Optimized Mapping Spiking Neural Networks onto Network-on-Chip

Yu Ji[1], Youhui Zhang[1,2,3(✉)], He Liu[1], and Weimin Zheng[1,2]

[1] Department of Computer Science and Technology,
Tsinghua University, Beijing, China
zyh02@tsinghua.edu.cn
[2] Technology Innovation Center at Yinzhou,
Yangtze Delta Region Institute of Tsinghua University, Jiaxing, Zhejiang, China
[3] Center for Brain-Inspired Computing Research,
Tsinghua University, Beijing, China

Abstract. Mapping spiking neural networks (SNNs) onto network-on-chips (NoCs) is pivotal to fully utilize the hardware resources of dedicated multi-core processors (CMPs) for SNNs' simulation. This paper presents such a mapping framework from the aspect of architecture evaluation. Under this framework, we present two strategies accordingly: The first tends to put highly communicating tasks together. The second is opposite, which aims at SNN features to achieve a balanced distribution of neurons according to their active degrees; for communication-intensive and unbalanced SNNs, this one can alleviate NoC congestion and improve the simulation speed more. This framework also contains a customized NoC simulator to evaluate mapping strategies. Results show that our strategies can achieve a higher simulation speed (up to 1.37 times), and energy consumptions can be reduced or rise very limited.

1 Introduction

The great potential of neural systems has aroused research enthusiasms [1]. Neural simulation is one of the important research methods. Typically, the neural network (NN) is expressed as a graph of neurons which take inputs from others and perform computation to produce an output which is, in turn, issued to other neurons. This model is known generally as an Artificial Neural Network (ANN). SNNs (Spiking Neural Networks) can be regarded as the third generation of ANNs [2]: in addition to neuronal and synaptic states, SNNs also incorporate the timing of the arrival of inputs (called spikes) into the operating model. It is believed that SNNs yield higher biological reality and have the potential of more computational power [3].

To speed up SNN simulation, very-large-scale integration (VLSI) systems have been widely used to mimic neuro-biological architectures. In addition, a multi-core

The work is supported by the Science and Technology Plan of Beijing under Grant No. Z161100000216126 and the Brain Inspired Computing Research of Tsinghua University under Grant No. 20141080934.

J. Carretero et al. (Eds.): ICA3PP 2016, LNCS 10048, pp. 38–52, 2016.
DOI: 10.1007/978-3-319-49583-5_3

processor (CMP) with a Network-on-Chip (NoC) has some characteristics similar to those of neural networks, which has emerged as a promising platform for neural network simulation, including [4–7], etc. On the other hand, a neural network usually includes a large number of neurons with a complex connectivity-scheme between them, which is fairly different from common NoC. Consequently, to design an optimized mapping method for distributing biology-analog networks onto hardware is an important topic, confronted with following challenges:

- Besides a large number of neurons, one issue is that each neuron is typically connected to many others; huge amounts of one-to-many communications may take place between the nodes simulating neurons. In contrast, common NoCs tend to feature one-to-one and one-to-a-few connections. Thus, much information about the neuron-connectivity has to be maintained and utilized efficiently.
- To meet timing constraint is another issue. SNNs have taken the arrival time of spikes into the operating model. But common NoCs usually employ the packet-switching technology so that shared resources can result in unwanted variation in transmission latencies, which may impact the accuracy of SNN operations [8].

Fortunately, neural networks also have some friendly characteristics: They show the locality property, i.e., a homogeneous collection of neurons (called a population) tends to connect a nearby population densely (the bundle of single connections between two populations is called a projection). Neurons also tend to fire at a relatively slow rate (measured in terms of hertz and the upper limit is 1000 Hz) while modern electronics operates at multiple gigahertz. Thus, time division multiplexing can be used to make a single processing node or communication channel handle many different neurons or connections at the same time.

This paper proposes a mapping methodology to cover the aforementioned issues, from the perspective of architecture evaluation: neural networks are regarded as parallel tasks and CMPs are the underlying execution substrate, while the mapping mechanism is a middle layer for optimized resource allocation. Furthermore, neuron populations can be regarded as sub-tasks while spikes are communications between them. We will formulate the mapping problem under the framework of application-mapping algorithms for NoCs; thus quite a few architecture-evaluation technologies can be used. In summary, the following contributions have been accomplished:

- We construct a mapping framework. Several existing SNN simulators are used to get golden models and running-traces of objective applications. Trace analyses illustrate that under different inputs, active degrees of one neuron-population are similar. It also contains a configurable NoC simulator for evaluation, which supports some proven effective features for SNN simulation.
- Through the framework, we analyze some existing mapping strategy and point out that transmission of any spike should be completed in one SNN cycle (regardless of the corresponding nominal delay-attribute) to avoid steep increase of the maximum transmission delay of spikes as the simulation speed is fairly high.
- Two mapping strategies are presented. The first is a relatively conventional approach that uses the Kernighan-Lin (KL) algorithm to put highly communicating tasks together. The second is optimized in the opposite direction, which aims at

SNN features to achieve a balanced distribution of neurons according to active degrees. Tests on the NoC simulator show that our strategies can achieve a higher simulation speed (up to 1.37 times) while energy consumptions can be reduced or rise very limited; for communication-intensive and unbalanced SNNs, the second can alleviate NoC congestion and improve the speed more.

2 Related Work

Neuromorphic VLSI systems usually consider neurons as the basic network components connected by directed edges (synapses) and describe the network in terms of neurons, their positions and projections (ignoring biological details). Moreover, communication through spikes is based on a dynamic event-driven scheme.

2.1 Neuromorphic Chips and Network-Mapping

Lots of works have been carried out to simulate SNNs using VLSI technologies. TrueNorth [4, 9] is a digital neuromorphic chip produced by IBM. It also proposed a programming paradigm, Corelet [10]. Moreover, TrueNorth has formulated the problem of mapping as a wire-length minimization problem in VLSI placement [11].

Neurogrid [6] is a brain-inspired analog/digital hybrid system. For mapping, three studies have been carried out: The first maps neuronal models onto neuromorphic hardware [12]; the second maps computations onto heterogeneous populations of spiking neurons based on a theoretical framework (Neural Engineering Framework [13], NEF); the last is to map a network to electronic circuits.

SpiNNaker [5]'s hardware is based on the CMPs of ARM cores. A sequential neuron-core mapping scheme was presented by [2]; [2] also disclaimed that the locality issues were not taken into account. In addition, SpiNNaker has provided technical details of NoC [14]; thus we use it as the reference design.

Dimitrios et al. [15] presented the optimal mapping of a biologically accurate neuron simulator on the Single-Chip Cloud Computer (SCC). But [15] is not a general solution; it is dedicated to inferior olive simulations and the SCC platform.

On the other side, there are quite a few studies [16, 17] on neuromorphic circuits to utilize emerging memory technologies to mimic synaptic behaviors, which are usually focused on prototype construction.

In addition, a few studies have designed customized NoCs to fix the transmission latency; the principle is resource-reservation. For example, EMBRACE [18] proposes such a ring topology for spike communications, which uses a time-stamped broadcast flow control scheme [19]. Philipp et al. [20] use isochronous connections to reserve network bandwidth, which relies on global synchronization of all nodes. Our work is complementary to them from the mapping aspect. Another related job is Vainbrand et al. [21], which performs the analytical evaluation and comparison of different interconnect architectures. It is shown that a multicast mesh NoC provides the highest performance/cost ratio. But no mapping strategy is proposed.

2.2 Application Mapping Algorithms for NoCs

Many studies have been done in application-to-NoC mapping algorithm. Their principles are usually similar: some algorithms have been used to map highly communicating tasks close to each other; and then some heuristic algorithms are used for optimization. For example, Zhu et al. [22] explored opportunities in optimizing application mapping for channel-based on-chip networks, based on the Kernighan-Lin (KL) algorithm. Others include Sahu et al. [23], KLMAP [24], Tosun et al. [25], etc.

As a summary, we compare our work with existing studies; the illuminations from them and our features are given as follows:

1. We adopt some NoC-architecture features that have been proved efficient for SNN simulation, including multicast-enabled routing and mesh topology.
2. From the methodology aspect, we try to carry out SNN studies from the computer architecture perspective and formulate the mapping problem under the application-mapping framework. Further, we have analyzed the limitation and application scope of existing mapping method, and then present our strategy accordingly. Especially, one strategy has extended the above-mentioned mapping framework. As far as we know, no existing study has done this way.

3 The Framework Design

In this section two essential factors of architecture evaluation have been presented first: (1) we introduce how to draw models and running-traces of quite a few objective applications (namely, SNNs), as well as the characteristic analysis; (2) we design and implement the corresponding simulator of NoC that is customized for SNN. In addition, we analyze the limitation of some existing mapping strategy.

3.1 Neural Networks from Software Simulators

Software simulation tools [26] have been widely used by the neuroscientists' community to obtain precise simulations of a given computational paradigm. Thus we can construct accurate neural networks and drive them on the simulators.

An SNN simulator (like NEST [27], Nengo [28], etc.) often provides programming interfaces for users to develop SNN models on the population/projection level; quite a few attributes of populations, connectivity and models can be set respectively.

The SNN model can be defined by user or set by the simulator under user's guidance. After construction, we extract the topological structure with information of nodes and edges. In addition, information of spikes can be obtained during the simulation phase; each record contains the ID of the source neuron and the issuing time. It means we can get the whole information of spikes, namely, running traces.

Then we analyze neurons' active degrees from the traces of some representative networks. Here the active degree for a neuron is defined as the average number of its spike-issues per unit time. For a population, the active degree is the sum of degrees of

all its neurons. Each SNN model is extracted from one of the following two popular simulators: NEST (NEural Simulation Tool) [27] and Nengo [28].

The following 11 representative SNNs have been used; for all of them, the neuron model is the Leaky Integrate-and-Fire (LIF) model and all synaptic weights are fixed.

1. Basal Ganglia [29, 30] (abbreviated to *BG*), which models the basal ganglia. BG contains about 1200 neurons of 26 populations.
2. A Question Answering Network (*Question*). It simulates the question-answering function, which provides the answer by learning examples. This model includes about 8000 neurons and 80 populations, available at Nengo web site.
3. A Controlled Question Answering Network (*QAWC*), which performs question answering based on storing items in a working memory, under control of a basal ganglia. It contains 12000 neurons and 60 populations, available at Nengo web site.
4. RBM Digit Recognition (Digit). It is created by training an RBM Deep Belief Network on the MNIST database. It contains 6000 neurons and 5 populations.
5. Bandit Task [31] (4 NNs), a set of four models to exhibit how a simulated rat responds different environments. It contains four NNs (*arm*, *env*, *halflearn* and *quarterlearn*); each includes more than 1000 neurons and 15–20 populations.
6. Temporal Differentiation (*Diff*). It performs the computation of temporal differentiation [32], which contains 5000 neurons and 3 populations.
7. Spatiotemporal Processing and Coincidence Detection (*Spat*), which aims at simulating connections between the retina and the cochlea, and realizes a co-incidence detector. It has 8500 neurons and 63 populations.
8. Neural Path Integrator [33] (*Path*). It incorporates representations and updating of position into a single layer of neurons. It has 1600 neurons and 12 populations.

We simulate each network for many times with randomly-selected legal inputs, and record the spike-information of each neuron. Without loss of generality, active degrees of neurons of the Question model have been shown in Fig. 1 (others own the similar feature): for clarity, the degree is illustrated in terms of population. The x-axis is population IDs; the y-axis is the active degree of each population. Legends represent different test sequences. Analyses show that each neuron-population's active degrees are similar under different inputs.

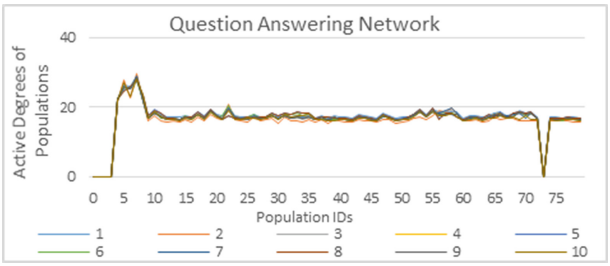


Fig. 1. Active degrees of populations

3.2 NoC of SNN Simulation

Metrics. From the aspect of NoCs' configuration, there is a very large design space. For simplification, we introduce one widely-used design option, multicast-enabled mesh, as the foundation, which has been proved highly efficient for simulation of neural networks [21]. In addition, the 2D-mesh NoC is widely used by CMP products, like SCC [34] and Tile [35]. Accordingly, the tree-based distributed routing is considered for multicast. Compared with the source routing mechanism, it has been proved that the distributed routing will introduce less storage overheads. Moreover, there are quite a few existing multicast routing strategies (Enright et al. [36], Fidalgo et al. [37], Rodrigo et al. [38] and so on) belonging to this category. Accordingly, the size of a multicast routing table will affect the ensuing energy and access time; thus the routing table size is considered a metric in evaluating mapping strategies.

The next metric is simulation speed. Usually, a real-time neuromorphic system means the simulated network is working as 'fast' as the real biological system. Considering a simulated neuron issues one spike per 1 ms (it is the upper bound for biological cells), from the aspect of the NoC electronics operating at hundreds of megahertz or gigahertz, 10^6 or 10^5 cycles will elapse between two issues. Therefore, the simulation speed can be denoted by its times as fast as the real-time speed (1000 Hz). Apparently, with the speed increase, more spikes per unit time will be inserted into the NoC and may cause traffic congestions, which inversely hinders the further speed improvement; otherwise some spike-packets with long latencies will violate timing constraints.

Multicast Routing. For the tree-based routing, a multicast continues along a common path and then branches the message when necessary to achieve a minimal route to each destination. At each hop, the router will complete corresponding operations based on the source ID of the incoming packet. By default, the X-Y routing strategy is used for each single message to avoid dead-lock. Specially, a two-level routing strategy is used as following:

As mentioned above, SNN models of software simulators usually represent projections between populations. Thus it looks beneficial to distribute all neurons of a population into one core as much as possible, or into several nearby cores if one cannot occupy all (it is just the principle of the existing sequential mapping strategy). Accordingly, we take the population ID as the look-up key of routing tables.

The second level is inside a core: on receipt of a spike, the target core will check which internal population should deal with it (if one node contains neurons from multiple populations). It is achieved by looking up a local table; the key is the source-population ID in the incoming packet.

Moreover, we propose that synaptic weights and other attributes are kept at the post-synaptic end (in the aforementioned local table). Hence, no synaptic information needs to be carried in a spike, which makes the multicast mechanism efficient.

Based on the above design, we can give the structure of entries of the routing table on a router, as well as the organization of a NoC-packet. The latter is simple, which just represents the arrival of a spike from some neuron. Correspondingly, a packet consists of only one flit (32-bit width): the first 8 bits represent the population ID; the second

14-bit is the neuron ID in the population and the subsequent 6-bit illustrates the issue time, so that the receiver can judge whether the incoming spike is in time or not. The remaining bits are reserved, which can be used to support inter-chip communication.

Moreover, the common scratchpad memory can be used as the routing table rather than expensive CAM (content addressable memory): population IDs are defined as a series of consecutive integers; thus they can be regarded as memory addresses to access the scratchpad. Accordingly, one entry of the routing table is represented in Table 1. We also use the turn-table routing: if population id is not found in the routing table, the default straight routing will be used. The same method can be applied to the local table, using the population & neuron ID as address.

Table 1. Entry of the routing table

Field	Up	Down	Left	Right	Core	Valid	Reserved
Description	To which directions the package should be transmitted					1/0	2 bits

Storage Consumptions of Routing. As mentioned previously, each router owns a routing table that can be regarded as common on-chip memory; population IDs are used as memory addresses. Therefore, the size of each table is not larger than the amount of populations (denoted as k) and the total consumption n cores is $k \times n$. k is usually limited as shown before.

Implementation of the NoC Simulator. We greatly modify Noxim [39] to implement a detailed, cycle-accurate simulator for NoCs that provides not only the flexibility needed in a high-level simulator but also detailed modeling of all key components of a router. Our modification is focused on the tree-based multicast. We have referenced the micro-architecture design of the VCTM multicast router [36]. Moreover, as a NoC packet contains only one flit, the store-and-forwarding flow control is used, which simplifies the channel management and does not impair the transfer latency. Accordingly, the wire link latency is set to 1 cycle and the maximum routing latency is 2 cycles.

We use the Orion 3.0 tool to get the energy consumption of each pipeline stage. For routing tables, the CACTI tool [40] is used. Now the NoC simulator supports the 2D-mesh topology with different scales and the simulation speed can be configured.

3.3 Preliminary Tests

Owing the framework, we can map representative SNNs onto our NoC simulator and drive it with traces from SNN simulators. Specially, we use the strategy of SpiNNaker as the reference because it presented enough details. Currently SpiNNaker just uses a sequential mapping scheme: Neurons are numbered so that IDs of all neurons in one

population are continuous. Then, they are uniformly distributed to NoC nodes in order. Thus, neurons in one population will be distributed into one core or nearby cores.

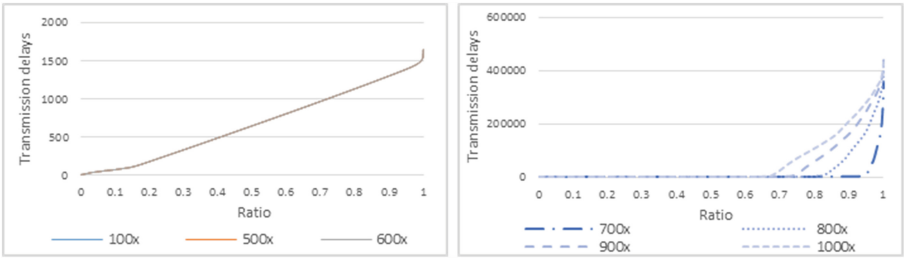
The NoC working frequency is set to 1 GHz while the SNN's is 1 kHz. The number of neurons that one processing node can occupy is set to 64, 128 and 256 respectively.

We study the effect of simulation speed on transmission delay. The distribution of transmission delays under different speeds has been presented. Without loss of generality, QAWC SNN is taken as the example (in Fig. 2) to show the cumulative distribution curve of spike-transmission delay. The y-axis stands for transmission delays and the x is the ratio of the spikes whose delays are less than the calibration value.

From Fig. 2(a), we can see that as the speed is relatively slow (600 times or less), the cumulative distributions are almost the same (in Fig. 2(a) all curves are overlapped with each other): The maximum transmission delay is 1660 NoC cycles while one simulated SNN contains 1667 NoC or more cycles, which means that all spikes can reach target nodes in one SNN cycle. Conversely, if not all spikes issued in one SNN cycle can reach their targets in the same cycle, the maximum transmission delay of spikes will be steeply raised: Fig. 2(b) shows that as the speed is 700 times or more, transmission delays of more than 3% of all spikes will reach 10^5 NoC cycles, far larger than several simulated SNN cycles. The reason lies in that during a short period, the number of issued spike from a population will remain relatively constant. Thus, when the previous case occurs, more and more delayed spikes will be accumulated to exacerbate the symptom.

More analyses can prove this situation: For the packets with long latencies, the ratio of the latency caused by channel-congestion to the total is over 90%. Thus, the effect of congestion on transmission will become vital as the speed is higher.

So far one preliminary conclusion can be drawn: the transmission of any spike should be completed in one SNN cycle to meet the timing constraint, regardless of its nominal delay-attribute that is usually one or several SNN cycles. The inference lies in that the decrease of the maximum transmission delay can improve simulation speed, which is one main target of optimized mapping.



(a) (b)

Fig. 2. Distribution of transmission delays under different speeds

4 Algorithm Design and Evaluations

Based on the aforementioned work, SNN mapping can be formulated under the application-mapping framework for NoCs.

Definition 1: Given an application characteristic graph, $G(V, E)$, it is a directed weighted graph in which $v_i \in V$ represents a task in the application; $e_{i,j} \in E$ represents the connection between v_i and v_j while $b_{i,j}$ is the traffic between v_i and v_j. From the SNN aspect, because we can get the activity degree of each population (as described in Sect. 3.1) that has shown similarity under different inputs, $b_{i,j}$ can be set to the degree of v_i.

Definition 2: Given a NoC topology graph, $P(R, P)$. $r_i \in R$ is a processing node of the NoC; $p_{i,j} \in P$ represents a link between $r_i$and r_j and $h_{i,j}$ is the Manhattan distance from r_i to r_j.

Accordingly, the mapping framework can be formulated as follows: Input $G(V, E)$ and $P(R, P)$ and output a mapping solution to distribute $G(V, E)$ onto $P(R, P)$, which will be evaluated based on the maximum transmission delay.

4.1 Strategy One

We propose a mapping algorithm that tends to put densely-communicating populations close together, and formulate it under the above-mentioned mapping framework.

Allocating highly-communicating populations onto nearby cores also accords with the principle of existing NoC mapping strategies [22–24] that map highly communicating tasks together. In addition, TrueNorth has formulated the problem of mapping neurons to cores as a wire-length minimization problem in VLSI placement [11], whose principle is similar with ours. Therefore, we outline this strategy here.

Without loss of generality, we use the Kernighan-Lin (KL) partitioning strategy[1] as the starting point. It bipartitions a set of modules, so that highly connected modules are kept in one partition. This procedure is applied (recursively and alternately along the two directions of 2D-mesh NoC, '*x*' and '*y*') till only the closest two nodes are left in any of the final partitions in a mesh. The motivation for using this algorithm is that the cores with more communication requirement should be attached nearby routers in the NoC. During the mapping phase, these partitions are taken into consideration to minimize the communication cost between mapped cores.

As the result depends on the initial partitioning, we run it for T (preset) times, each starting with a randomly generated initial partition. The best one is used for subsequent improvement; here the *best* means the sum of traffic ($b_{i,j}$) of the edges across partitions is the smallest. Afterward, the simulated annealing (SA) algorithm is used for optimization; its energy function is the total hop count of transmission.

[1] https://en.wikipedia.org/wiki/Kernighan%E2%80%93Lin_algorithm.

The whole workflow is given below:

Step 1: To draw the connection matrix of a SNN (we can get them from models on a SNN simulator, as mentioned in Sect. 3.1).
Step 2: To distribute neuron-populations to cores. It distributes all neurons in a population to a core as much as possible, or to nearby cores if one cannot occupy all.
Step 3: Placement of cores based on the previous algorithms.
Step 4: Based on Step 2 & 3, all table entries of each router can be filled: a multicast packet will follow the X-Y routing to complete common path before branch.

4.2 NoC Congestion

Now we carry out the evaluation of the first mapping strategy through our simulator. The simulation configuration is just the same as those of Sect. 3.3.

Results that for most models, the KL&SA algorithm improve the simulation speed. In other words, it can often decrease the maximum transmission delay because it tends to reduce the communication cost between mapped cores; results are presented in Table 2. The interesting point lies in that for Diff and Spat, the delay increases.

Detailed analyses show that as the KL&SA algorithm may aggravate the traffic congestion and lead to performance degradation: the default mapping principle tends to put neurons from the same population together and such neurons usually share the same set of destinations. In each simulated SNN cycle, those cores that contain active populations may produce a lot of packets, which often share the same destinations and may be blocked in the channel between the core and router or in some local queues.

Table 2. Maximum transmission delays (the number of neurons in one node is 64)

SNN model	SpiNNaker's sequential mapping	KL&SA
arm	128	80
env	123	115
halflearn	121	85
quarterlearn	136	110
bg	406	291
path	663	627
digit	165	125
diff	**994**	**1034**
spat	**2035**	**2358**
question	2460	1969
qawc	1648	1606

For example, Fig. 3 gives the average spike-processing time of each route for the two SNNs. We can see that after KL&SA, the maximum processing time is longer and the average delay on each router is more unbalanced. In addition, according to

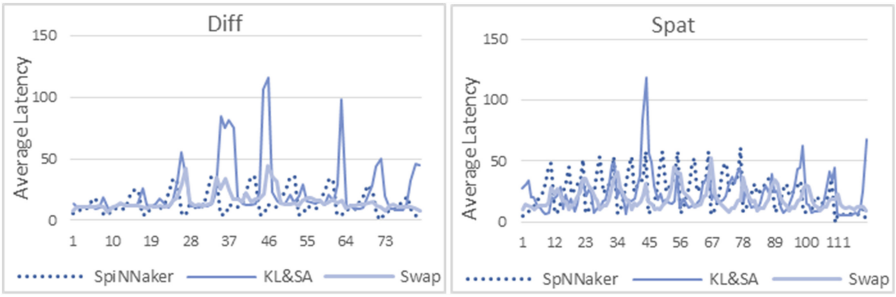


Fig. 3. Average transmission-delays of each router (the unit of the Y-axis is a NoC cycle; the X-axis is the router ID. Three strategies)

spike-traces, these two SNNs are the most active: their average active degrees per neuron are 398 and 57 respectively. In contrast, the maximum degree of others is only 16. Thus, it indicates that for communication-intensive SNNs, some local congestion has happened to impede the improvement of simulation. Therefore, we present a new mapping strategy in the next subsection.

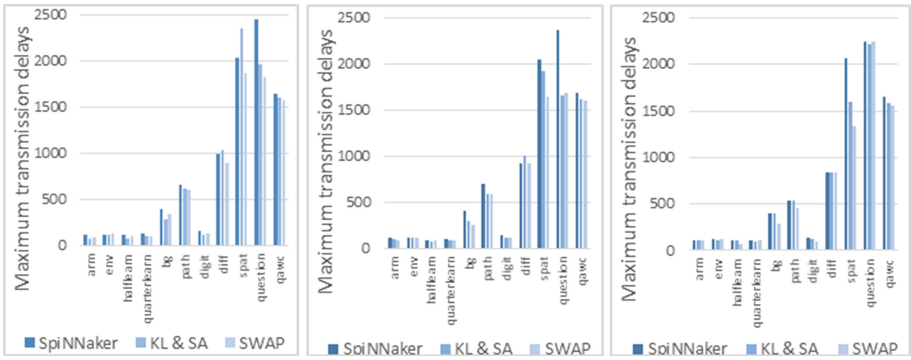
4.3 Optimization in the Opposite Way

Under the existing application-mapping framework for NoCs, normally, any subtask is sequential and cannot be divided further. But SNN populations are different: they are divisible. Accordingly, we present a new strategy to reduce congestion. The principle is to swap neurons with each other in populations with different active degrees, to achieve a balanced distribution. As there is no connection among a single population, it will not introduce extra spikes.

However, the swap mechanism makes populations fragmented: neurons of a single population will be distributed into more cores, which causes more communications and longer paths (it is why we call it optimization in the opposite way). Thus, there is a tradeoff between the fragmentation and balance of active degrees: apparently, if the gap of two populations' degrees is limited, it is unnecessary to switch their neurons.

Accordingly, after the KL&SA mapping, we first sort all cores in a queue based on their active degrees. Second, the most active and most inactive cores will exchange half of neurons before removed from the queue, if the ratio of their degrees is larger than a threshold. This procedure will repeat till there is no exchange or all populations have been browsed. For the threshold, our test shows that 2 is a proper value. From the aspect of routing tables, the storage consumption is fixed, as we use the on-chip memory for storage and use population IDs as memory addresses.

After swap, the imbalance is weakened (in Fig. 3, too). Figure 4 gives the maximum delays under different mapping strategies: For cases of 64 neurons per node, the swap strategy decreases the maximum delay for 10 SNNs and 7 of them outperform the KL&SA; the latter decreases the maximum delay of 9 SNNs. For cases of 128, the three values are 11, 7 and 9 respectively. For cases of 256, they are 11, 8 and 11.



(a) 64 neurons per node (b) 128 neurons per node (c) 256 neurons per node

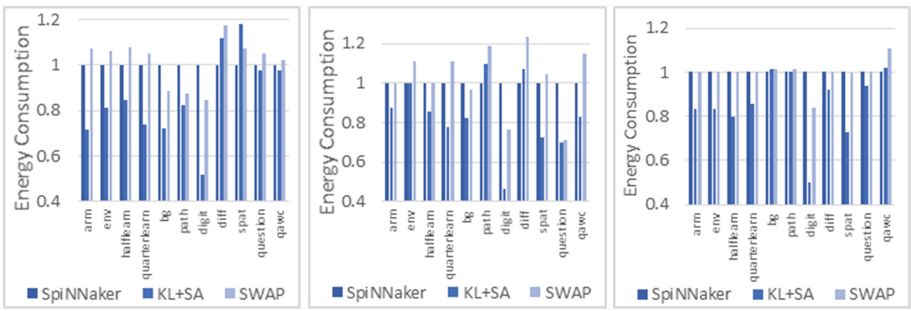
Fig. 4. Maximum delays of transmission

For KL&SA, the maximum improvement is 37% and the average improvement is 11%. For swap, the maximum is 36% and the average is 16%.

Statistics also show that compared with the sequential mapping, the KL&SA algorithm decreases the average transmission delay in all cases: For cases of 64 neurons per node, the average reduction is 4.3%; for 128 the reduction is 7.3%; for 256, it is 9.6%. For the swap strategy, because it makes populations fragmented, the average delay in cases of 128 increases a little, by 1.1%. In cases of 64 and 256, it decreases by 6.7% and 6.0% respectively.

Corresponding energy consumptions are presented in Fig. 5. The swap strategy causes more communications and more energy consumptions: compared with linear mapping, it increases the consumption by 1.7% averagely for cases of 64 neurons per node; for cases of 128, the average increase is 2.4%; for 256, the value is almost the same. KL&SA decreases the consumptions, by 14%, 16% and 14% respectively.

As a summary, two mapping optimizations are presented. The KL&SA tends to put highly communicating tasks close while the swap tries to achieve the balance in terms of populations' active degree. From the aspect of the simulation speed (the maximum



(a) 64 neurons per node (b) 128 neurons per node (c) 256 neurons per node

Fig. 5. Energy consumptions of a simulated second

delay), the swap achieves better results. However, it consumes more energies because of fragmentation. Anyway, it gives a new optimization direction, considering the tradeoff between simulation speed and energy consumption. Specially, it is more suitable for communication-intensive and unbalanced SNNs.

5 Conclusion

This paper presents a methodology for the SNN-to-NoC mapping problem, from the aspect of architecture evaluations. Based on analyses of running traces from neural network simulators that model representative networks, we find that distributions of neurons with diverse active degrees show similarities to a great extent. Accordingly, we formulate the SNN mapping problem under the application-mapping framework for NoCs. We also present strategies that are optimized in the opposite directions, which extend the existing mapping framework.

We believe it not only benefits the exploration of design space but also bridges the gap between applications and neuromorphic hardware.

References

1. National Academy of Engineering: Reverse-Engineer the Brain (2012). http://www.engineeringchallenges.org/cms/8996/9109.aspx
2. Jin, X.: Parallel simulation of neural networks on spinnaker universal neuromorphic hardware. Ph.D. thesis, University of Manchester (2010)
3. Paugam-Moisy, H., Bohte, S.: Computing with spiking neuron networks. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 335–376. Springer, Heidelberg (2012)
4. Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., et al.: A million spiking-neuron integrated circuit with a scalable communication network and interface. Science **345**(6197), 668–673 (2014)
5. 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. **62**(12), 2454–2467 (2013)
6. Benjamin, B.V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A.R., Bussat, J.-M., Alvarez-Icaza, R., Arthur, J.V., Merolla, P.A., Boahen, K.: Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. Proc. IEEE **102**(5), 699–716 (2014)
7. http://brainscales.kip.uni-heidelberg.de/
8. Pande, S., Morgan, F., Smit, G., Bruintjes, T., Rutgers, J., McGinley, B., Cawley, S., Harkin, J., McDaid, L.: Fixed latency on-chip interconnect for hardware spiking neural network architectures. Parallel Comput. **39**, 357–371 (2013)
9. Seo, J.S., Brezzo, B., Liu, Y., et al.: A 45 nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In: IEEE Custom Integrated Circuits Conference (CICC) (2011)
10. Esser, S.K., Andreopoulos, A., Appuswamy, R., Datta, P., Barch, D., Amir, A.: Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores. In: The International Joint Conference on Neural Networks (2013)

11. Akopyan, F., Sawada, J., Cassidy, A., et al.: TrueNorth design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Trans. Comput. Aided Des. Integr. Circ. Syst. **34**(10), 1537–1557 (2015)
12. Gao, P., Benjamin, B.V., Boahen, K.: Dynamical system guided mapping of quantitative neuronal models onto neuromorphic hardware. IEEE Trans. Circ. Syst. **59**(11), 2383–2394 (2011)
13. Eliasmith, C., Anderson, C.H.: Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. A Bradford Book, Cambridge (2004)
14. Davies, S., Navaridas, J., Galluppi, F., Furber, S.: Population-based routing in the SpiNNaker neuromorphic architecture. In: International Joint Conference on Neural Networks (IJCNN 2012), Brisbane, Australia, 10–15 June 2012
15. Carrillo, S., Harkin, J., McDaid, L.J., Morgan, F., Pande, S., Cawley, S., McGinley, B.: Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations. IEEE Trans. Parallel Distrib. Syst. **24**(12), 2451–2461 (2013)
16. Rodopoulos, D., Chatzikonstantis, G., Pantelopoulos, A., Soudris, D., De Zeeuw, C.I., Strydis, C.: Optimal mapping of inferior olive neuron simulations on the single-chip cloud computer. In: 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (2014)
17. Prezioso, M., Merrikh-Bayat, F., Hoskins, B.D., Adam, G.C., Likharev, K.K., Strukov, D.B.: Training and operation of an integrated neuromorphic network based on metal-oxide memristors. Nature **521**, 61–64 (2015)
18. Wendt, K., Ehrlich, M., Schüffny, R.: A graph theoretical approach for a multistep mapping software for the FACETS project. In: 2nd WSEAS International Conference on Computer Engineering and Applications (CEA 2008) (2008)
19. Pandea, S., Morgan, F., Smitb, G., Bruintjesb, T., Rutgersb, J., McGinleya, B., Cawleya, S., Harkin, J., McDaid, L.: Fixed latency on-chip interconnect for hardware spiking neural network architectures. Parallel Comput. J. (Elsevier) **39**, 357–371 (2013)
20. Philipp, S., Grübl, A., Meier, K., Schemmel, J.: Interconnecting VLSI spiking neural networks using isochronous connections. In: Sandoval, F., Prieto, A., Cabestany, J., Graña, M. (eds.) IWANN 2007. LNCS, vol. 4507, pp. 471–478. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73007-1_58
21. Vainbrand, D., Ginosar, R.: Scalable network-on-chip architecture for configurable neural networks. Microprocess. Microsyst. **35**, 152–166 (2011)
22. Zhu, D., Chen, L., Yue, S., Pedram, M.: Application mapping for express channel-based networks-on-chip. In: Proceedings of Design, Automation and Test in Europe, DATE (2014)
23. Sahu, P.K., Manna, N.S., Chattopadhyay, S.: Extending Kernighan-Lin partitioning heuristic for application mapping onto Network-on-Chip. J. Syst. Archit. **60**(7), 562–578 (2014)
24. Nisarg, S., Kanchan, M., Santanu, C.: An application mapping technique for butterfly-fat-tree network-on-chip. In: Proceedings of 2nd International Conference on Emerging Applications of Information Technology, EAIT, pp. 383–386 (2011)
25. Tosun, S.: Cluster-based application mapping method for Network-on-Chip. Adv. Eng. Softw. **42**(10), 868–874 (2011)
26. Brette, R., Rudolph, M., Carnevale, T., et al.: Simulation of networks of spiking neurons: a review of tools and strategies. J. Comput. Neurosci. **23**(3), 349–398 (2007)
27. Plesser, H.E., Eppler, J.M., Morrison, A., Diesmann, M., Gewaltig, M.-O.: Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 672–681. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74466-5_71
28. The Nengo Neural Simulator. http://www.nengo.ca/

29. Alexander, G.E., Crutcher, M.D.: Functional architecture of basal ganglia circuits: neural substrates of parallel processing. Trends Neurosci. **13**(7), 266–271 (1990)
30. Redgrave, P., Prescott, T.J., Gurney, K.: The basal ganglia: a vertebrate solution to the selection problem? Neuroscience **89**(4), 1009–1023 (1999)
31. Stewart, T.C., Bekolay, T., Eliasmith, C.: Learning to select actions with spiking neurons in the basal ganglia. Front. Neurosci. **6**, 2 (2012)
32. Tripp, B.P., Eliasmith, C.: Population models of temporal differentiation. Neural Comput. **22**(3), 621–659 (2010)
33. Conklin, J., Eliasmith, C.: A controlled attractor network model of path integration in the rat. J. Comput. Neurosci. **18**, 183–203 (2005)
34. Mattson, T.G., Van der Wijngaart, R.F., Lehnig, T., Brett, P., Haas, W., Kennedy, P.: The 48-core SCC processor: the programmer's view. In: Proceedings of 2010 International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, LA (2010)
35. Wentzlaff, D., Griffin, P., Hoffmann, H., Bao, L., Edwards, B., Ramey, C., Mattina, M., Miao, C.-C., Brown III, J.F., Agarwal, A.: On-chip interconnection architecture of the tile processor. IEEE Comput. Soc. **27**, 15–31 (2007)
36. Jerger, N.E., Peh, L.-S., Lipasti, M.: Virtual circuit tree multicasting: a case for on-chip hardware multicast support. In: ISCA (2008)
37. Fidalgo, P.A., Puente, V., Gregorio, J.A.: MRR: enabling fully adaptive multicast routing for CMP interconnection networks. In: HPCA (2009)
38. Rodrigo, S., Flich, J., Duato, J., Hummel, M.: Efficient unicast and multicast support for CMPs. In: MICRO, pp. 364–375 (2008)
39. Noxim - the NoC Simulator. http://noxim.sourceforge.net/
40. CACTI - An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. http://www.hpl.hp.com/research/cacti/