Ji Y, Zhang YH, Zheng WM. Modelling spiking neural network from the architecture evaluation perspective. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 31(1): 50–59 Jan. 2016. DOI 10.1007/s11390-016-1611-0

Modelling Spiking Neural Network from the Architecture Evaluation Perspective

Yu Ji, You-Hui Zhang, *Member, CCF, ACM, IEEE*, and Wei-Min Zheng, *Fellow, CCF, Member, ACM, IEEE*

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

E-mail: maple.jiyu@hotmail.com; {zyh02, zwm-dcs}@tsinghua.edu.cn

Received July 15, 2015; revised November 19, 2015.

Abstract The brain-inspired spiking neural network (SNN) computing paradigm offers the potential for low-power and scalable computing, suited to many intelligent tasks that conventional computational systems find difficult. On the other hand, NoC (network-on-chips) based very large scale integration (VLSI) systems have been widely used to mimic neurobiological architectures (including SNNs). This paper proposes an evaluation methodology for SNN applications from the aspect of micro-architecture. First, we extract accurate SNN models from existing simulators of neural systems. Second, a cycle-accurate NoC simulator is implemented to execute the aforementioned SNN applications to get timing and energy-consumption information. We believe this method not only benefits the exploration of NoC design space but also bridges the gap between applications (especially those from the neuroscientists' community) and neuromorphic hardware. Based on the method, we have evaluated some typical SNNs in terms of timing and energy. The method is valuable for the development of neuromorphic hardware and applications.

Keywords spiking neural network, network-on-chip, architecture simulation

1 Introduction

Although it is still not yet clear about how the brain works, the great potential of neural systems has aroused research enthusiasms. In addition to the discovery of the brain's computational paradigm, these studies offer the possibility to implement neuromorphic circuits with high parallelism and low power consumption compared with the traditional von Neumann computer paradigm.

It is a well-established idea that the information processing in the mammalian brain relies on the electric discharges (spikes) of neurons, which interact at specialized points of contacts, the synapses. A generated spike arrives at one or more target neurons after a delay of a few milliseconds and causes a small change in the neurons' membrane potentials. A standard approach to neural modeling is to consider neurons as the basic network components and describe the network in terms of the neurons and their positions and projections.

SNN (Spiking neural network) is such an abstraction of real neural networks, which is also believed to yield higher biological realism and has the potential of more computational power[1]. SNNs attempt to emulate the information processing in the mammalian brain based on massively parallel populations of neurons that communicate via spike events. In addition to neuronal and synaptic states, they also incorporate the timing of the arrival of spikes into the operating model.

Accordingly, SNNs are widely used in the community of computational neuroscience. It is important to note that different from artificial neural networks (ANNs) that own complete mathematical paradigms and have become an important branch of computer science, SNNs lack common mathematical models. Therefore, simulations are used to investigate models of the nervous system at functional or process levels in computational neuroscience. A lot of efforts have been put into developing appropriate simulation tools and techniques[2].

On the other hand, very large scale integration

Regular Paper

Special Section on Computer Architecture and Systems with Emerging Technologies

The work is supported by the Science and Technology Plan of Beijing, titled "Research on Efficient Parallel Acceleration Technology for Cognitive Computing Platform", and the Brain Inspired Computing Research of Tsinghua University under Grant No. 20141080934.

(VLSI) systems have been widely employed to mimic neuro-biological architectures (called neuromorphic engineering). In addition, a multi-core processor (CMP) with a network-on-chip (NoC) has many characteristics similar to those of neural networks, which has emerged as a promising platform for neural network simulations. There is quite a little such work, including [3-6], etc.

This is an interdisciplinary field. From the traditional aspect of computer architecture, the design of neuromorphic circuits is promising to develop a new type of computers and the computational models of SNNs can be regarded as the objective applications. Accordingly, we try to bridge the gap between applications from the neuroscientists' community and the neuromorphic hardware. The principle is simulation-centric: existing software simulators widely used in the neuroscientists' community are utilized to extract accurate SNN models and/or running traces. Moreover, we give a configurable NoC simulator to emulate the hardware in design and to execute applications, which supports some proven effective features for SNN simulation (like multicast). Namely, we try to model SNNs from the architecture perspective and then quite a few architecture evaluation methods can be used to explore the design space.

In summary, the following contributions are accomplished.

1) We design a workflow to simulate SNN models on the micro-architecture level. In detail, we give the method to extract models from existing software simulators and drive them on the architecture-level simulator.

2) A trace-driven configurable NoC simulator is implemented for SNN evaluation.

3) Based on the previous work, we evaluate some typical SNNs in terms of timing and energy, which proves the feasibility of this methodology and its ability to guide the development of real hardware.

Moreover, we believe this method can be used as the joint for interdisciplinary research. Some development interfaces will be open for researchers from other fields to integrate the new computational models of neurons/synapses and the behavior models of new neuromorphic devices in the near future.

2 Related Work

Although there are some criticisms, neuromorphic engineering is still a promising direction to explore a new computer architecture beyond von-Neumann architecture. It is necessary to note that there is quite a little work[7-10] on custom architectures to accelerate AI (artificial intelligence) algorithms, like machine-learning algorithms of convolutional and deep neural networks. In contrast, we focus on the simulation of biological neuron networks, not algorithm-acceleration.

2.1 Hardware Spiking Neural Network Systems

Fidjeland *et al.*[11] presented NeMo, a platform for SNN simulations which achieves high performance through the use of highly parallel hardware in the form of graphics processing units (GPUs). A time multiplexed FPGA (field programmable gate array) embedded processor SNN implementation reports 4.2K neurons and more than 1.9M synapses[12]. This system relies on the external memory for spike transfer management.

The FACETS (Fast Analog Computing with Emergent Transient States)[13] project and its successor BrainScaleS[6] have produced wafer-scale IC systems for neural networks. A mixed-signal SNN architecture of 2 400 analogue neurons implemented by using switched capacitor technology and communicating via an asynchronous event-driven bus has been reported.

SpiNNaker[4]'s hardware is based on the CMPs of ARM processors. From the logic view, the SpiNNaker architecture is an array of computation nodes arranged in a 2D triangular mesh, wrapped into a torus. Each NoC node in the SpiNNaker system models 1 000 leaky-integrate-and-fire (LIF) neurons, each having 1 000 synapse inputs.

One of the latest products is TrueNorth[3,14], a fully digital neuromorphic chip developed by IBM in 2014. This chip includes 4 096 neurosynaptic cores. Each core brings memory ("synapses"), processors ("neurons"), and communication ("axons") together and is connected with each other via a 2D-mesh NoC. Moreover, IBM has proposed a programming paradigm, Corelet (including the programming language, library, etc.)[15], to represent some cognitive algorithms as the networks of neurosynaptic cores. According to its open literature, this representation is completed manually, under the help of Corelet library and an architectural simulator (Compass).

EMBRACE[16-17] is a compact hardware SNN architecture for embedded computing platforms. It owns a hierarchical architecture. The on-chip communication is based on a customized array of configurable NoC routers. Specially, it follows the modular neural net-

work (MNN) computing paradigm[18] to partition complex application tasks into smaller subtasks executing on distinct neural network modules.

All above-mentioned studies try to design a new computer architecture inspired by the brain. Thus, architecture evaluation technologies (including hardware simulation) have been used. For example, IBM implemented the Compass simulator for TrueNorth, which played a key role in the construction of the new computing ecosystem. EMBRACE presented a System-C based NoC-simulator for performance analysis. FACTES constructed a comprehensive workflow for modeling SNNs on its hardware, which integrates a detailed simulation of the final hardware platform.

This paper can be regarded as complementary to existing work. Our objective is to make it easier to port the applications from the neuroscientists' community to the neuromorphic hardware through architecture simulation. A similar strategy has been used by SpiNNaker. However, the difference lies in that SpiNNaker uses ARM cores to execute software models of neurons/synapses directly. Although software-based solutions are flexible, it costs more overhead. As the computational model of single neuron is well developed, dedicated ASIC units can provide lower power consumption and better performance. Our proposal mainly focuses on solutions based on dedicated ASIC units for neural simulation.

2.2 NoC Simulators

The general trend in the SoC/processor development has moved from dual- and quad-core processor chips to ones with tens or even hundreds of cores that are connected by network-on-chip (NoC). This requires an NoC-based design to ensure an optimum management of the transit of internal data. To facilitate the development of systems containing a network-on-chip, several dedicated tools have been proposed, including NoC simulators. Some typical ones are presented as follows.

NS-2① was first developed for prototyping and simulating ordinary computer networks. However, since NoCs share many characteristics with classic networks, NS-2 was widely used by many NoC researchers to simulate NoCs[19].

DARSIM[20] allows the simulation of mesh NoC architectures of two and three dimensions. It offers a multitude of NoC simulation configurations with various parameters. This includes two generation modes of data generation: trace-driven and execution-driven.

Noxim② is developed in SystemC language. It allows the user to define a 2D mesh NoC architecture with various parameters. Noxim allows the evaluation of NoCs in terms of throughput, latency and power consumption.

ORION[21-22] is a simulator dedicated primarily to the estimation of power and space for NoCs architectures. It has integrated the support for new semiconductor technology through the models of transistors and capacitances upgraded from industry.

3 Simulation Workflow

In the common process of developing a new computer architecture, the first step is to locate objective applications (or benchmarks) and then extensive simulations will be carried out to explore the design space for adaptation before the hardware design and production.

We follow this methodology, which contains two main tasks. The first is to extract accurate models from some representative SNN software simulators as the objective applications. The second is an SNN-specific NoC simulator used to execute applications. Others include mapping SNNs onto NoC and so on.

3.1 Neural Networks from Software Simulators

The neuronal networks in brains can be described as weighted, directed graphs, with neurons as nodes and synaptic connections as edges. Neurons communicate by sending and receiving spikes through the connections (synapses). For large neuronal networks, the geometric and biophysical complexity of individual neurons can be neglected. Each neuron is described as point-like object with a dynamic state. The common state variable is the membrane potential, which is influenced by spikes that arrive at the neuron's synapses. As it crosses a threshold, the neuron issues a spike, which is transmitted to all connected neurons with a delay. Each connection can have a different delay and weight.

Accordingly, existing neural system simulators support detailed or simple representations of the neurons/synapses to form the weighted and directed graph. They also provide programming interfaces for users to

①http://www.isi.edu/nsnam/ns/, Nov. 2015.

②http://www.noxim.org/, Nov. 2015.

develop SNN models. Usually the model representation operates on the population (group of homogeneous neurons)/projection (bundle of single connections between populations) level rather than on the single-neuron/connection level. This strategy is not just for development simplification. It also accords with the real organization of our brains. In detail, the brain is a heterogeneously structured system, composed of numerous neuron areas which are distinguished by their anatomical, physiological, and functional properties. The structure of the brain itself also conveys important information.

For example, Fig.1 presents the pseudo-code that creates two neuron-populations and a projection connecting them together. Besides the simple mode in Fig.1, more schemes of the connection mode and attribute-configurations have been supported.

```
// Two populations have been created; the neuron
// number and type have been set
pre = Population (1000, one_neuron_type)
post = Population (500, another_neuron_type)
// An all-to-all-connection projection between two
// populations has been created; then the weight and
// latency of each connection are set.
excitatory_connections = Projection (pre, post,
    AllToAllConnector (), StaticSynapse (weight = 0.13))
excitatory_connections.set (delay = 0.4)
                    ⋮
```

Fig.1. Simple example of SNN.

Thus, the following information can be extracted from SNN simulators.

Node Information. It contains the neuron type and class, as well as the ID of the population it belongs to (in the current design, the population ID is of 8-bit width). The former indicates that it is a functional node or a common neuron; the latter is the neuron model, like leaky integrate-and-fire, Hudgkin-Huxley③. In addition, a functional node means the node is just a direct-current generator (for input neurons), not a real neuron. Each node is identified by an integer ID.

Edge Information. It contains the IDs of its source and target nodes, as well as the weight and delay. A connection between a neuron and a functional node can be regarded as an edge with 0-delay and 1-weight.

It is necessary to point out the relationship between edges and nodes. As mentioned above, SNN-model representation operates on the population/projection level. Moreover, for almost all SNN examples we have studied, the connection mode between populations is all-to-all. Thus, the edge information can be described on the population level (the weight of each connection of a single projection can be different). For other connection modes (like the random connection mode in which the existence of a connection of two neurons from different populations depends on a probability), 0-weight edges will be introduced. Thus, we can normalize all modes.

In addition, the information of issued-spikes by neurons can be obtained during the simulation procedure. Each log contains the ID and issuing time of the source neuron. Then, we can get the whole information of spikes, namely, running traces.

3.2 NoC Aspect

NoC is a common and mature technology that connects IPs on chip together. Thus, we should focus on some features that are SNN-specific.

As we know, in SNN, one neuron is typically connected to many others. Thus huge amounts of one-to-many communications must take place between processing nodes. Therefore, multicast-enabled routing looks highly efficient for the simulation of neural networks, which is also proven by some existing work. For example, Vainbrand and Ginosar[23] showed that among common topologies, multicast mesh NoC provides the highest performance/cost ratio and consequently it is one of the most suitable interconnect architectures for configurable neural network implementation. In addition, for SpiNNaker[4], a multicast mechanism is provided for efficient one to many communications.

Furthermore, as a case study, we focus on the mesh NoCs because the topology is widely used by CMP products. Accordingly, the tree-based distributed routing method 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 well-known multicast routing strategies[24-26] belonging to this category.

For the tree-based routing, a multicast continues along a common path and then branches (replicates) 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

③Different models also contain different parameters, like threshold voltage, leakage voltage, and recovery period.

strategy is used for each single message to avoid deadlock. Specially, a two-level routing strategy is used and we give the outline here.

As mentioned in Subsection 3.1, SNN models of software simulators usually represent bundles of single connections (projections) between populations. Thus it is beneficial to distribute as many as possible neurons of a population into one core, or into several nearby cores if one cannot occupy all of them. This strategy can decrease communication overheads.

Accordingly, we take the population ID as the lookup key of routing tables. A spike then can be directed (or duplicated first and then directed) to the target set of nodes (or from the aspect of SNNs, to one or more cores that occupy the neurons of the connected population).

The second level is inside a core. On receipt of a spike, neurons in the target core will check whether this spike should be dealt with by itself or not, as one node may contain neurons from multiple populations. It is achieved by looking up a local table. The key of this table is the source-neuron ID carried by the incoming packet as the payload.

The last task is to fill in routing tables, which depends on the strategy of mapping SNN onto NoC. Similar to SpiNNaker's method, a linear mapping algorithm is used.

First, all neurons are re-numbered so that the IDs of all neurons in a single population will be continuous. Second, neurons are uniformly allocated to NoC nodes in order. Thus, all neurons of a population will be distributed into one core or into several nearby cores (Fig.2).

4 Implementation and Evaluations

4.1 Model Extraction

Now we can extract SNN models from the following software simulators.

NEST (Neural Simulation Tool)④ is able to model different neurons, along with various synaptic models and plasticity methods.

Nengo⑤, the tool used to implement the neural engineering framework principles[27]. It affords the user the possibility to map a wide range of neurocomputational dynamics to spiking neural networks.

Moreover, there are some open resources of computational models of SNNs (like ModelDB[28]). Each model is associated with some simulators. Therefore, using open simulators and models, we can construct accurate neural networks and drive them.

By way of illustration, we refer to the extraction of NEST that has provided friendly interfaces for such model extraction, including the node and edge information. The pseudo-code for node information is presented in Fig.3.

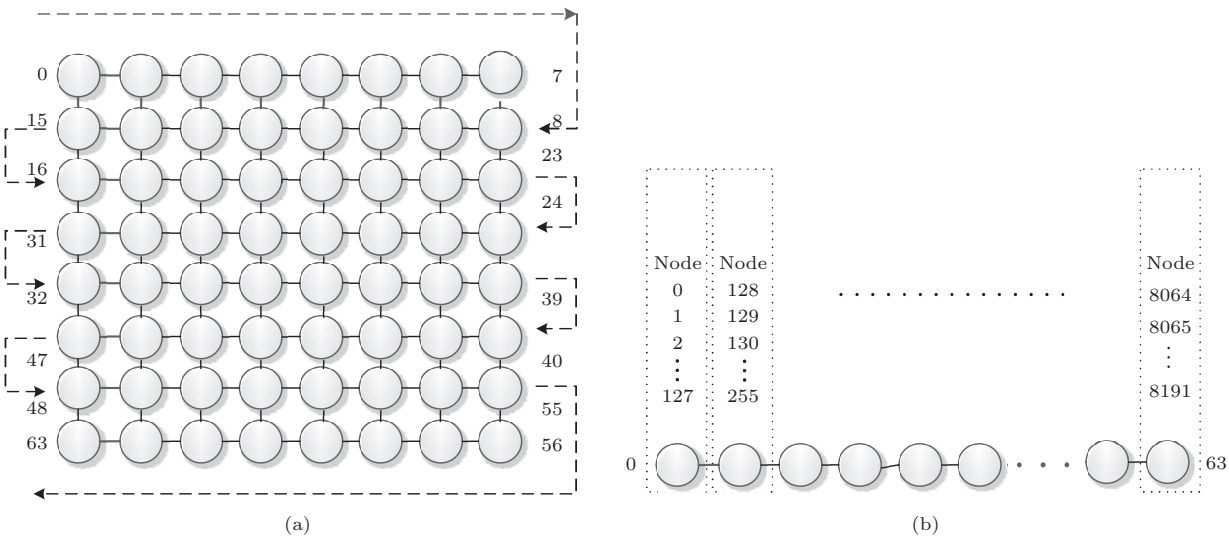

Fig.2. Mapping neuron nodes to NoC. (a) Processing node IDs of an 8×8 2D-mesh NoC. (b) Distribution of ordered neurons. Here we assume that one NoC node can simulate 128 neurons.

④http://www.nest-simulator.org/, Nov. 2015.

⑤http://www.nengo.ca/, Nov. 2015.

```
//Get the kernel status of NEST
kernel_status←NESTGetKernelStatus()
//Get the number of nodes and the node list
size←kernel_status['network_size']
node_status_list←NESTGetStatus(tuple(size))
//Get the node information from the list for analyses
foreach(node_status in node_status_list):
  print AnalyzeNode(node_status)
end foreach

//Classify the node and then get the corresponding information
function AnalyzeNode(node)
  if node['element_type'] == 'neuron' then
    return FilterInformation (node, feature_neuron)
 else if node['element_type'] == 'stimulator' and
 node['model'] == 'dc_generator' then
    return FilterInformation (node, feature_dc_generator)
 else
  //more types
  ...
 end if
end function

//Get the needed features of the input node
function FilterInformation(node, feature_list)
  result←empty list
  foreach(feature in feature_list)
    result.append(node[feature])
  end foreach
  return result
end function
```

Fig.3. Pseudo-code for node extraction.

4.2 Simulator

We refer to Noxim[22] to implement a trace-driven cycle-accurate simulator for NoCs, which also supports the evaluation of power consumption.

4.2.1 Router Pipeline

Our work is focused on the tree-based multicast routing. We refer to the micro-architecture design of the VCTM multicast router[24].

The design foundation is a common four-stage router pipeline. The first stage is the buffer write (BW). The routing computation (RC) occurs in the second stage. In the third stage, virtual channel allocation (VA) and switch allocation (SA) are performed. In the fourth stage, the flit traverses the switch (ST). Each pipeline stage takes one cycle followed by one cycle to do the link traversal (LT) to the next router. As one packet is just one flit, no speculation optimization is used.

For multicast packets, the processing is simpler than VCTM because multicast virtual circuit is decided during the mapping process, rather than the run time. Namely, the contents of routing tables are fixed. In detail, the original routing computation stage is replaced with the operation of routing-table look-up. If a packet is replicated at this node, this stage will be repeated till all replications have been inserted into the next stage.

Moreover, the common scratchpad memory can be used as the routing table, instead of the 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 scratchpad⑥. One entry of the routing table just contains the operation type (8-bit) that represents a type in Table 1 or a combination of them.

Table 1. Operation Types

Operation Type	Description
Turn_up	It will be transmitted to the up node
Turn_down	It will be transmitted to the down node
Turn_left	It will be transmitted to the left node
Turn_right	It will be transmitted to the right node
Sink	Approaching destination
Valid	1/0
Reserved	2 bits

Now the NoC simulator supports the 2D-mesh topology with the stall-and-forwarding flow control. The number of virtual channels (for each direction, the default number is 5) and the NoC scale can be configured.

4.2.2 Power Consumption

We use the Orion 2.0 tool to get the power consumption of each pipeline stage.

The number of table-entries that each router contains should be equal to the number of populations, denoted as k. For an SNN that can be occupied in one NoC chip, k is usually limited. Accordingly, we can use the CACTI⑦ tool to get the power consumption of table accesses.

The default technology is 45 nm CMOS and the running frequency is 1 GHz. The SNN's frequency is 1 KHz, which is the upper bound for biological cells.

4.3 Evaluations

Models and running traces of the following 11 SNNs have been extracted (in the 8th item, there are four models).

1) *RBM Digit Recognition* (*Digit*). It is a model for digit recognition, created by training an RBM deep belief network on the MNIST database. This model contains 6 000 neurons and five populations.

⑥Each router has its own table; thus the scratchpad memory is private.

⑦http://www.hpl.hp.com/research/cacti/, Nov. 2015.

2) *Basal Ganglia*[29] (*BG*). It models the basal ganglia, a group of interconnected subcortical nuclei, associated with a variety of functions including control of voluntary motor movements, procedural learning, etc. This model implements the topology and real-time work-situation of the basal ganglia, which contains about 1 200 neurons of five populations.

3) *Controlled Question Answering Network* (*QAWC*). This demo performs question answering based on storing items in a working memory, and under control of a basal ganglia. It contains about 12 000 neurons and 13 populations.

4) *Question Answering Network* (*Question*). This model simulates the question-answering function, which provides the answer by learning examples. This model includes about 8 000 neurons and 80 populations.

5) *Temporal Differentiation* (*Diff*)[30]. This model performs the computation of temporal differentiation, which contains 5 000 neurons and three populations.

6) *Spatiotemporal Processing and Coincidence Detection* (*Spat*). This demo aims at simulating connections between the retina and the cochlea, and realizes a co-incidence detector. It has 8 500 neurons and 18 populations.

7) *Neural Path Integrator*[31] (*Path*). This model incorporates representations and updating of position into a single layer of neurons without using multiplicative synapses. This model has 1 600 neurons and 12 populations.

8) *Bandit Task*[32]. This is a set of four bandit task models⑧ to exhibit how a simulated rat responds to several different environments. It contains four networks with similar topologies. Each includes more than 1 000 neurons and 15∼20 populations.

We have simulated each network for 10 times with random (but legal) inputs, and recorded the spike-information of each neuron. Without loss of generality, the active degrees of neurons of some models with different scales have been shown in Fig.4 (others own the similar feature): for clarity, 100 consecutive neurons from each model are randomly selected to display. The X-axis is neuron IDs; the Y-axis is the testing sequences; the Z-axis represents the active degree of each neuron, namely, the ratio of the number of its spike-issues to the maximum number of all neurons. From Fig.4, we can find that the distributions of neurons with diverse active degrees show similarities to a great extent, under randomly-generated inputs.

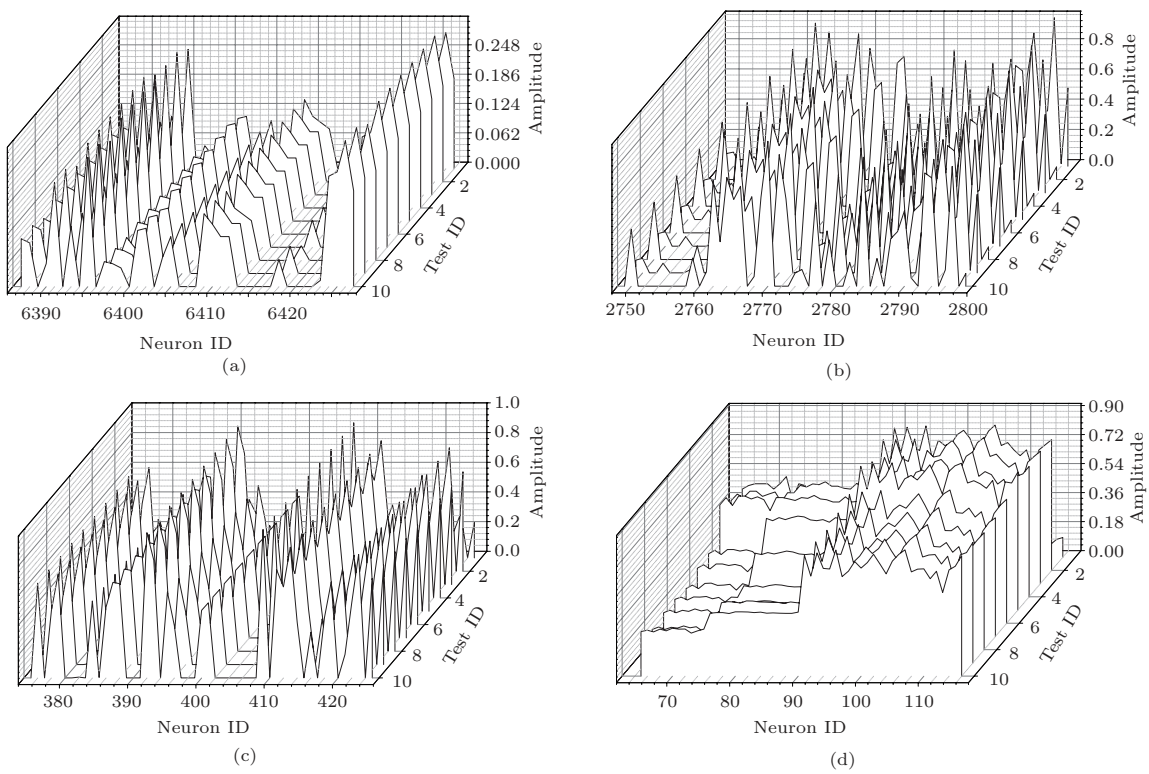

Fig.4. Active degrees of neurons. (a) QAWC. (b) Diff. (c) Bandit task (quarterlearn). (d) Digit.

⑧Four models are abbreviated as *arm*, *env*, *halflearn* and *quarterlearn* respectively.

We believe it is a meaningful finding. As mentioned previously, we try to carry out research from the perspective of computer architecture. This invariance helps handle the on-chip SNN simulation as parallel tasks in which each task (population) has different active degrees in terms of communication. Thus, some typical architectural topics, including parallel task distribution, and load balance, may be emerging.

After the linear mapping (as described in Subsection 3.2), we can drive the simulator with traces.

First, we get the energy-consumption information of above-mentioned SNNs (the default simulation is set to 100; namely, one simulated SNN cycle contains 10 000 NoC cycles). Three cases are tested: the number of neurons that one processing node can occupy is set to 64, 128 and 256 respectively. Results are presented in Fig.5, which gives the average energy-consumption of SNNs in one simulated second. Apparently, the energy-consumptions of the NoC decrease with the increase of the number of neurons that one node can simulate, because the NoC scale has been reduced, as well as the spike communications.

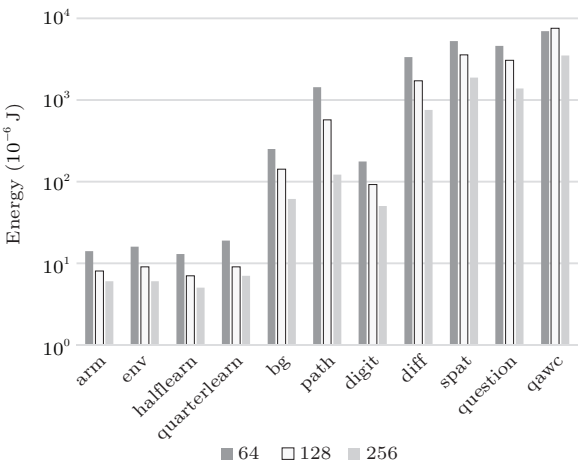

Fig.5. Energy consumptions of NoC (different legends show the different numbers of neurons that one processing node can occupy).

Correspondingly, for most of the SNN applications, when the number has increased from 64 to 128, the average transmission-latencies decrease (in Fig.6). While the number reaches 256, the latencies of some SNNs (like arm, env, quarterlearn, spat) increase. The reason lies in that some local congestion becomes severer. Although the hop count of transmission keeps decreasing, each spike spends more time at every node. This reason does also hold for those SNNs (including question and qawc) whose latencies have grown as the number increases from 64 to 128.

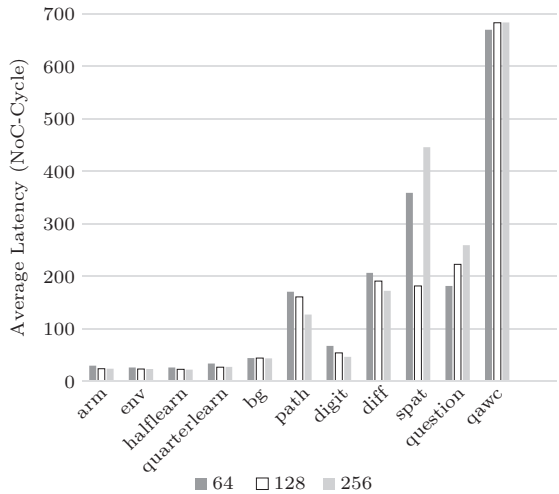

Fig.6. Average transmission-latencies (different legends show different numbers of neurons that one processing node can occupy).

5 Conclusions

This paper presented a methodology to bridge the gap between the applications from the neuroscientists' community and neuromorphic hardware, focusing on simulation technologies. Some key methods, like SNN modes' extraction (from the functional simulator) and evaluation (on the architecture-level simulator) were proposed.

On the other hand, it is just a beginning. The aspect of computer architecture will bring forward more interesting topics.

Besides the exploration of design space of micro-architecture, the strategies of mapping SNNs to NoC may be pivotal to fully utilize the hardware resources. Moreover, hardware implementation is confronted with more constraints (for example, a neuron's connectivity will be limited), which may further put constraints on applications. The interaction is worth studying.

Anyway, we intend to use it as the joint to combine research from different fields.

References

[1] Paugam-Moisy H, Bohte S. Computing with spiking neuron networks. In *Handbook of Natural Computing*, Rozenberg G, Bäck T, Kok J N (eds.), Springer-Verlag Berlin Heidelberg, 2012, pp.335-376.

[2] Hereld M, Stevens R, Sterling T, Gao G R. Structured hints: Extracting and abstracting domain expertise. Technical Report, ANL/MCS-TM-303, Argonne National Lab-

oratory, 2009. http://www.mcs.anl.gov/papers/ANL-MCS-TM-303.pdf, Nov. 2015.

[3] 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*, 2014, 345(6197): 668-673.

[4] Furber S B, Lester D R, Plana L A *et al.* Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers,* 2013, 62(12): 2454-2467.

[5] Boahen K. Neurogrid: Emulating a million neurons in the cortex. In *Proc. the 28th IEEE EMBS Annual International Conference,* Aug.30-Sept.3, 2006, p.6702.

[6] Schemmel J, Bruderle D, Grubl A *et al.* A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proc. the 2010 IEEE International Symposium on Circuits and Systems,* May 30-June 2, 2010, pp.1947-1950.

[7] Liu D, Chen T, Liu S *et al.* PuDianNao: A polyvalent machine learning accelerator. In *Proc. the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2015, pp.369-381.

[8] Chen Y, Luo T, Liu S *et al.* DaDianNao: A machine-learning supercomputer. In *Proc. the 47th IEEE/ACM Int. Symp. Microarchitecture* (*MICRO*), Dec. 2014, pp.609-622.

[9] Chen T, Du Z, Sun N *et al.* DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGPLAN Notices,* 2014, 49(4): 269-284.

[10] Chakradhar S, Sankaradas M, Jakkula V *et al.* A dynamically configurable coprocessor for convolutional neural networks. *ACM SIGARCH Computer Architecture News,* 2010, 38(3): 247-257.

[11] Fidjeland A K, Roesch E B, Shanahan M P *et al.* NeMo: A platform for neural modelling of spiking neurons using GPUs. In *Proc. the 20th IEEE ASAP,* July 2009, pp.137-144.

[12] Glackin B, McGinnity T M, Maguire L P *et al.* A novel approach for the implementation of large scale spiking neural networks on FPGA hardware. In *Proc. the 18th IWANN,* June 2005, pp.552-563.

[13] Wendt K, Ehrlich M, Schüffny R. A graph theoretical approach for a multistep mapping software for the facets project. In *Proc. the 2nd WSEAS International Conference on Computer Engineering and Applications,* Jan. 2008, pp.189-194.

[14] Seo J, Brezzo B, Liu Y *et al.* A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Proc. IEEE CICC,* Sept. 2011.

[15] Amir A, Datta P, Risk W P *et al.* Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores. In *Proc. the 2013 International Joint Conference on Neural Networks* (*IJCNN*), Aug. 2013.

[16] Carrillo S, Harkin J, McDaid L J *et al.* Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations. *IEEE Transactions on Parallel and Distributed Systems,* 2013, 24(12): 2451-2461.

[17] Pande S. Design exploration of EMBRACE hardware spiking neural network architecture and applications [Ph.D. Thesis]. Electrical & Electronic Engineering College of Engineering and Informatics, National University of Ireland, Feb. 2014.

[18] Happel B L M, Murre J M J. Design and evolution of modular neural network architectures. *Neural Networks*, 1994, 7(6/7): 985-1004.

[19] Ali M, Welzl M, Adnan A *et al.* Using the NS-2 network simulator for evaluating network on chips (NoC). In *Proc. IEEE ICET*, Nov. 2006, pp.506-512.

[20] Lis M, Shim K S, Cho M H *et al.* DARSIM: A parallel cycle-level NoC simulator. http://wwweb.eecs.umich.edu/MoBS/2010/proceedings/2-mobs6-lis.pdf, Nov. 2015.

[21] Wang H S, Zhu X, Peh L S *et al.* Orion: A power-performance simulator for interconnection networks. In *Proc. the 35th Annual IEEE/ACM International Symposium on Microarchitecture,* Nov. 2002, pp.294-305.

[22] Kahng A B, Li B, Peh L S *et al.* ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proc. Design, Automation and Test in Europe*, April 2009, pp.423-428.

[23] Vainbrand D, Ginosar R. Scalable network-on-chip architecture for configurable neural networks. *Microprocessors and Microsystems*, 2011, 35(2): 152-166.

[24] Jerger N E, Peh L S, Lipasti M. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *Proc. the 35th International Symposium on Computer Architecture,* June 2008, pp.229-240.

[25] Abad P, Puente V, Gregorio J. MRR: Enabling fully adaptive multicast routing for CMP interconnection networks. In *Proc. the 15th IEEE HPCA*, Feb. 2009, pp.355-366.

[26] Rodrigo S, Flich J, Duato J *et al.* Efficient unicast and multicast support for CMPs. In *Proc. the 41st IEEE/ACM MICRO,* Nov. 2008, pp.364-375.

[27] Eliasmith C, Anderson C H. Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. MIT Press, 2004.

[28] Hines M L, Morse T, Migliore M *et al.* ModelDB: A database to support computational neuroscience. *Journal of Computational Neuroscience,* 2004, 17(1): 7-11.

[29] Redgrave P, Prescott T J, Gurney K. The basal ganglia: A vertebrate solution to the selection problem? *Neuroscience*, 1999, 89(4): 1009-1023.

[30] Tripp B P, Eliasmith C. Population models of temporal differentiation. *Neural Computation*, 2010, 22(3): 621-659.

[31] Conklin J, Eliasmith C. A controlled attractor network model of path integration in the rat. *Journal of Computational Neuroscience*, 2005, 18(2): 183-203.

[32] Stewart T C, Bekolay T, Eliasmith C. Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Neuroscience*, 2012, 6(2).

Yu Ji received his B.S. degree in physics from Tsinghua University, Beijing, in 2011. Now he is a Ph.D. student in the Department of Computer Science and Technology at Tsinghua University, Beijing.

You-Hui Zhang received his B.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, in 1998 and 2002 respectively. He is currently a professor in the Department of Computer Science and Technology at Tsinghua University, Beijing. His research interests include computer architecture, cloud computing, and high-performance computing. He is a member of CCF, ACM and IEEE.

Wei-Min Zheng received his B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, in 1970 and 1982 respectively. Now he is a professor in the Department of Computer Science and Technology at Tsinghua University, Beijing. His research interests include high-performance computing, network storage and distributed computing. He is a fellow of CCF and a member of ACM and IEEE.