures on GPUs.

In response to the limitations of traditional computing paradigms, neuromorphic computing has emerged as a promising alternative to achieve the efficiency of the human brain (Mead 1990; Chicca, Stefanini, and Indiveri 2013; Benjamin et al. 2014; Friedmann et al. 2017; Davies et al. 2018; Frenkel and Indiveri 2022). Distributed local memory, where processing and memory are closely intertwined is a key characteristic of brains, neuromorphic hardware and physical computing in general. In the brain, synapses both store information and perform computation, resulting in no additional necessary data transfer as prevalent in today's von Neumann architectures. The other two core characteristics of the brain are recurrence and sparsity (Friston 2008). Recurrence enables the reuse of locally stored information and weights, thereby minimizing data transfer. Data transfer across processors is the most costly operation in modern technology (Horowitz 2014). Reducing transfer via sparsity can lead to significant energy efficiency and throughput gains. Together, networks mimicking these brain-like characteristics can pave the way towards an energy-efficient and sustainable future for large-scale model deployment. In this article, we demonstrate an architecture emphasizing these characteristics, namely local distributed memory and a novel algorithm for training networks featuring sparsity and recurrence as in the brain that vastly outperforms their equivalent implementations in GPUs in throughput. GPUs (and TPUs (Jouppi et al. 2017)), the prevalent hardware for deep learning, are massively parallel single instruction multiple data (SIMD) architectures, optimized for processing large chunks of data and executing identical operations in parallel. However, networks featuring recurrence and sparsity and thereby sequential computations, cannot fully utilize the parallel processing power of GPUs. Using a multiple instruction mutliple data (MIMD) manycore processor with distributed in-processor memory, we demonstrate speed-ups reaching 20$\times$ on training tasks for especially challenging network architectures featuring both recurrence and dynamic sparsity, namely spiking neural networks (SNNs).

SNNs are a class of neuromorphic algorithms that emulate key computational principles of biological neurons. Unlike traditional Artificial Neural Networks (ANNs), which output continuous-valued activations and redundant computations, SNNs transmit information through discrete spike events, potentially resulting in significant energy savings (Zenke et al. 2021). Their sensor counterpart, *i.e.* event-based cameras, departs from fixed-time interval sampling,

responding asynchronously to changes in stimuli, leading to reduced data redundancy and ultra-low data rates (Gallego et al. 2019). This feature has proven ground-breaking for applications requiring high temporal resolution, such as robotics (control) and agile drone flight (Rosinol et al. 2018). SNNs are ideally suited to process such event-based sensor data.

Still, training large-scale SNN models and generally models featuring recurrence and sparsity remains prohibitively expensive due to the inadequacies of widely used training hardware.

To address these constraints and foster the exploration of alternative hardware architectures, this work focuses on a massively parallel multiple instruction multiple data (MIMD) architecture with distributed local memory, the Intelligent Processing Unit (IPU). By distributing neurons on dedicated tiles of the IPU and by implementing binary sparse representations for the SNN's sparse spike activation vector, we take advantage of the IPU's memory locality and efficient fine-grained memory access. This way we both minimize data transfer within and between IPUs and maximize the computational efficiency of the implementation. By examining the IPU's capabilities, we aim to unlock new avenues for AI and ML research, breaking free from the hardware lottery hypothesis and opening the door to innovative, efficient, and biologically plausible computing systems.

Related Work

Sparse neural networks have garnered substantial interest for their potential to enhance both training and inference efficiency. A multitude of research efforts have been directed towards developing algorithmic and hardware approaches to harness sparsity's benefits (Hoefler et al. 2021). However, achieving substantial gains in throughput or training time on SIMD and systolic array based architectures like GPUs and TPUs remains challenging. Notably, the SIMD nature of GPUs, including the incorporation of Tensor Cores, presents limitations for efficiently accelerating sparse operations (Gale et al. 2020; Huang et al. 2022). Consequently, achieving benefits from sparse matrix matrix multiplication (SpMM) in the absence of specific sparsity patterns proves difficult, except for extreme sparsity levels of around 99% and higher (Hoefler et al. 2021). To address this, techniques have emerged to introduce and exploit structure within sparse matrices (Huang et al. 2022; Mishra et al. 2021; Wang 2020). For instance, NVIDIA's 2:4 sparsity support optimally utilizes tensor cores (Mishra et al. 2021). Additionally, strategies like the Inspector-Executor Framework and Autotuning (Wang 2020) as well as other techniques to optimize data layouts for memory access and load balancing (Gale et al. 2020) have been employed.

While existing techniques predominantly focus on inference acceleration using static sparsity patterns in pruned models, dynamic sparsity patterns are also gaining attention. These encompass evolving weight sparsity during training or rapid changes in activation-based sparsity, particularly in event-driven recurrent architectures (Subramoney et al. 2023; Knight and Nowotny 2022; Eshraghian et al. 2021). However, only a few works effectively leverage sparse activations for throughput gains (Knight and Nowotny 2022; Perez-Nieves and Goodman 2021). Recurrent neural network architectures, such as LSTM, GRU, and SNNs, face challenges due to their recursive execution compared to inherently parallel models like Transformers. Specific efforts enable parallel timesteps execution via customized CUDA kernels while restricting SNN architectures to feed-forward structures (Shrestha and Orchard 2018; Bauer et al. 2023; Fang et al. 2020).

For neuroscience-driven spiking neural network simulations with binary activations and no gradient computation, scalable GPU and CPU simulators have been developed (Yavuz, Turner, and Nowotny 2023; Knight and Nowotny 2023; Niedermeier et al. 2022; Golosio et al. 2021; Gewaltig and Diesmann 2007). Beyond conventional hardware, neuromorphic hardware aims to enable power efficient and fast execution of brain-inspired recurrent and sparse models, specifically SNNs. However, these hardware architecutres are mainly geared towards inference of SNN models or brain inspired learning rules that rely on local information. Prominent examples include TrueNorth(Merolla et al. 2014), Loihi (Davies et al. 2018), Spinnaker (Furber et al. 2014), and the mixed-signal Brainscales (Friedmann et al. 2017) architecture. Several FPGA implementations have been realized to accelerate both biological SNN simulations (Kauth et al. 2023; Wang, Thakur, and Van Schaik 2018) as well as CNN applications with sparse activations (Aimar et al. 2019).

Prior exploration of SNN behavior on the IPU has produced inconclusive results in comparison to GPU or TPU implementations: Biologically inspired neuron dynamics implementations on various accelerators reveal non-favorable IPU performance (Landsmeer et al. 2023). Another study (Sun et al. 2022) demonstrates enhanced throughput for single hidden layer architectures, but its results for more complex designs remain unclear. Notably, these studies do not optimize code beyond IPU compilation, neglecting the potential of tile-locality and sparse activation tensors for given SNN models, and were limited in scale. In contrast, our work focuses on exploiting the unique IPU architecture to take advantage of recurrence dynamic activation sparsity. Thereby we capitalizes on both tile-locality and sparse activation tensors, enhancing performance via a dedicated sparse matrix multiplication algorithm. Additionally this renders our implementation more memory efficient allowing the deployment of larger scale models.

Summarizing, our contributions are the following:

- As a model for sparse and recurrent architectures, we implement a sparse SNN training algorithm based on BPTT on a manycore processor, the IPU.
- We explicitly capitalize on the hardware's in-processor memory by distributing neurons in a tile-local fashion leading to acceleration factors of up to 20× compared to GPU implementations.
- We verify that our sparse training algorithm maintains final accuracies and convergence rates on-par with the dense equivalent.
- We demonstrate the scalability of our implementation to larger model sizes and mulit-IPU implementations.

Methods

The Graphcore IPU

The Intelligence Processing Unit (IPU) (Jia et al. 2019) by Graphcore is a massively parallel compute architecture with distributed local memory ideally suited for multiple data multiple instruction (MIMD) workloads.

The IPU programming paradigm based on the Bulk Synchronous Parallel (BSP) model features the sequential execution of multiple supersteps composed of a local computation phase, a communication phase, and a barrier synchronization phase. During each computation phase, every core has access to 624kB of its dedicated SRAM. The core and its dedicated SRAM together form a *tile*.

In this work, the IPU is programmed via the C++ based Poplar SDK for fine-grained control over the computational graph and memory allocation. Additionally a Python API via all major machine learning libraries (Pytorch (Paszke et al. 2019), Tensorflow (Abadi et al. 2015), ONNX, Jax (Bradbury et al. 2018)) gives simple and flexible access to the IPU.

The IPU features low latency, high bandwidth intra- and inter-IPU interconnect. This feature becomes especially useful when designing large scale, distributed networks.

The IPU architecture has the potential to provide improvements over GPU and TPU architectures for recurrent neural network architectures with sparse connectivity and activations. This is because sequential memory read and write operations between computation operations are faster on the IPU as memory latency is extremely low (equivalent to L1 caches on GPUs). Furthermore, the IPU can efficiently utilize unstructured sparsity compared to GPUs, as it allows for efficient reading and writing of small data packets (8 bytes for highest bandwidth) compared to the minimal data read size of 128 bytes (32 values of 32 bits) for NVIDIA GPUs. Therefore we believe it to be especially well suited for SNNs which combine both recurrence and sparsity.

Spiking Neural Network (SNN) Training

SNNs can be viewed as special cases of recurrent neural networks with binary and sparse activations and a specific set of internal dynamics (Neftci, Mostafa, and Zenke 2019). Additionally, complex connectivity structures like explicit recurrence where higher levels feed back to lower layers are more common in brain-inspired networks and SNN architectures (Kubilius et al. 2019) than for standard RNN architectures, like GRUs or LSTMs, which are typically implemented as feed-forward architectures with local recurrence.

The spiking neuron dynamics can be described by various sets of differential equations. In this work we chose the standard leaky-integrate-and fire (LIF) neuron model, though our approach can be easily generalized to more complex neuron dynamics. Equations (1-3) below show the time-discretized versions of the LIF equations that are implemented in our models:

$$I_i[t+1] = f(I_i[t], \mathbf{S}^{\text{in}}[t]) = \sum w_{ij}\ S_j^{\text{in}}[t], \tag{1}$$

$$u_i[t+1] = \alpha\ u_i[t](1 - S_i^{\text{out}}[t]) + (1-\alpha)\ \frac{1}{C}\ I_i[t], \tag{2}$$

$$S_i^{\text{out}}[t] = \Theta(u_i[t] - \vartheta_i). \tag{3}$$

Equation (1) describes the calculation of the input current I_i of neuron i. Generally, I_i can be formulated as a function of the current at the previous timestep and the spikes. The synaptic weights w_{ij} are the adjustable and trainable parameters. Equation (2) describes the dynamics of the LIF-neuron's membrane potential u_i including a reset mechanism that resets the membrane potential to zero after a spike S_i^{out} was emitted.

Equation (3) defines the spike generation function modeled via the Heaviside function Θ. The objective function can be minimized with respect to the synaptic weights by using stochastic gradient descent (SGD) generally result in good model performance when training SNNs (Neftci, Mostafa, and Zenke 2019). The calculation of gradients across the non-differentiable spiking activation function 3 was enabled by the surrogate gradient approach (Zenke and Ganguli 2018; Neftci, Mostafa, and Zenke 2019). The surrogate gradient approach introduces a separate function to overwrite the spiking activations behavior in the backward pass for the gradient computation. However, it does not alter the behavior in forward pass. Different choices for such surrogate functions exist (Neftci, Mostafa, and Zenke 2019) with mostly similar performance (Zenke and Vogels 2020). Here, we choose the SuperSpike surrogate function (Zenke and Ganguli 2018) as defined in Equation 5.

$$S(x) = \Theta(x), \tag{4}$$

$$\frac{\partial S(x)}{\partial x} = h(x) = \frac{1}{(\beta|x|+1)^2}. \tag{5}$$

Using backpropagation through time (BPTT) the gradients of the loss with respect to the weights $\frac{\partial L}{\partial w_{ij}}$ can be calculated iteratively backwards through time, which enables the use of standard neural network optimizers. In this work either Adam (Kingma and Ba 2014) or plain stochastic gradient descent (SGD) is used.

Exploiting SNN Sparsity on the IPU

We implemented a BPTT based training routine for multi-layer spiking neural networks (SNNs) following Equations (1-3) on the Graphcore IPU and use sparse spiking vectors together with a sparse matrix multiply algorithm that utilizes the IPU's MIMD nature and ability of fine-grained low latency memory access. The IPU's design featuring distributed in-processor memory with parallel processes fits well with the brain's principle of distributed local computation and therefore with the deployment of SNNs in neuromorphic hardware.

Distributed Neuron Allocation As illustrated in Figure 1, the IPU's distributed in-processor memory allows allocating neurons on a dedicated tile for the whole training process, which includes the simulation of their dynamics, the synaptic weights and weight gradients and the input current calculation. Only the spike tensors and auxiliary data during their computation have to be communicated between tiles. This drastically reduces data transfer on the chip compared to other architectures which rely on external memory, like GPUs or TPUs, and thereby improves efficiency.

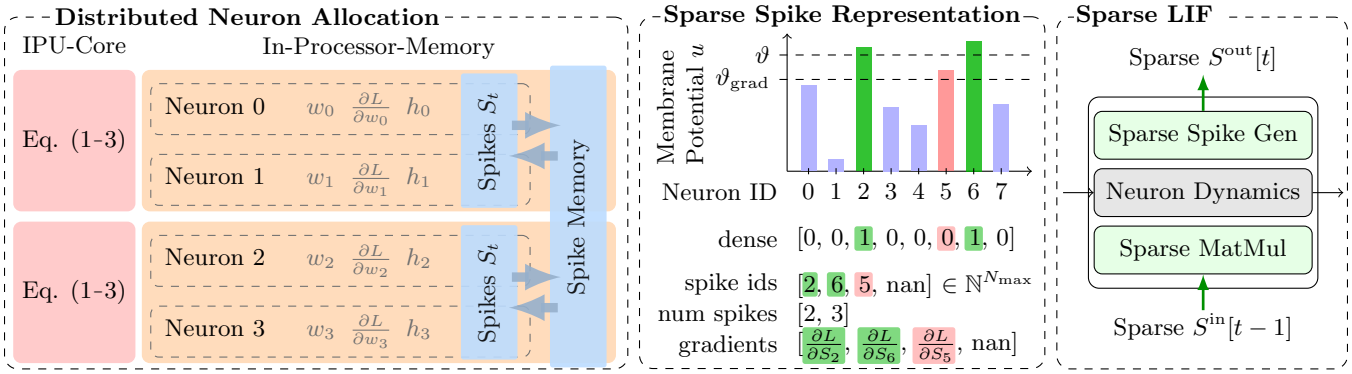

Figure 1: SNN implementation on the Graphcore IPU. Left: Illustration of the neuron placement on dedicated IPU-tiles including weights w, gradients $\frac{\partial L}{\partial w}$ and states h, which minimizes the necessary data transfer to solely spike based input and output data. Middle: Dense vs sparse spike vector representation as generated from membrane potentials and threshold vectors. Right: Computational graph of our sparse LIF cell with operations involving sparse spiking vectors marked in green.

Sparsifying Forward and Backward pass The right panel in Figure 1 shows the computational graph of the forward pass of a single layer of spiking neurons, featuring a function that computes the input current based on input spikes (Eq. 1), a function that integrates the neuron dynamics based on the input current (Eq. 2) and a function that generates the output spikes based on the membrane potential (Eq. 3). The integration of internal neuron dynamics is naturally modeled as a dense function where every neuron's state variables are updated every timestep. While the spiking activations and the input generating functions are typically also modeled as dense tensors and dense functions, in this work we utilize the sparsity of these variables to obtain efficiency gains with improved throughput and latency. For linear layers the forward pass is simplified from a matrix multiplication in the dense case, to a simple read-and-sum operation in the sparse case. Similarly, the corresponding functions in the backward pass can benefit from the generated sparse spiking tensor representations.

Sparse Spike representation In order to make use of the sparse activation of SNNs we use a sparse spike representation that stores the indices and the number of neurons that spiked. Our choice of sparse spike representation is driven by two factors:

- A naive implementation that contains only information about neurons that spiked can lead to inaccurate gradients in the backward pass. We circumvent this issue by taking a very similar approach as in (Perez-Nieves and Goodman 2021): In addition to propagating spikes from neurons whose membrane potential has crossed the spiking threshold ϑ, information is also transmitted from neurons above a secondary threshold ϑ^{grad}.
- The IPU is constrained to static tensor sizes that must be known during compile time. It is not possible to allocate or communicate varying tensor sizes depending on the number of spikes. Therefore we define the maximal number of considered spikes $N_{\max}$ as a hyperparameter and allocate spike tensors accordingly. This requires allocating a second tensor that contains the number of spikes.

Based on these two considerations we define the sparse spike representation for a given batchsize B as illustrated in the middle panel of Figure 1: It contains spike ids $\in \mathbb{N}^{B \times N_{\max}}$, the number of spikes and gradients $\in \mathbb{N}^{B \times 2}$ and the spike gradient tensor $\in \mathbb{R}^{B \times N_{\max}}$, which is only present in the backward pass. If the number of neurons above threshold exceeds $N_{\max}$ at a certain timestep, excess spikes are randomly dropped.

Integration into the SNN training workflow The current work features two sparse implementations for the IPU: Firstly, we developed a custom fully connected multi-layer LIF network in order to achieve maximal gains in regards to throughput and memory reduction. Secondly, we enable the integration of custom dynamic sparse operations into a standard implementation of neuron dynamics with Tensorflow's Python API. This gives flexibility to the programmer in the choice of neuron model and network architecture while still taking advantage of sparse activations. The first, fully custom approach is used to generate the benchmarking results.

Results

Benchmarking Setup

We benchmarked the sparse SNN implementation for the IPU by evaluating the throughput of multiple SNN models using the SHD (Cramer et al. 2020), N-MNIST (Orchard et al. 2015) and DVSGesture (Amir et al. 2017) dataset and compared to an equivalent dense implementation on a GPU (NVIDIA GeForce RTX 3090, NVIDIA V100, NVIDIA A100). We demonstrate this for fully-connected multi-layer feed-forward SNN architectures. If not specified otherwise, the network contains 2 neurons per tile, totaling $2 \times 1472 = 2944$ neurons. While the input and output sizes are dataset dependent, the remaining neurons are equally distributed into the specified number of hidden layers. The implementation is based on Keras (Chollet et al. 2015) with Tensorflow (Abadi et al. 2015) backend on the Python side and custom code via the Poplar SDK for the IPU in C++. The execution time is measured via a python timer within

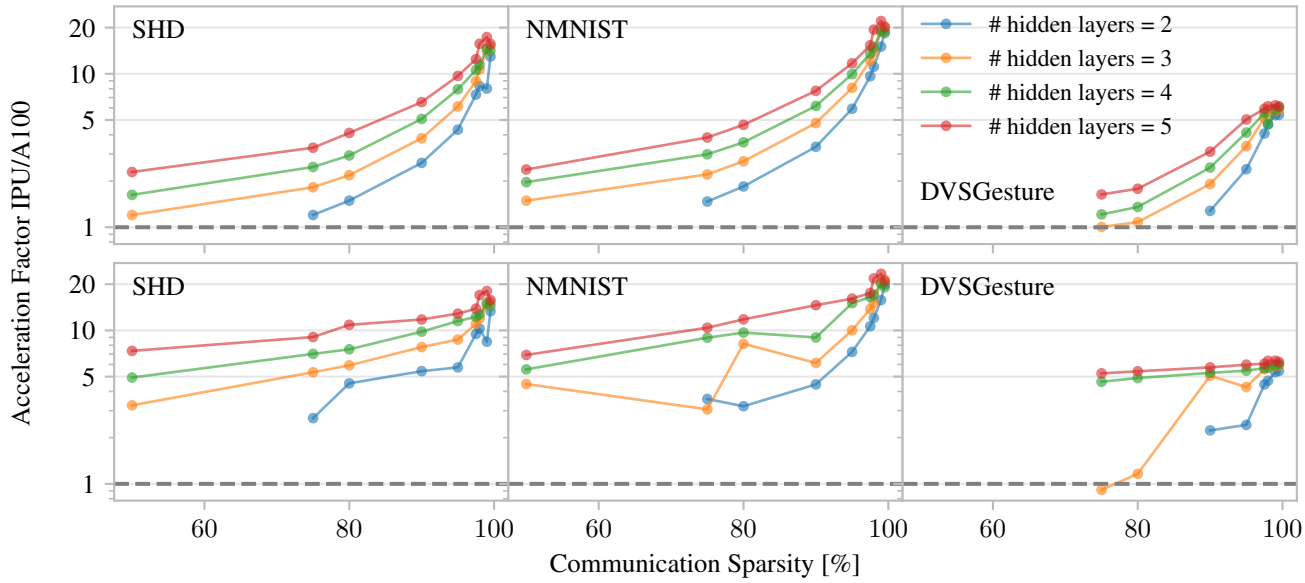

Figure 2: Acceleration factor of sparse IPU vs. dense A100 GPU implementations across varying levels of communication sparsity (1 - maximal activity) for a fixed network size of 2944 neurons with different hidden layer configurations on the SHD (left), NMNIST (middle) and DVSGesture (right) datasets. Top: "fixed activity" mode, all neurons spike every timestep, representing a lower bound approximation of throughput. Bottom: "natural activity" mode, where actual sparsity can exceed communication sparsity set by $N_{\max}$ resulting in maintained high acceleration factors for lower levels of communication sparsity.

a Keras callback. All implementations use 32-bit floating point precision, except for integer valued sparse spike tensors. The dense GPU implementation utilizes tensor cores using the `tf.fp32` datatype. To further improve throughput we choose a large value for `steps_per_execution` to process the whole data either within 1 or 2 cycles and we set `jit_compile=True` for the GPU implementation. We discard the first obtained time per epoch due to possible initial slowdows as a sideffect of compilation times and other overheads and average over the remaining times. We load the data to CPU RAM before execution to prevent a degradation in throughput due to bottlenecks in reading from disk. We fix the batch size for all benchmarking experiments to 48 to reduce the number of hyperparameters to sweep. While we observe even greater gains on the IPU for smaller batchsizes, we believe a batchsize of 48 to be a fair and insightful size for real world usecases. In order to obtain reliable results that are independent from random weight initialization and other factors, we execute the runs on the IPU with the sparse algorithm implementation with two different settings:

1. Natural activity: We don't manipulate the spiking activity, but randomly initialize the weight matrices and let the network run. For deeper layers, the activity tends to converge towards zero. This can be viewed as an conservative approximation of the **upper bound in throughput** and acceleration compared to the GPU runs.
2. Fixed activity: We force all neurons to spike. This leads to a saturation in the spike vectors, and thereby fixes the activity to the maximally allowed activity, which is chosen via $N_{\max}$ for the spike vector sizes. This mode can be viewed as approaching the **lower bound in throughput**.

The input spike tensors are fixed to a dataset dependent size, that was chosen based on the activity distribution of the data (NMNIST: 32, SHD: 48, DVSGesture: 96). The code is available at https://github.com/PGI15/SparseSNNsIPU.

For easy of use and reproducibility, the experiments for training convergence and accuracy are performed using a sparse tensor implementation in Jax with custom CUDA kernels for execution on the GPU. The custom GPU code replicates our custom implementation on the IPU with sparse tensors and sparse matrix multiplication. The Jax based code is available at https://github.com/PGI15/SparseSpikesJax.

Experimental Evaluation

Our results show a clear advantage in using sparse activations with our multi-layer SNN implementation on the IPU, demonstrating increasing gains with increasing sparsity of the activations. We do not observe significant differences in performance across different NVIDIA GPUs (RTX 3090, V100, A100) leading us to focus on results relative to the NVIDIA A100 in our discussions. The acceleration factor is determined as the ratio of GPU batch processing time to IPU batch processing time: $\text{acceleration factor} = \frac{\text{time per batch on the GPU}}{\text{time per batch on the IPU}}$.

Acceleration on Single IPU Figure 2 shows the acceleration factor on the IPU, where Figure 2 (top) demonstrates the behavior for the "fixed activity" and Figure 2 (bottom) for the "natural activity" case. The measurement of the acceleration factor includes all necessary operations for the training process, meaning the forward pass, the backward pass to calculate the gradients, and the weight update. By utilizing the

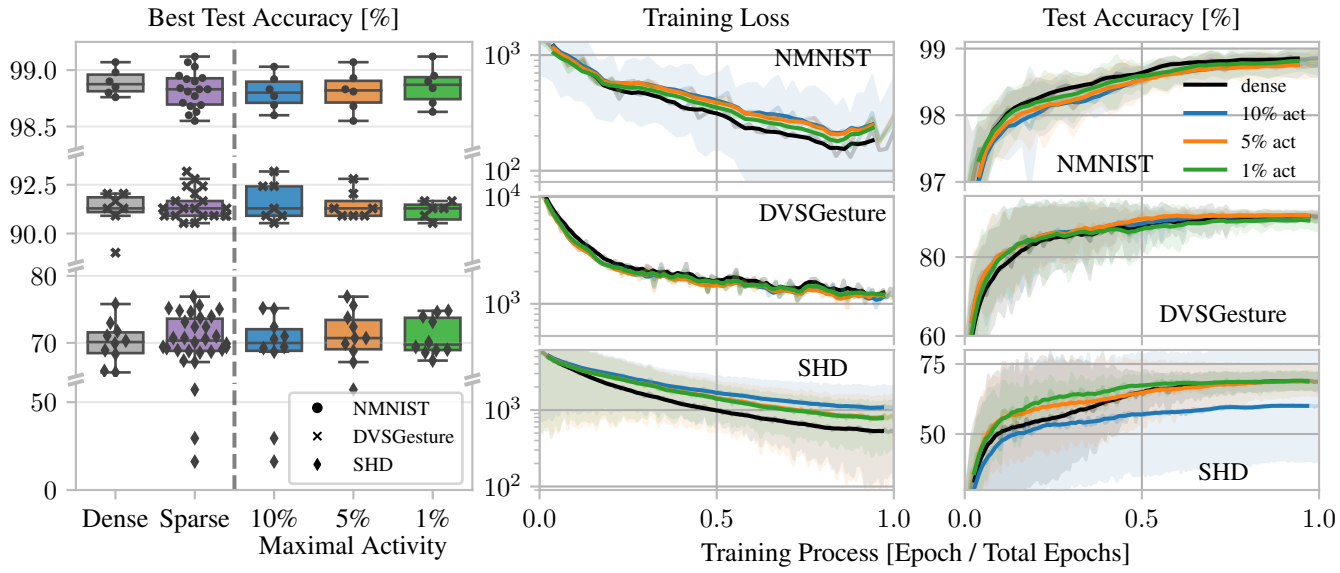

Figure 3: Achieved test accuracy (left) and convergence of training loss (middle) and test accuracy (right) of dense and sparse implementations for the same set of hyperparameters on NMNIST (top), DVSGesture (middle) and SHD (bottom) dataset. Slight differences in training loss convergence between dense and sparse implementations show no impact on test accuracy.

sparse activations we achieve substantial gains in throughput on the IPU by at least a factor of 5-10× compared to the GPU. For more extreme levels of activation sparsity which are still relevant for training runs in practice, even higher acceleration factors of 15-20× can be achieved. Averaging over all runs up to including maximal activity of 10 %, resulting in an average communication sparsity of 97.8 %, we obtain a throughput of 1.472 million event-frames or 14.72 thousand sequences (100 timesteps) per second for the NMNIST dataset using the sparse implementation on the IPU. The dense implantation on the A100 achieves 102 thousand frames or 1.02 thousand sequences per second (batchsize 48, architecture: [2048, 2934 neurons split over n layers, 10]). Somewhat reduced gains are observed for the DVSGesture dataset. Here, the network architecture showcases less homogeneity, particularly with a considerably larger input layer, revealing the necessity for enhancements in our mapping strategy to achieve superior load balancing on the IPU. Comparing "fixed activity" and "natural activity" scenarios, we observe that the former shows the anticipated proportional scaling with activity, while the latter maintains high acceleration factors for lower sparsity values. This suggests that execution speed is more influenced by network activity rather than specific hyperparameters limiting maximal activity. This fact underscores the robustness of our IPU implementation in the face of varying sparsity hyperparameters that might overestimate network activity, thus minimally compromising execution speed in practical applications.

Training Convergence and Accuracy Our analysis on final training accuracy and training convergence (Fig. 3) aligns with prior research (Perez-Nieves and Goodman 2021), indicating that sparse activation tensors don't impede training convergence or final model performance. Thus, the observed acceleration factors in throughput (Fig. 2) directly translate to faster overall training times. In order to fairly assess the differences in training behaviour, we initially determine a best performing set of hyperparameters (learning rate, number of time steps, decay constants, ...) for the dense implementation. Subsequently, other hyperparameters, namely the number of hidden layers and the use of augmentations, were varied for both the dense and the sparse implementations leading to the showcased training runs in our analysis. The left panel in Figure 3 shows the achieved test accuracy on the NMNIST, DVSGesture and SHD dataset across dense and sparse implementations, factoring in varying levels of sparsity. Except for a few outliers that did not converge in the sparse case, the test accuracies for dense and sparse implementations exhibit similar distributions. Notably, the best result on all three datasets was achieved with a sparse implementation. While some slowdowns occur in training loss convergence (Figure 3 middle), the corresponding test accuracies (Figure 3 right) show neither a reduced convergence speed nor a reduction in final accuracy. These results suggest an implicit regularization effect of the sparse activation tensors similar to Dropout (Hinton et al. 2012).

Scalability Analysis In order to analyse the scalability of our implementation, we both deploy larger networks on a single IPU by placing more neurons on every tile, and we perform weak scaling experiments, where the network size per IPU stays constant but is replicated to multiple IPUs.

Figure 4 (left) shows that for increasing network size on a single IPU, the acceleration compared to the GPU baseline further increases. To increase the network size, we modify the number of neurons that are allocated on each tile and extend the number of layers in the network architecture accord-

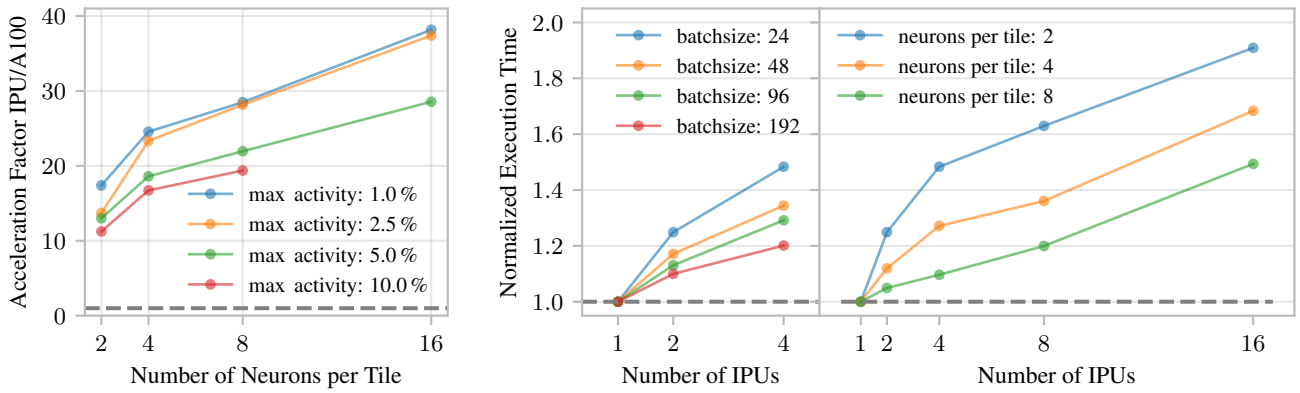

Figure 4: Left: Acceleration factor of Sparse IPU vs. A100 implementation for diverse network sizes achieved by adjusting number of neurons per IPU-tile. Right: Weak scaling results on SHD dataset with "natural activity" at max activity of 5%, exploring compute-workload to communication-latency ratios through batch size variation and network size per IPU adjustment.

ingly. Due to memory constraints we reduced the number of timesteps to 10 for the displayed experiments and used SGD instead of Adam. While these configurations may not align with standard training setups, they nevertheless provide informative insights and should translate to practical setups. Collectively, we observe that the acceleration is more pronounced in the case of the "natural activity" but also present for the "fixed activity" setup and it differs across datasets. The advantageous scaling on the IPU provides evidence that scaling up the network size on a single IPU in production settings further benefits acceleration compared to the GPU.

Figure 4 (right) shows the results of our weak scaling experiment for the SHD dataset. Here, the number of IPUs is increased while simultaneously increasing the network size (in terms of number of layers) accordingly. The number of neurons per tile and therefore the workload per tile and IPU is thereby kept constant. The ideal scaling behavior would lead to no slowdown in the total execution time for multi IPU settings as indicated by the grey dashed line. We analyse how the slowdown for multi-IPU settings behaves for settings ranging from 1 to 16 IPUs. Primarily, we observe that the slowdown can be reduced by increasing the computational workload compared to communication latency. We demonstrate this by increasing the batch size (Figure 4B left) and the number of neurons per tile meaning the network size per IPU (Figure 4B right). We clearly observe that for increased network density per IPU we achieve better scaling results, lowering the slowdown from 1.9× to 1.5× for the shown experiment when executing on 16 IPUs compared to a single IPU run. Additionally, we observe very little slowdown for large batchsizes, where we observe minimal slowdowns of 1.2× for 4 IPUs for large batchsizes of 192 samples per batch. Overall, this demonstrates improved scalability with network and data size.

In conclusion, exploiting sparse activations of SNNs and the IPU's MIMD nature with local, in-processor memory, significantly accelerates SNN training compared to modern GPUs. Positive scaling for increased network size on a single IPU and across multiple IPUs showcases the promise of accelerated large-scale SNN training on the Graphcore IPU.

Discussion

Our work underscores the suitability of manycore processors for recurrent models and highlights the advantages of their local memory and low-latency memory access patterns for sparse implementations. Concretely, we have demonstrated the potential for accelerating the training of spiking neural networks through the use of sparse spike vectors and a sparse vector matrix multiply algorithm on the Graphcore IPU. We have implemented the backpropagation through time (BPTT) training algorithm and achieved a substantial increase in throughput for realistic training scenarios - at least 5-10× higher compared to traditional architectures, and even higher in some special but realistic cases. Since the sparse training algorithm is not exact in the general case, one could suspect that the true acceleration factor in terms of model convergence may not be realized. However, we observed no slowdown in training convergence as a result of the introduced sparsity. This means that the acceleration in throughput directly translates to faster total training time. Looking ahead, we have noted promising scaling behavior for larger networks per IPU and in multi-IPU settings. Specifically, we have observed a significant increase in acceleration compared to GPUs when training larger networks per IPU, with up to 30-40× improvement for 16 neurons per tile. Our weak scaling results also indicate good scaling behavior, with only a 1.4× slowdown in throughput when scaling up to 16 IPUs, resulting in an effective 11× increase in compute.

We chose to demonstrate these ideas on the Graphcore IPU as it is a widespread and commercially available manycore processor with a very well developed software stack. We make sure to not limit our argument to just the IPU by excluding IPU specific features from our implementation like the accumulating Matrix Product (AMP) units and the additional streaming memory. Beyond LIF based SNNs, we try to make it easy to verify of our findings and adjust our implementation to other network architectures, topologies and neuron dynamics. For parts of the code base we enable flexible changes in the model architecture by exposing functional building blocks of our custom implementation

for neuron distribution and sparse workloads to Tensorflow's standard Python API.

As with GPUs, the amount of memory required to perform training using BPTT becomes a limiting factor for IPUs. This challenge is further amplified on IPUs due to the reduced total SRAM memory on the chip. Consequently, it becomes difficult to scale up to larger networks, where we observe even greater acceleration factors. Therefore, techniques like gradient checkpointing and exploring new training algorithms beyond BPTT such as three-factor rules and eligibility propagation (Neftci, Mostafa, and Zenke 2019; Bellec et al. 2020) is crucial to enable efficient implementation of spiking neural networks on hardware, particularly for problems and datasets that necessitate larger networks.

In combining such techniques with the observed acceleration factors on the IPU, we see a clear path towards large scale SNNs on the IPU and therefore the potential to revolutionize the deep learning model and hardware landscape.

Acknowledgments

This work was sponsored by the Federal Ministry of Education, Germany (project NEUROTEC-II grant no. 16ME0398K and 16ME0399) and Neurosys as part of the initiative "Cluster4Future" funded by the Federal Ministery of Education and Research BMBF (03ZU1106CB). The authors gratefully acknowledge computing time on the supercomputer JURECA (Jülich Supercomputing Centre 2021) at Forschungszentrum Jülich, providing access to NVIDIA A100 GPUs and Graphcore IPUs (as part of the JURECA DC Evaluation Platform) studied in this work.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Aimar, A.; Mostafa, H.; Calabrese, E.; Rios-Navarro, A.; Tapiador-Morales, R.; Lungu, I.-A.; Milde, M. B.; Corradi, F.; Linares-Barranco, A.; Liu, S.-C.; and Delbruck, T. 2019. NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. *IEEE Transactions on Neural Networks and Learning Systems*, 30(3): 644–656.

Amir, A.; Taba, B.; Berg, D.; Melano, T.; McKinstry, J.; Di Nolfo, C.; Nayak, T.; Andreopoulos, A.; Garreau, G.; Mendoza, M.; et al. 2017. A Low Power, Fully EventBased Gesture Recognition System. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 72437252.

Bauer, F. C.; Lenz, G.; Haghighatshoar, S.; and Sheik, S. 2023. EXODUS: Stable and efficient training of spiking neural networks. *Frontiers in Neuroscience*, 17.

Bellec, G.; Scherr, F.; Subramoney, A.; Hajek, E.; Salaj, D.; Legenstein, R.; and Maass, W. 2020. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1): 115.

Benjamin, B. V.; Gao, P.; McQuinn, E.; Choudhary, S.; Chandrasekaran, A. R.; Bussat, J.; AlvarezIcaza, R.; Arthur, J. V.; Merolla, P.; and Boahen, K. 2014. Neurogrid: A mixedanalogdigital multichip system for largescale neural simulations. *Proceedings of the IEEE*, 102(5): 699716.

Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; and Zhang, Q. 2018. JAX: composable transformations of Python+NumPy programs.

Chicca, E.; Stefanini, F.; and Indiveri, G. 2013. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of IEEE*.

Chollet, F.; et al. 2015. Keras. https://keras.io.

Cramer, B.; Stradmann, Y.; Schemmel, J.; and Zenke, F. 2020. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Davies, M.; Srinivasa, N.; Lin, T. H.; Chinya, G.; Joshi, P.; Lines, A.; Wild, A.; and Wang, H. 2018. Loihi: A Neuromorphic Manycore Processor with OnChip Learning. *IEEE Micro*, PP(99): 11.

Eshraghian, J. K.; Ward, M.; Neftci, E.; Wang, X.; Lenz, G.; Dwivedi, G.; Bennamoun, M.; Jeong, D. S.; and Lu, W. D. 2021. Training Spiking Neural Networks Using Lessons From Deep Learning. *CoRR*, abs/2109.12894.

Fang, W.; Chen, Y.; Ding, J.; Chen, D.; Yu, Z.; Zhou, H.; Masquelier, T.; Tian, Y.; and other contributors. 2020. SpikingJelly. https://github.com/fangwei123456/spikingjelly. Accessed: YYYY-MM-DD.

Frenkel, C.; and Indiveri, G. 2022. ReckOn: A 28nm Submm2 TaskAgnostic Spiking Recurrent Neural Network Processor Enabling OnChip Learning over SecondLong Timescales. In *2022 IEEE International SolidState Circuits Conference (ISSCC)*, volume 65, 13. IEEE.

Friedmann, S.; Schemmel, J.; Grübl, A.; Hartel, A.; Hock, M.; and Meier, K. 2017. Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE transactions on biomedical circuits and systems*, 11(1): 128142.

Friston, K. J. 2008. Hierarchical Models in the Brain. *PLoS Computational Biology*, 4.

Furber, S. B.; Galluppi, F.; Temple, S.; Plana, L.; et al. 2014. The spinnaker project. *Proceedings of the IEEE*, 102(5): 652665.

Gale, T.; Zaharia, M.; Young, C.; and Elsen, E. 2020. Sparse GPU Kernels for Deep Learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press. ISBN 9781728199986.

Gallego, G.; Delbruck, T.; Orchard, G.; Bartolozzi, C.; Taba, B.; Censi, A.; Leutenegger, S.; Davison, A.; Conradt, J.; Daniilidis, K.; et al. 2019. Eventbased vision: A survey. *arXiv preprint arXiv:1904.08405*.

Gewaltig, M.-O.; and Diesmann, M. 2007. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4): 1430.

Golosio, B.; Tiddia, G.; De Luca, C.; Pastorelli, E.; Simula, F.; and Paolucci, P. S. 2021. Fast Simulations of Highly-Connected Spiking Cortical Models Using GPUs. *Frontiers in Computational Neuroscience*, 15.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R. 2012. Improving neural networks by preventing coadaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hoefler, T.; Alistarh, D.; Ben-Nun, T.; Dryden, N.; and Peste, A. 2021. Sparsity in Deep Learning: Pruning and Growth for Efficient Inference and Training in Neural Networks. *J. Mach. Learn. Res.*, 22(1).

Hooker, S. 2021. The Hardware Lottery. *Commun. ACM*, 64(12): 58–65.

Horowitz, M. 2014. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International SolidState Circuits Conference Digest of Technical Papers (ISSCC)*, 1014. IEEE.

Huang, G.; Li, H.; Qin, M.; Sun, F.; Ding, Y.; and Xie, Y. 2022. Shfl-BW: Accelerating Deep Neural Network Inference with Tensor-Core Aware Weight Pruning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, 1153–1158. New York, NY, USA: Association for Computing Machinery. ISBN 9781450391429.

Jia, Z.; Tillman, B.; Maggioni, M.; and Scarpazza, D. P. 2019. Dissecting the Graphcore IPU Architecture via Microbenchmarking. *CoRR*, abs/1912.03413.

Jouppi, N. P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. 2017. Indatacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 112.

Jülich Supercomputing Centre. 2021. JURECA: Data Centric and Booster Modules implementing the Modular Supercomputing Architecture at Jülich Supercomputing Centre. *Journal of large-scale research facilities*, 7(A182).

Kauth, K.; Stadtmann, T.; Sobhani, V.; and Gemmeke, T. 2023. neuroAIxFramework: design of future neuroscience simulation systems exhibiting execution of the cortical microcircuit model 20× faster than biological realtime. *Frontiers in Computational Neuroscience*, 17.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knight, J.; and Nowotny, T. 2023. Larger GPU-accelerated brain simulations with procedural connectivity.

Knight, J. C.; and Nowotny, T. 2022. Efficient GPU Training of LSNNs Using EProp. In *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, NICE '22, 8–10. New York, NY, USA: Association for Computing Machinery. ISBN 9781450395595.

Kubilius, J.; Schrimpf, M.; Hong, H.; Majaj, N. J.; Rajalingham, R.; Issa, E. B.; Kar, K.; Bashivan, P.; PrescottRoy, J.; Schmidt, K.; et al. 2019. Brainlike object recognition with highperforming shallow recurrent anns. *arXiv preprint arXiv:1909.06161*.

Landsmeer, L. P. L.; Engelen, M. C. W.; Miedema, R.; and Strydis, C. 2023. Tricking AI chips into Simulating the Human Brain: A Detailed Performance Analysis. *CoRR*, abs/2301.13637.

Mead, C. 1990. Neuromorphic Electronic Systems. *Proceedings of the IEEE*, 78(10): 162936.

Merolla, P. A.; Arthur, J. V.; AlvarezIcaza, R.; Cassidy, A. S.; Sawada, J.; Akopyan, F.; Jackson, B. L.; Imam, N.; Guo, C.; Nakamura, Y.; et al. 2014. A million spikingneuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197): 668673.

Mishra, A. K.; Latorre, J. A.; Pool, J.; Stosic, D.; Stosic, D.; Venkatesh, G.; Yu, C.; and Micikevicius, P. 2021. Accelerating Sparse Deep Neural Networks. *ArXiv*, abs/2104.08378.

Neftci, E. O.; Mostafa, H.; and Zenke, F. 2019. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of GradientBased Optimization to Spiking Neural Networks. *IEEE Signal Processing Magazine*, 36(6): 5163.

Niedermeier, L.; Chen, K.; Xing, J.; Das, A.; Kopsick, J.; Scott, E.; Sutton, N.; Weber, K.; Dutt, N.; and Krichmar, J. L. 2022. CARLsim 6: An Open Source Library for Large-Scale, Biologically Detailed Spiking Neural Network Simulation. In *2022 International Joint Conference on Neural Networks (IJCNN)*, 1–10.

Orchard, G.; Lagorce, X.; Posch, C.; Furber, S.; Benosman, R.; and Galluppi, F. 2015. Realtime eventdriven spiking neural network object recognition on the SpiNNaker platform. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, 24132416. IEEE.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Perez-Nieves, N.; and Goodman, D. F. 2021. Sparse Spiking Gradient Descent. *Advances in Neural Information Processing Systems*.

Rosinol, A.; Rebecq, H.; Horstschaefer, T.; and Scaramuzza, D. 2018. Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High Speed Scenarios. *IEEE Robotics and Automation Letters*, PP: 1–1.

Shrestha, S. B.; and Orchard, G. 2018. SLAYER: Spike Layer Error Reassignment in Time. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 1419–1428. Curran Associates, Inc.

Subramoney, A.; Nazeer, K. K.; Schöne, M.; Mayr, C.; and Kappel, D. 2023. Efficient recurrent architectures through

activity sparsity and sparse back-propagation through time. In *The Eleventh International Conference on Learning Representations*.

Sun, P.-S. V.; Titterton, A.; Gopiani, A.; Santos, T.; Basu, A.; Lu, W. D.; and Eshraghian, J. K. 2022. Intelligence Processing Units Accelerate Neuromorphic Learning. *arXiv preprint arXiv:2211.10725*.

Wang, R. M.; Thakur, C. S.; and Van Schaik, A. 2018. An FPGAbased massively parallel neuromorphic cortex simulator. *Frontiers in neuroscience*, 12: 213.

Wang, Z. 2020. SparseRT: Accelerating Unstructured Sparsity on GPUs for Deep Learning Inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, 31–42. New York, NY, USA: Association for Computing Machinery. ISBN 9781450380751.

Yavuz, E.; Turner, J.; and Nowotny, T. 2023. GeNN: a code generation framework for accelerated brain simulations.

Zenke, F.; Bohté, S. M.; Clopath, C.; Comşa, I. M.; Göltz, J.; Maass, W.; Masquelier, T.; Naud, R.; Neftci, E. O.; Petrovici, M. A.; et al. 2021. Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron*, 109(4): 571575.

Zenke, F.; and Ganguli, S. 2018. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6): 15141541.

Zenke, F.; and Vogels, T. P. 2020. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *BioRxiv*.