electronics

Article

Topology-Aware Mapping of Spiking Neural Network to Neuromorphic Processor

Chao Xiao , Yao Wang, Jihua Chen and Lei Wang *

The College of Computer Science, National University of Defence Technology, Changsha 410073, China
* Correspondence: leiwang@nudt.edu.cn

Abstract: Neuromorphic processors, the new generation of brain-inspired non-von Neumann computing systems, are developed to better support the execution of spiking neural networks (SNNs). The neuromorphic processor typically consists of multiple cores and adopts the Network-on-Chip (NoC) as the communication framework. However, an unoptimized mapping of SNNs onto the neuromorphic processor results in lots of spike messages on NoC, which increases the energy consumption and spike latency on NoC. Addressing this problem, we present a fast toolchain, NeuToMa, to map SNNs onto the neuromorphic processor. NeuToMa exploits the global topology of SNNs and uses the group optimization strategy to partition SNNs into multiple clusters, significantly reducing the NoC traffic. Then, NeuToMa dispatches the clusters to neuromorphic cores, minimizing the average hop of spike messages and balancing the NoC workload. The experimental results show that compared with the state-of-the-art technique, NeuToMa reduces the spike latency and energy consumption by up to 55% and 86%, respectively.

Keywords: spiking neural network (SNN); neuromorphic processor; mapping; topology; toolchain

Citation: Xiao, C.; Wang, Y.; Chen, J.; Wang, L. Topology-Aware Mapping of Spiking Neural Network to Neuromorphic Processor. *Electronics* **2022**, *11*, 2867. https://doi.org/10.3390/electronics11182867

Academic Editor: Domenico Mazzeo

Received: 21 August 2022
Accepted: 8 September 2022
Published: 10 September 2022

1. Introduction

The neuromorphic processor [1–5] is the new generation of brain-inspired non-von Neumann computing systems, which uses in-memory or near-memory computing. The neuromorphic processor consists of multiple computing cores, which are called neuromorphic cores. Each core implements a limited number of spiking neurons. For example, there are 128 neuromorphic cores in Loihi [2] and a single core can at most accommodate 1024 spiking neurons. Network-on-Chip (NoC) [6] is generally adopted as the communication framework in those neuromorphic processors.

Neuromorphic processors typically implement machine learning tasks using spiking neural networks (SNNs) [7]. SNN, known as the third generation of the neural network, has been applied in various application fields including object recognition [8], electrocardiogram heartbeat classification [9], and image classification [10]. The basic units of an SNN are spiking neurons and synapses. A spiking neuron integrates spikes from presynaptic neurons, fires a spike when the membrane potential reaches the threshold, and sends spikes to the postsynaptic neurons. Consequently, the spiking neurons operate asynchronously, and there is a lot of spike communication between neurons, making the traditional CPUs or GPUs inefficient for the simulation of SNNs. Instead, the neuromorphic processors provide parallel computing and efficient communication, which better support the execution of SNNs.

Before executing an SNN-based application on neuromorphic processors, it should assign neurons into cores, load synaptic weights, and fill routing tables. The mapping process can be segmented into two steps. First, an SNN that cannot be accommodated in a single core is partitioned into several clusters so that the number of neurons per cluster does not exceed the capacity of a single core. Second, it selects appropriate cores for the execution of all clusters.

A complex SNN has lots of spiking neurons and synapses interconnecting neurons. The firing neuron needs to send spikes to all destination cores where the postsynaptic neurons reside, which leads to numerous spike messages on NoC. Although the neuromorphic processors provide parallel communication, the large amount of spike messages results in heavy pressure on NoC and degrades the execution performance.

Recently, some methods [11–14] have been proposed to tackle the mapping of SNNs, aiming at reducing the spike messages on NoC and improving the NoC performance. SpiNeMap [13] uses the Kernighan–Lin (KL) graph partitioning algorithm [15] to partition SNNs, aiming at reducing the number of spike messages on NoC. Then, SpiNeMap employs the particle swarm optimization (PSO) algorithm to search for the cluster-to-core mapping scheme. SNEAP [14] partitions SNNs using a multi-level (ML) graph partitioning algorithm [16] followed by a Simulated Annealing (SA) algorithm to map clusters to cores. Both methods transform an SNN into a graph and put neurons with high-frequency communication in the same cluster. However, due to the difference between the *neuron-to-neuron* communication and the *neuron-to-core* communication (see Section 2.2), both methods are easily trapped into local optimum.

In this paper, we focus on supervised machine learning tasks, where an SNN is first trained offline and then deployed on neuromorphic processors for inference. We propose NeuToMa (**To**pology-aware **Ma**pping of spiking neural network to **Neu**romorphic processor), which is a systematic and efficient toolchain for mapping SNN-based applications to neuromorphic processors. NeuToMa first partitions the SNNs into multiple clusters using the *group optimization* strategy. Then, NeuToMa selects appropriate cores for the execution of all clusters, minimizing the average hop of spike messages and balancing the workload of NoC. Our key contributions are as follows.

- We exploit the global topology of SNNs, divide the SNNs into multiple groups, and use the *group optimization* strategy to partition SNNs into multiple clusters. The *group optimization* strategy reduces the destination cores for a group of neurons simultaneously, which significantly reduces the spike messages on NoC.
- To map the partitioned clusters onto the multicore neuromorphic processor, we propose a heuristic mapping algorithm that minimizes the average hop of spike messages and balances the NoC workload.
- To obtain the NoC workload, we propose to use the spike firing rates of all spiking neurons to approximate the workload of physical links.

Table 1 compares NeuToMa against prior state-of-the-art approaches. We evaluate NeuToMa, SpiNeMap, and SNEAP using six SNN-based applications with different kinds of topology. The experimental results show that compared to SpiNeMap and SNEAP, NeuToMa on average reduces the NoC traffic by 47% and 13%, deceases the energy consumption by 86% and 51%, and has 55% and 31% lower spike latency, respectively.

Table 1. NeuToMa versus state-of-the-art methods.

	Partitioning	**Cluster-to-Core Mapping**	**Consistent Results after Multiple Tests?**
PSOPART [12]	particle swarm optimization	not optimized	×
SpiNeMap [13]	Kernighan–Lin graph partitioning algorithm	particle swarm optimization	×
SNEAP [14]	multi-level graph partitioning algorithm	simulated annealing algorithm	×
NeuToMa	topology-aware partitioning algorithm	a traversal algorithm with two optimization objectives	✓

2. Background and Related Work

2.1. Spiking Neural Network

Spiking neural networks, known as the third generation of neural network models, are event-driven computational models. The basic units of an SNN are spiking neurons and synapses interconnecting the neurons. Figure 1 illustrates an SNN with 11 spiking neurons organized into a fully connected network. A spiking neuron accepts spikes from its presynaptic neurons and updates its membrane potential. If the membrane potential exceeds the firing threshold, the neuron fires a spike and sends the spike to all postsynaptic neurons. After firing a spike, the membrane potential is reset to the resting potential. Neuroscientists have proposed a variety of spiking neuron models, such as the Leaky Integrate-and-Fire (LIF) model [17] that is generally implemented in many neuromorphic processors [2,4,5] due to its biological interpretability and low implementation complexity. The dynamics of the LIF neuron is defined as

$$\tau \frac{dV(t)}{dt} = -(V(t) - V_{rest}) + X(t) \tag{1}$$

where $V(t)$ and $X(t)$ are the membrane potential and input of the neuron at time t, respectively, and V_{rest} and τ are the resting potential and membrane time constant, respectively.

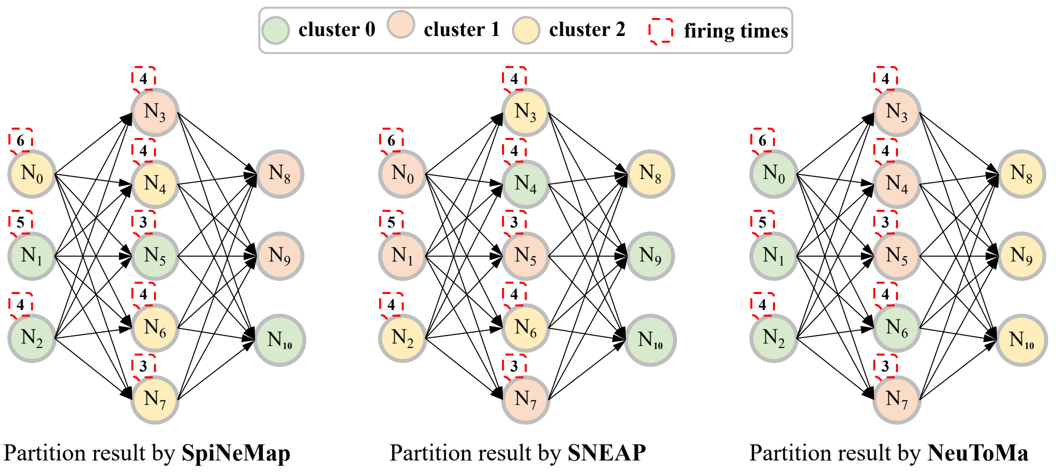
cluster 0
cluster 1
cluster 2
firing times
N0 N1 N2 N3 N4 N5 N6 N7 N8 N9 N10
6 5 4 4 4 3 4 3
Partition result by **SpiNeMap**
Partition result by **SNEAP**
Partition result by **NeuToMa**

Figure 1. An SNN is partitioned by SpiNeMap [13], SNEAP [14], and NeuToMa, respectively. The capacity of a single core is set to 4.

Prior works have proposed many SNN models [18–21]. According to the topology, those SNN models fall into three categories: feed-forward network [21], recurrent network [20], and hybrid network [18,19]. The neurons in the feed-forward network are organized in layers, and each neuron only connects to the neurons in the previous layer or next layer. There are no synapses between neurons within the same layer. The examples include the multi-layer perceptron network (MLP) and spiking convolutional neural network. In recurrent networks, there are lots of feedback synapses. The recurrent networks can better reflect the connections between real neurons, but they increase the difficulty in learning algorithm design, which makes them be mainly used in theoretical study. A hybrid network includes both feed-forward and recurrent synapses. Figure 2 shows the synfire chain (SFC) [19] consisting of three populations, an example of hybrid network. The synapses within each population are disordered, and the synapses across populations are feed-forward. Another example of a hybrid network is the liquid state machine (LSM) [18]. LSM contains three parts: the *input layer*, *liquid layer*, and *readout layer*. The synapses in the *liquid layer* are randomly initialized and then remain unchanged. The topology determines the direction of data flow. For example, the spike communication in the feed-forward network is unidirectional and from front to back.

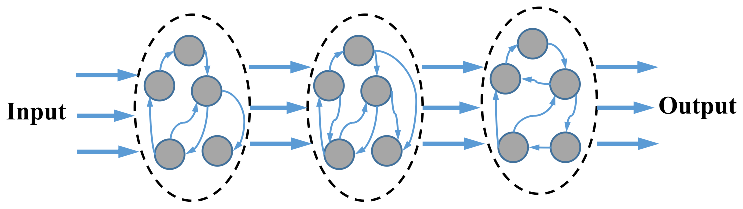

Figure 2. A synfire chain network with 3 populations.

2.2. Spike Communication in Neuromorphic Processor

The neuromorphic processor is a multicore system that integrates multiple neuromorphic cores. Each core implements a limited number of neurons and synapses. When a neuron fires a spike, it should transmit the spike to its postsynaptic neurons. Therefore, there will be spike communication between cores if a neuron and its postsynaptic neurons are assigned to different cores.

Compared to the traditional bus-based communication, NoC has an advantage in flexibility, scalability, and parallelism. Hence, NoC has been extensively used as a communication framework in neuromorphic processors. Each core is connected to the router via the network interface that converts the spike into a packetized form. If a neuron reaches the firing threshold and generates a spike, the processor will query the routing table and iterate over a list of destination cores for the firing neuron's fan-out distribution. Based on the routing table lookup, the processor generates packets with routing information, including neuron ID, source, and destination addresses.

Instead of sending a spike message to every synaptic neuron (*neuron-to-neuron* communication), neuromorphic processors generally adopt the *neuron-to-core* communication mechanism [1,2,4], which helps reduce the spike messages on NoC. The *neuron-to-core* communication is implemented hierarchically in two stages. First, once a neuron generates a spike, the source core where the firing neuron resides sends a spike message to every destination core. In other words, the fan-out connections of one neuron are projected to a list of core-to-core edges. Second, on reaching the destination core, the spike event is re-distributed within the core. An example is presented in Figure 3. The neuron N_0 and its postsynaptic neurons (N_1, N_2, N_3, N_4, and N_5) are distributed in four cores. N_1 and N_2 (N_3 and N_4) are mapped to the same core. Once N_0 fires a spike, $core_0$ will totally send three spike messages to $core_1$, $core_2$ and $core_3$. When the spike message reaches $core_1$ ($core_3$), the spike event will be reused by N_1 and N_2 (N_3 and N_4). Therefore, in the *neuron-to-core* communication mechanism, the source core on which the firing neuron is mapped just sends one spike message per destination core no matter how many postsynaptic neurons are assigned to a single destination core. The total number of spike messages on NoC can be calculated as

$$T_{spike} = \sum_{i=1}^{|N|} S(i) \times C_{pos}(i) \tag{2}$$

where $S(i)$ is the spike firing times of the ith neuron and $C_{pos}(i)$ is the number of cores where the postsynaptic neurons of the ith neuron are distributed, i.e., the destination cores. The core where the source neuron resides is not included in $C_{pos}(i)$.

At the end of every time step, the neuromorphic processor must ensure that all spike messages have been received by destination cores and force all neuromorphic cores to synchronize to the next time step. Thus, the high spike communication latency will impair the execution performance of neuromorphic processors.

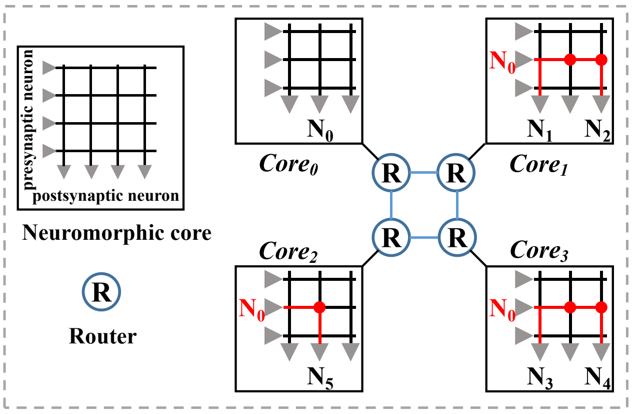

Figure 3. The *neuron-to-core* communication in neuromorphic processors.

2.3. Related Work

The mapping of applications to the multicore system is a hot research problem. The mapping of classical applications to traditional architectures is to allocate a set of tasks with certain constraints and dependencies to each computing core in a reasonable way to achieve a high computing resource efficiency and meet the timing constraints of all tasks. Neuromorphic processors are self-contained and near-memory computing systems, which means that the mapping of SNN-based applications onto neuromorphic processors must satisfy hardware resource constraints. The timing constraints present in classical applications have less influence in the SNN-based applications because SNN itself transmits information via the precise timing of spike trains consisting of a series of spikes.

PACMAN [11] is a proprietary tool for mapping SNNs onto SpiNNaker [3]. PSOPART [12] utilizes the PSO algorithm to partition an SNN into local and global synapses, aiming at reducing the spike messages on the time-multiplexed interconnect. However, the increasing searching space derived from the increasing size of SNNs and neuromorphic processors will consume lots of time. Both SpiNeMap and SNEAP transform an SNN into a graph by replacing the synaptic weights with the communication traffic between a pair of neurons. They use conventional graph partitioning algorithms to divide an SNN into multiple clusters. Figure 1 presents the partition result by SpiNeMap and SNEAP, respectively. The number of spike messages on NoC can be calculated as (see Equation (2))

$$T_{spike} = \begin{cases} S(N_{0,1,2,4,6,7}) \times 2 + S(N_{3,5}) \times 1 = 59 & SpiNeMap \\ S(N_{0,1,2,5,7}) \times 2 + S(N_{3,4,6}) \times 1 = 54 & SNEAP \end{cases} \tag{3}$$

The main idea of those partitioning algorithms is to put the neurons with high-frequency communication in the same cluster. The underlying assumption is that the basic communicating unit is each node (i.e., each neuron). Such a strategy would be effective if the neuromorphic processor sends one spike message to every postsynaptic neuron (i.e., *neuron-to-neuron* communication). However, as mentioned in Section 2.2, the fan-out connections of spiking neurons are projected to core-to-core edges. Therefore, the neuromorphic core can be regarded as the the basic communicating unit after mapping SNNs onto the neuromorphic processor. The mismatch between the *neuron-to-neuron* communication in those conventional partitioning algorithms and *neuron-to-core* communication in neuromorphic processors is the main reason for both methods being easily trapped into local optimum.

3. NeuToMa

3.1. Overview

The high-level overview of NeuToMa is shown in Figure 4. NeuToMa is composed of four steps, including SNN transformation, partitioning, mapping, and evaluation.

NeuToMa first transforms an SNN into a graph where the synaptic weights are replaced with the spike firing rates. Then, NeuToMa divides the SNN into multiple groups and uses the *group optimization* strategy to partition the SNN into multiple clusters, aiming at minimizing the spike communication traffic between clusters. Next, NeuToMa maps the partitioned clusters onto neuromorphic cores using a heuristic algorithm. Finally, a state-of-the-art neuromorphic processor is employed to evaluate our proposed approach.

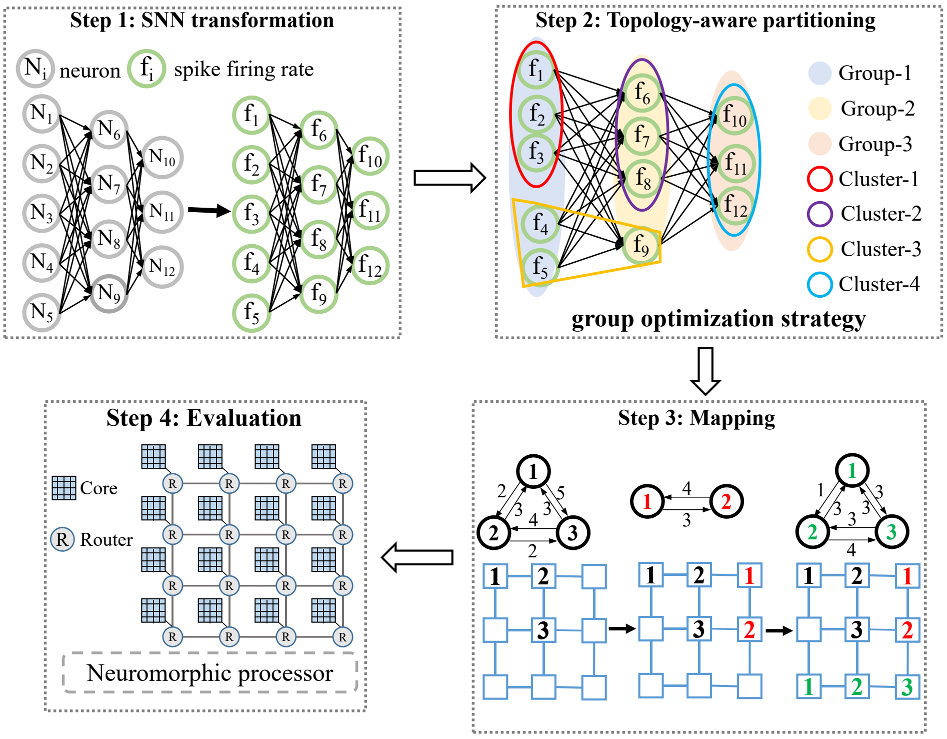

Figure 4. High-level overview of NeuToMa.

3.2. SNN Transformation

Before deploying an SNN-based application on neuromorophic processors for inference, the model is first trained offline. One way of acquiring a trained SNN is to train the SNN model in software simulators directly using training data. Another way is to convert the trained artificial neural networks (ANNs) to SNNs. In this paper, we use Brian2 [22], a software simulator, and SNN-TB tool [10], a conversion tool, for the direct training and conversion, respectively. Once a trained SNN is available, we simulate the model in Brian2 or SNN-TB using the training data, record the spike firing times of all neurons, calculate the spike firing rates, and replace the synaptic weights with the spike firing rates. A neuron's spike firing rate can be formulated as

$$f_i = S(i)/T \tag{4}$$

where T is the total simulation time. After the transformation, the SNN can be represented as a graph $\mathbf{G}(N,S)$ where N is the set of neurons and S is the set of synapses between neurons.

3.3. Topology-Aware Partitioning

The partitioned SNN can be represented as $\Phi(V,E)$ where V is the set of clusters and E is the set of edges between clusters. The SNN partitioning problem is transformed into $\mathbf{G}(N,S) \rightarrow \Phi(V,E)$. Let M_c be the maximum capacity of a single core. There are two constraints for the partitioning task. First, each neuron can be assigned to only one cluster. Second, the number of neurons per cluster cannot exceed M_c. The optimization objective in the partitioning stage is to minimize the communication traffic between clusters, which improves the energy consumption and spike latency on NoC.

As shown in Equation (2), the key to reducing the spike messages is decreasing the $C_{pos}(i)$. The strategy of prior works, putting the neurons with high-frequency communication in the same cluster, is essentially trying to reduce the $C_{pos}(i)$ of each individual neuron. This strategy is easily trapped into local optimum because the connections are locally intertwined. Instead of optimizing the $C_{pos}(i)$ for each individual neuron, we exploit the topology of SNNs, divide all neurons into multiple groups, and optimize the C_{pos} for each group's neurons simultaneously. This *group optimization* strategy can be formulated as

$$Partition_{scheme} = min\{C_{pos}(\,\mathbf{group}(\,i\,)\,)\ i \in 1, 2, ..., r\} \tag{5}$$

where $\mathbf{group}(\,\mathrm{i}\,)$ is the ith group and r is the number of groups after the *division*.

The topology-aware partitioning process of NeuToMa is shown in Figure 5. It includes three substeps: *division*, *partition*, and *merging*.

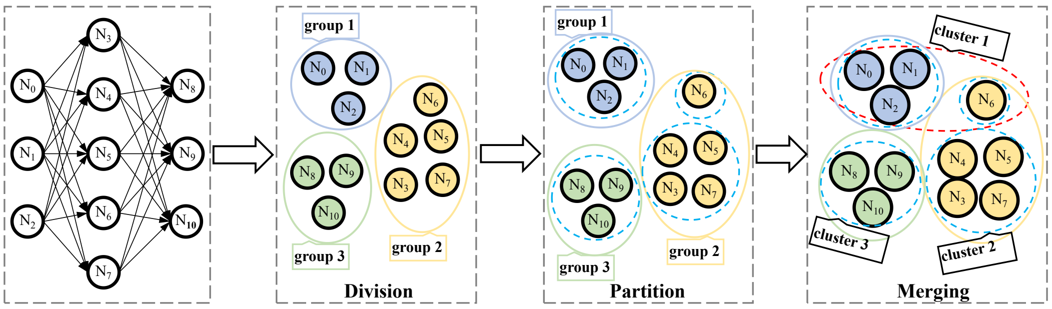

Figure 5. The topology-aware partitioning process.

3.3.1. Division

Although the synapses in an SNN are locally intertwined, the topology of the entire SNN is globally regular. In the SNNs with feed-forward topology, the neurons are regularly arranged in layers. The postsynaptic neurons of the ith layer's neurons are mainly distributed over the $(i + 1)$th layer. The examples are spiking convolutional neural network and MLP. In the SFC network, although the synapses within each population are disordered, the synapses across populations are feed-forward. Consequently, the postsynaptic neurons of the same population's neurons are distributed in the current population and the next population.

Based on the above observation, NeuToMa divides an SNN into multiple groups so that the postsynaptic neurons of one group's neurons are chiefly distributed in a few groups. For example, one population of an SFC network or a layer of an MLP network can be treated as a *group*. In terms of the LSM network, the global topology is irregular because the connections are randomly initialized. Therefore, the whole LSM network is regarded as one *group*. The *division* operation exploits the global topology of an SNN and gathers neurons that have the same postsynaptic and presynaptic neurons into one *group*.

3.3.2. Partitioning and Merging

To distribute the postsynaptic neurons of one *group* in as few partitioned clusters as possible, instead of partitioning an entire SNN directly, NeuToMa partitions each *group* into multiple clusters while satisfying the hardware resource constraints. The partitioning strategy for intra-group neurons is to put the neurons with high-frequency communication in the same cluster.

After partitioning all groups into clusters, there may exist some small clusters that just contain a small number of spiking neurons. For example, as shown in Figure 5, after partitioning the second *group*, one of the partitioned clusters just contains a spiking neuron (i.e., N_3). Instead of mapping a small cluster to a single core, NeuToMa merges some small clusters into a bigger cluster while meeting the hardware resource constraints.

The merged cluster is mapped to a single core. Figure 1 shows the partition result by NeuToMa. The total number of spike messages on NoC after the partition is

$$T_{spike} = \left\{ \; S(N_{0,1,2,3,4,5,6,7}) \times 1 = 33 \qquad NeuToMa \right. \tag{6}$$

Compared with SpiNeMap and SNEAP, NeuToMa significantly reduces the spike messages. The critical step is to divide an SNN into multiple groups according to the global topology, which collects the postsynaptic neurons of each group in a few groups. Moreover, the partition is only applied to each group, not the entire SNN, which avoids the re-dispersion of postsynaptic neurons.

3.4. Mapping

When there are enough hardware resources for multiple applications, it is expected to execute those applications simultaneously to increase the throughput of the neuromorphic processor. Figure 6 illustrates four candidate mapping schemes of two partitioned SNNs to the processor with 3×3 cores arranged in a mesh topology. The routing algorithm in Figure 6 is the X-Y routing algorithm, which is a deterministic dimensional routing algorithm. The numbers on the physical links are the sum of spike messages passing through the physical link. Obviously, different mapping schemes lead to different utilizations of physical links and hops of spike messages, which impacts both spike latency and energy consumption.

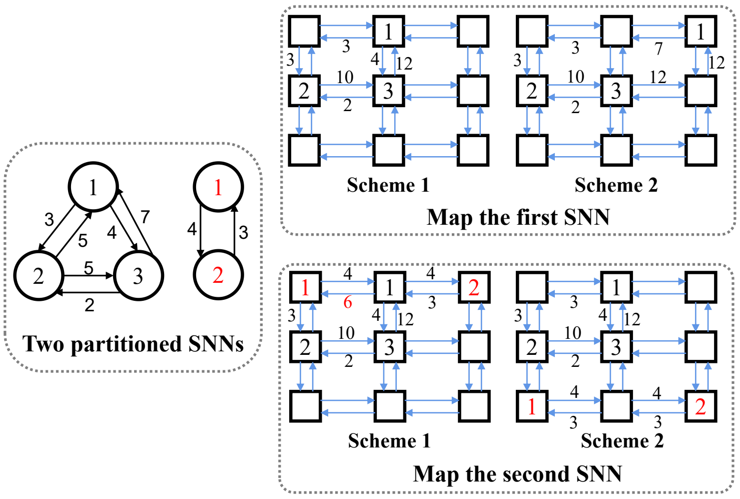

Figure 6. Different mapping schemes for two SNNs.

Let $\Psi(C, I)$ be the neuromorphic processor with a set C of neuromorphic cores and a set I of physical links. Mapping can be transformed into M: $\Phi(V, E) \rightarrow \Psi(C, I)$. The result of mapping can be expressed as a $\mid V \mid \times \mid C \mid$ matrix m$_{ij}$, where each element is 0 or 1. The value of m$_{ij}$ is defined as

$$m_{ij} = \left\{ \begin{array}{ll} 1 & if\ cluster\ V_i\ is\ mapped\ to\ core\ C_j \\ 0 & otherwise. \end{array} \right. \tag{7}$$

There are two constraints for the mapping task. First, a cluster can be mapped to only one neuromorphic core. Second, a core can accommodate at most one cluster, which makes the cores at work unavailable for the new application.

The workload imbalance of NoC may cause local congestion, which increases the communication delay and degrades the performance. Hence, the workload balance of NoC is one of the optimization objectives in NeuToMa.

The key is to obtain the workload of all physical links. Although the real-time monitoring provides the latest traffic on NoC, it requires additional monitoring units and complicates the NoC design. Instead, we analyze the communication patterns within the SNNs and obtain the approximate workload of physical links based on the knowledge.

A property of SNN models is that the synapses dynamically change over time. If a spiking neuron fires a spike and the membrane potential returns to the resting potential at time t, it is hard for the neuron to fire again at time t + 1, as recharging needs time. Therefore, no spike passes through the synapses between the neuron and its postsynaptic neurons at time t + 1. This fact makes the communication traffic of the entire SNN at each time step unpredictable.

Although it is hard to acquire the communication traffic for every time step, we can obtain the average communication traffic for a period of time, i.e., the spike firing rates. The average firing rates of all neurons over a period of time can be obtained by statistical analysis in the training stage (see Section 3.2). Instead of relieving the NoC congestion for every time step, NeuToMa tries to improve the congestion overall. We first calculate the communication frequency between a pair of cores using Equation (8)

$$frequency(C_i, C_j) = F(i) \times H(i,j) + F(j) \times H(j,i) \tag{8}$$

Here, $F(i)$ is a vector of the firing rates of the ith core's neurons. The $H(i,j)$ is a vector containing 0 or 1. If a neuron of the ith core has postsynaptic neurons mapped to the jth core, the neuron's corresponding position in $H(i,j)$ is 1; otherwise, it is 0. Then, we use Equation (9) to approximate the link workloads. L_m is the workload of the mth link. The $O(i,j,m)$ is 1 if the route from the ith to the jth core passes the mth link; otherwise, it is 0.

$$L_m = \sum_{j=1}^{|C|} \sum_{i=1}^{|C|} frequency(C_i, C_j) \times O(i,j,m) \tag{9}$$

The energy consumption of transmitting one spike message from core$_i$ to core$_j$ can be calculated by:

$$E(i,j) = n_{hop} \times E_S + (n_{hop} - 1) \times E_L \tag{10}$$

where E_S and E_L represent the energy consumption on the routers and links, respectively, and n$_{hop}$ is the number of routers the spike message traverses from core$_i$ to core$_j$. Reducing the hop of spike messages improves both spike latency and energy consumption. Therefore, the hop of spike messages is the second optimization objective.

We formalize the mapping process as shown in Algorithm 1. The major part of the mapping algorithm is a doubly nested loop (lines 2–12). For every cluster, the algorithm explores all available cores to minimize the average hop and balance the workload of NoC (lines 4–10). The algorithm first pre-maps the current cluster on the selected core and calculates the average hop and $Balance_{load}$, the link workload variance (lines 5–6). Then, by comparing with the historical optimal record, the algorithm determines the better choice (lines 7–9). In the process of determining a better core, the average hop is the first optimization objective. Optimizing the average hop preferentially can achieve two goals. (i) It reduces the route length and consequently improves energy consumption and spike latency. (ii) It maps the clusters from the same application onto the contiguous cores, which diminishes mutual interference among multiple running applications.

Algorithm 1: Mapping algorithm

```
   Input: the partitioned SNN $\Phi(V, E)$, the neuromorphic processor $\Psi(C, I)$
   Output: the mapping matrix $M$
1  $M \leftarrow \{0\}^{|V|\times|C|}$;
2  for $curCluster \in V$ do
3      $Hop_{best}, Balance_{best}, selectCore \leftarrow +\infty, +\infty, -1$;
4      for $curCore \in availableCores$ do
5          Pre-map $curCluster$ on $curCore$;
6          Calculate $Hop_{cur}$ and $Balance_{cur}$;
7          if $Hop_{cur} < Hop_{best}$ or $(Hop_{cur} == Hop_{best}$ and $Balance_{cur} < Balance_{best})$
            then
8              $Hop_{best}, Balance_{best}, selectCore \leftarrow Hop_{cur}, Balance_{cur}, curCore$;
9          end
10     end
11     $M_{curCluster,\, selectCore} \leftarrow 1$ ;
12 end
```

4. Experiment Setup

4.1. Experiment Platform

Unicorn [4], a multicore neuromorphic processor, is adopted as the target neuromorphic processor to evaluate our mapping approach. Unicorn integrates two spiking neuron models, LIF and integrate-fire (IF) models. Unicorn supports the unconstrained fan-out and flexible fan-in of neurons. A 3×3 C-Mesh NoC with an X–Y routing algorithm is adopted by Unicorn for inter-core spike communication.

We extend the neuromorphic processor size to 16×16. Each neuromorphic core can accommodate at most 256 neurons. Noxim++ [13], a cycle-accurate NoC simulator, is used to obtain the key performance statistics of NoC, such as spike latency and energy consumption.

4.2. Evaluated SNNs

Table 2 lists a group of SNN-based applications used to evaluate the proposed mapping approach. MLP-MNIST and MLP-FaMNIST (MLP-Fashion-MNIST) are two MLP-based networks for the MNIST dataset [23] and Fashion-MNIST dataset [24], respectively. We use two CNN models to perform image classification on the Fashion-MNIST dataset and CIFAR10 dataset [25]. Both CNN-FaMNIST and CNN-CIFAR10 are first trained with the back-propagation algorithm using the training dataset. Then, the trained CNNs are converted into SNNs using the SNN-TB tool [10]. We create two hybrid networks, LSM-FSDD and SFC-FSDD for the FSDD dataset (accessed on https://github.com/Jakobovski/free-spoken-digit-dataset), which is a simple audio and speech dataset consisting of recordings of spoken digits at 8 kHz. The “E_800” and “I_200” represent 800 excitatory neurons and 200 inhibitory neurons. SFC-FSDD contains 1000 spiking neurons organized into three populations.

The images from MNIST, Fashion-MNIST, and CIFAR10 are converted into Poisson-distributed spike trains, with firing rates proportional to the intensity of the pixels. MLP-MNIST, MLP-FaMNIST, LSM-FSDD, and SFC-FSDD are simulated in Brian2 [22], an SNN software simulator, using the LIF model. After the conversion, CNN-FaMNIST and CNN-CIFAR10 are simulated in INIsim, a built-in simulator of SNN-TB, supporting the IF model. During the simulation, we record all spike firing information.

Column 3 reports the number of neurons present in the SNN-based applications. Column 4 reports the number of synapses. Three combinations of the six applications as follows are mapped onto the neuromorphic processor simultaneously to evaluate the three approaches.

- **Group 1**: MLP-MNIST, MLP-FaMNIST;
- **Group 2**: LSM-FSDD, SFC-FSDD;

- **Group 3**: MLP-MNIST, MLP-FaMNIST, CNN-FaMNIST.

Table 2. SNN-based applications used for evaluation.

SNNs	Topology	Neurons	Synapses
MLP-MNIST	Feed-foward	738	548,480
MLP-FaMNIST	Feed-foward	738	548,480
CNN-FaMNIST	CNN	16,714	1,948,288
CNN-CIFAR10	CNN	19,894	2,343,464
LSM-FSDD	LSM(E_800,I_200)	1000	399,970
SFC-FSDD	SFC(400,300,300)	1000	27,097

5. Results

Due to the randomness of SpiNeMap and SNEAP, we have conducted the experiments several times. We present all experimental results in this section.

5.1. Partitioning and Mapping Performance

5.1.1. Partitioning Performance

We illustrate the total number of spike messages on NoC in Figure 7, which can be calculated using Equation (2). We make the following two observations.

First, SNEAP has an average 34% lower spike count compared to SpiNeMap, which is due to the difference between the ML algorithm and KL algorithm. The KL algorithm first distributes the neurons arbitrarily to multiple clusters. Next, it iteratively swaps neurons between clusters to reduce the total communication. The ML algorithm merges two neurons with the maximum communication traffic into a new node and repeats this process iteratively. Then, it fine-tunes the neurons to satisfy the hardware resource constraint. Compared with the KL algorithm, the ML algorithm is better for jumping out of local optimum.

Second, compared with SpiNeMap and SNEAP, NeuToMa generates the lowest number of spike messages (on average, 47% lower than SpiNeMap and 13% lower that SNEAP). This reduction is because NeuToMa exploits the topology of SNNs and uses the *group optimization* strategy to reduce the C_{pos} for a set of neurons, which avoids trapping in the local optimum. For the CNN-FaMNIST and CNN-CIFAR10 applications, NeuToMa even reduces the spike message count by 87% and 91%, respectively, compared to SpiNeMap. This is because as the SNN size increases, the increased number of partitioned clusters increases the distribution of postsynaptic neurons, which makes SpiNeMap more likely to be trapped into local optimum. In terms of LSM-FSDD application, SNEAP, SpiNeMap, and NeuToMa perform nearly the same. It is because there are lots of recurrent synapses between neurons in the network. After the partition, the postsynaptic neurons of each neuron are distributed to all clusters.

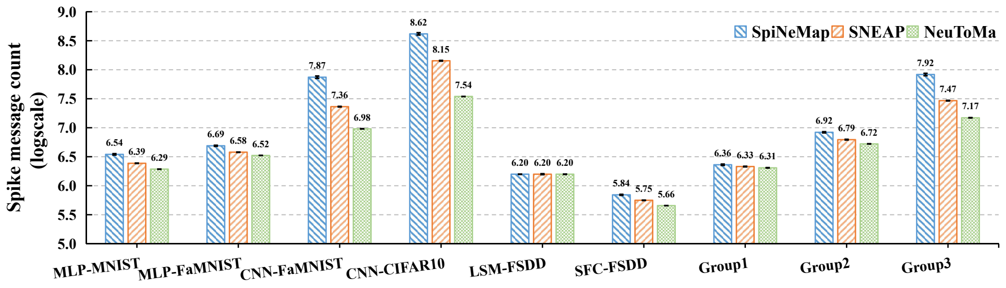

Figure 7. The total number of spike messages on NoC.

5.1.2. Mapping Performance

We illustrate the average hop of all spike messages in Figure 8. Compared with SpiNeMap and SNEAP, NeuToMa on average has a 69% and 68% reduction in the average hop. Both SpiNeMap and SNEAP adopt the average hop as the only optimization objective in the mapping stage. SpiNeMap utilizes an instance of PSO to search for the optimum mapping and SNEAP employs the SA algorithm to find the best mapping scheme. Although both algorithms can optimize the average hop to some extent, they are easily trapped into local optimum due to the random searching strategy. It should be noted that after several tests, the average hop obtained by the PSO algorithm nearly equals that of the SA algorithm. NeuToMa traverses all unallocated cores for the current cluster and chooses the best alternative. NeuToMa may not guarantee that the mapping result is globally optimal, but the selected core is the best choice from all unallocated cores at present.

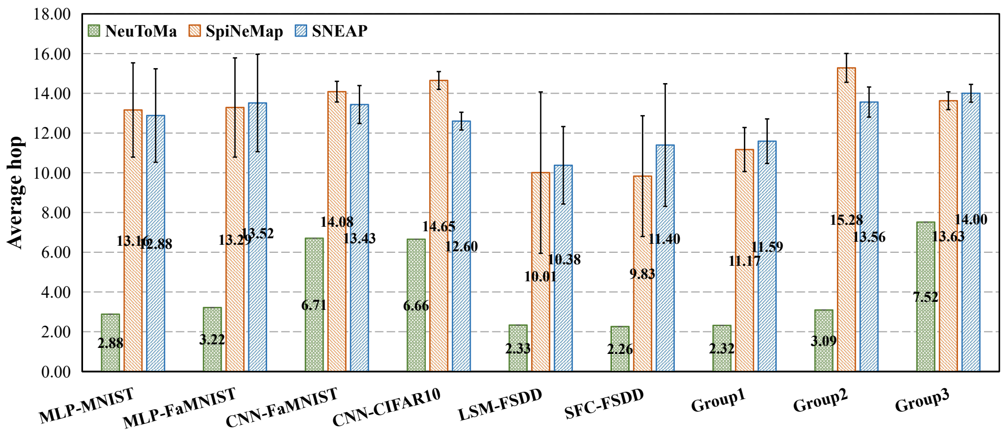

Figure 8. The average hop of spike messages.

When the number of neuromorphic cores greatly exceeds the number of partitioned clusters, such as the MLP-MNIST and MLP-FaMNIST applications, the mapping algorithm of NeuToMa shows more advantages. Figure 9 illustrates the average hop when mapping the partitioned MLP-MNIST onto the neuromorphic processors with different sizes. As the size of the target neuromorphic processor increases, the searching space of SpiNeMap and SNEAP in the mapping process is rapidly expanding, which makes them easier to fall into local optimum and increases the final average hop. Furthermore, a larger searching space leads to a higher fluctuation in the average hop, which increases the uncertainty of the mapping result. Instead, the average hop of NeuToMa remains constant. This is because the searching process in NeuToMa is sequential and deterministic. Consequently, when the neuromorphic cores greatly outnumber the partitioned clusters, the distant cores will be deserted when choosing the best core from candidate cores.

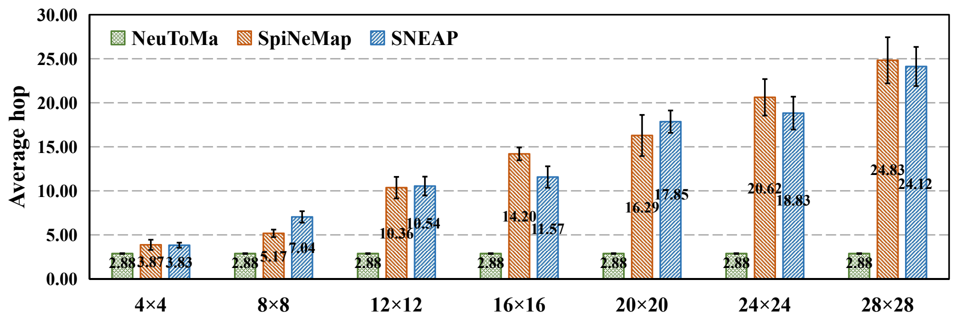

Figure 9. The average hop of spike messages obtained under different neuromorphic processor sizes.

5.2. NoC Performance

5.2.1. Spike Latency

This is the average spike latency experienced by spike messages on NoC. Figure 10 reports the spike latency of the nine tasks for the evaluated approaches normalized to SpiNeMap. Compared with SpiNeMap and SNEAP, NeuToMa has the lowest spike latency (on average, 55% lower than SpiNeMap and 31% lower than SNEAP). For the CNN-

FaMNIST, CNN-CIFAR10, and Group 3 tasks, NeuToMa reduces the spike latency by 89%, 87%, and 87%, respectively, compared to SpiNeMap.

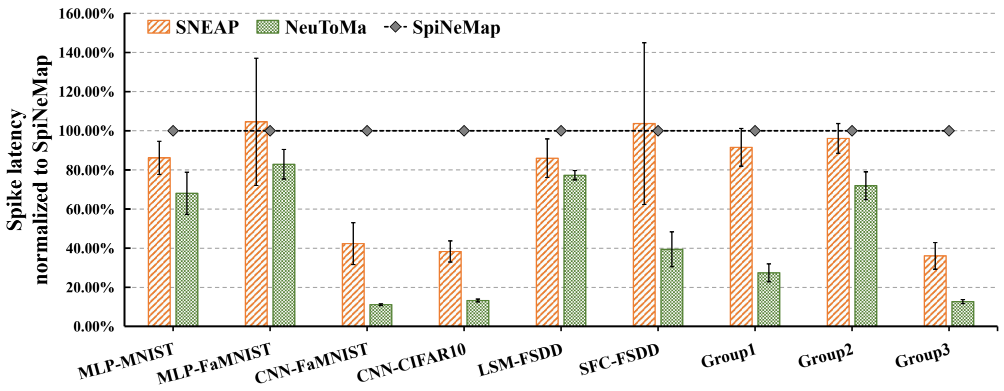

Figure 10. Spike latency normalized to SpiNeMap.

Three reasons are responsible for the improvement in spike latency. First, NeuToMa reduces the most spike messages on NoC among the three mapping approaches, which globally relieves the NoC congestion. Second, in the mapping stage, the average hop is employed as one of the two optimization objectives in NeuToMa. As shown in Section 5.1.2, NeuToMa significantly reduces the average hop for all evaluated applications. A shorter routing path contributes to less time consumed in traversing the spike messages from the source core to the destination core. Third, the workload balance is selected as another optimization target in the mapping stage, which locally relieves the communication pressure on some critical physical links.

5.2.2. Energy Consumption on NoC

This is the total energy consumption consumed by all spike messages on NoC. Figure 11 reports the energy consumption of the evaluated applications for the three mapping approaches normalized to SpiNeMap. Compared with SpiNeMap and SNEAP, NeuToMa has the lowest energy consumption (on average, 86% lower than SpiNeMap and 51% lower than SNEAP). Especially for both CNN-FaMNIST and CNN-CIFAR10 applications, NeuToMa reduces the energy consumption by 94% and 96%, respectively, compared to SpiNeMap.

There are two reasons accounting for this improvement. First, as detailed in Section 5.1.1, NeuToMa reduces the most spike messages on NoC among the evaluated methods, which is the main reason for reducing the energy consumption. Second, NeuToMa employs the average hop as the first optimization objective, which shortens the routing path of spike messages. A shorter routing path contributes to less energy consumption for transmitting a spike message from the source core to the destination core.

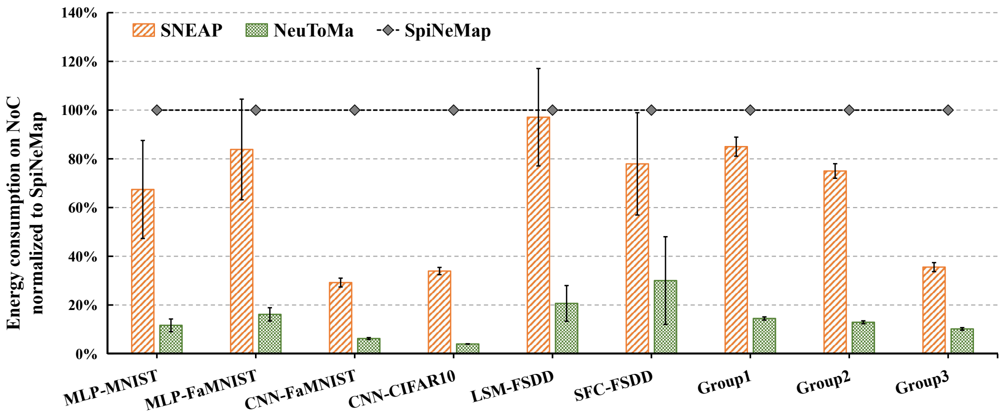

Figure 11. Energy consumption normalized to SpiNeMap.

6. Conclusions

In this paper, we introduce NeuToMa, a toolchain to map SNN-based applications to the multicore neuromorphic processor. NeuToMa exploits the global topology of SNNs and divides an SNN into multiple groups so that the postsynaptic neurons of a group's neurons are distributed in a few groups. Then, NeuToMa partitions each group into multiple clusters and merges some small clusters while satisfying the hardware resource constraint. In the mapping stage, both the average hop of spike messages and workload balance of NoC are employed as the optimization objectives. Instead of using the meta-heuristics algorithms that are easily trapped into local optimum when the neuromorphic cores greatly outnumber the partitioned clusters, we propose a traversal algorithm to search for the best cluster-to-core mapping scheme. Compared to SpiNeMap and SNEAP, NeuToMa on average reduces the spike messages by 47% and 13%, deceases the energy consumption by 86% and 51%, and has 55% and 31% lower spike latency, respectively.

Author Contributions: data curation, C.X., Y.W. and L.W.; investigation, C.X., Y.W., J.C. and L.W.; methodology, C.X., Y.W., J.C. and L.W.; software, C.X.; visualization, C.X. and J.C.; validation, C.X., Y.W., J.C. and L.W.; resources, C.X, Y.W. and L.W.; writing—original draft preparation, C.X., Y.W. and L.W.; writing—review and editing, C.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Key R&D Program of China grant number 2018YFB2202603.

Data Availability Statement: The data presented in this study are available on request from corresponding authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Merolla, P.A.; Arthur, J.V.; Alvarez-Icaza, R.; Cassidy, A.S.; Sawada, J.; Akopyan, F.; Jackson, B.L.; Imam, N.; Chen, G.; Yutaka, N.; et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **2014**, *345*, 668–673. [CrossRef] [PubMed]
2. Davies, M.; Srinivasa, N.; Lin, T.H.; Chinya, G.; Joshi, P.; Lines, A.; Wild, A.; Wang, H. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
3. Furber, S.B.; Lester, D.R.; Plana, L.A.; Garside, J.D.; Painkras, E.; Temple, S.; Brown, A.D. Overview of the SpiNNaker System Architecture. *IEEE Trans. Comput.* **2013**, *62*, 2454–2467. [CrossRef]
4. Yang, Z.; Wang, L.; Wang, Y.; Peng, L.; Chen, X.; Xiao, X.; Wang, Y.; Xu, W. Unicorn: A Multicore Neuromorphic Processor with Flexible Fan-In and Unconstrained Fan-Out for Neurons. In Proceedings of the 59th ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 10–14 July 2022. [CrossRef]
5. Wang, L.; Yang, Z.; Guo, S.; Qu, L.; Zhang, X.; Kang, Z.; Xu, W. LSMCore: A 69k-Synapse/mm2 Single-Core Digital Neuromorphic Processor for Liquid State Machine. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, *69*, 1976–1989. [CrossRef]
6. Benini, L.; Micheli, G.D. Networks on chip: A new paradigm for systems on chip design. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Antwerp, Belgium, 14–23 March 2002.
7. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
8. Xiang, S.; Jiang, S.; Liu, X.; Zhang, T.; Yu, L. Spiking VGG7: Deep Convolutional Spiking Neural Network with Direct Training for Object Recognition. *Electronics* **2022**, *11*, 2097. [CrossRef]
9. Xing, Y.; Zhang, L.; Hou, Z.; Li, X.; Shi, Y.; Yuan, Y.; Zhang, F.; Liang, S.; Li, Z.; Yan, L. Accurate ECG Classification Based on Spiking Neural Network and Attentional Mechanism for Real-Time Implementation on Personal Portable Devices. *Electronics* **2022**, *11*, 1889. [CrossRef]
10. Rueckauer, B.; Liu, S.C. Conversion of analog to spiking neural networks using sparse temporal coding. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.
11. Galluppi, F.; Davies, S.; Rast, A.; Sharp, T.; Plana, L.A.; Furber, S. A hierachical configuration system for a massively parallel neural hardware platform. In Proceedings of the 9th conference on Computing Frontiers, Caligari, Italy, 22–23 September 2012; p. 183.
12. Das, A.; Wu, Y.; Huynh, K.; Dell'Anna, F.; Catthoor, F.; Schaafsma, S. Mapping of Local and Global Synapses on Spiking Neuromorphic Hardware. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018.

13. Balaji, A.; Das, A.; Wu, Y.; Huynh, K.; Dell'Anna, F.; Indiveri, G.; Krichmar, J.L.; Dutt, N.; Schaafsma, S.; Catthoor, F. Mapping Spiking Neural Networks to Neuromorphic Hardware. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *28*, 76–86. [CrossRef]
14. Li, S.; Guo, S.; Zhang, L. SNEAP: A Fast and Efficient Toolchain for Mapping Large-Scale Spiking Neural Network onto NoC-based Neuromorphic Platform. In Proceedings of the 30th Great Lakes Symposium on VLSI (GLSVLSI 2020), Beijing, China, 27–29 May 2020.
15. Kernighan, B.W.; Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* **1970**, *49*, 291–307. [CrossRef]
16. Karypis, G.; Kumar, V. Multilevelk-way Partitioning Scheme for Irregular Graphs. *J. Parallel Distrib. Comput.* **1998**, *48*, 96–129. [CrossRef]
17. Dayan, P.; Abbott, L. Theoretical Neuroscience : Computational and Mathematical Modeling of neural systems. *Philos. Psychol.* **2001**, *15*, 154–155.
18. Maass, W.; Natschläger, T.; Markram, H. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Comput.* **2002**, *14*, 2531–2560. [CrossRef] [PubMed]
19. Bienenstock, E. A model of neocortex. *Netw. Comput. Neural Syst.* **1995**, *6*, 179–224. [CrossRef]
20. Shen, J.; Liu, J.K.; Wang, Y. Dynamic Spatiotemporal Pattern Recognition with Recurrent Spiking Neural Network. *Neural Comput.* **2021**, *33*, 2971–2995. [CrossRef] [PubMed]
21. Sinha, D.B.; Ledbetter, N.M.; Barbour, D.L. Spike-timing computation properties of a feed-forward neural network model. *Front. Comput. Neurosci.* **2014**, *8*, 5. [CrossRef] [PubMed]
22. Stimberg, M.; Brette, R.; Dan, G. Brian 2, an intuitive and efficient neural simulator. *eLife* **2019**, *8*, e47314. [CrossRef] [PubMed]
23. Haykin, S.; Kosko, B., Gradient-Based Learning Applied to Document Recognition. In *Intelligent Signal Processing*; IEEE: Piscataway, NJ, USA, 2001; pp. 306–351. [CrossRef]
24. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747.
25. Hinton, A.K. Learning multiple layers of features from tiny images. In *Handbook of Systemic Autoimmune Diseases*; Elsevier: Amsterdam, The Netherlands, 2009.