A Large-Scale Spiking Neural Network Accelerator for FPGA Systems

Kit Cheung[1], Simon R. Schultz[2], and Wayne Luk[1]

[1] Department of Computing,
[2] Department of Bioengineering,
Imperial College London
{k.cheung11,s.schultz,w.luk}@imperial.ac.uk

Abstract. Spiking neural networks (SNN) aim to mimic membrane potential dynamics of biological neurons. They have been used widely in neuromorphic applications and neuroscience modeling studies. We design a parallel SNN accelerator for producing large-scale cortical simulation targeting an off-the-shelf Field-Programmable Gate Array (FPGA)-based system. The accelerator parallelizes synaptic processing with run time proportional to the firing rate of the network. Using only one FPGA, this accelerator is estimated to support simulation of 64K neurons 2.5 times real-time, and achieves a spike delivery rate which is at least 1.4 times faster than a recent GPU accelerator with a benchmark toroidal network.

Keywords: Spiking neural networks, simulation, FPGA.

1 Introduction

Despite the vast amount of anatomical and functional knowledge of the brain, the complete picture of how higher cognitive function emerges from neuronal and synaptic dynamics still eludes us. Large-scale simulation is useful in this regard, since we can investigate how such functions emerge from deterministic simulation.

There are previous attempts to simulate large number of neurons. Izhikevich investigates the mammalian thalamo-cortical system (10^6 neurons with 5×10^8 synapses) [1], Blue Brain employs highly biologically precise models (10^6 neurons) [2]. IBM focuses on the computing power and mainframe infrastructure (1.6×10^9 neurons with 8.87×10^{12} synapses) [3]. However, the availability, cost and energy consumption of such supercomputers can be a concern. The interest in customized platforms for neural network simulation has therefore been growing, resulting in platforms such as SpiNNaker targeting a network of microprocessor chips [4], NeMo targeting GPU [5], and FACETS targeting custom VLSI chip and silicon wafers [6].

This work, building on an earlier proof of principle [7], involves designing a large-scale SNN accelerator employing Izhikevich spiking neuron model. One contribution of this work is a module capable of efficient parallel weight distribution. We utilize on-chip memory to store frequently accessed variables in this module, such that most of the memory bandwidth is used to access neuronal parameters and synaptic data.

A.E.P. Villa et al. (Eds.): ICANN 2012, Part I, LNCS 7552, pp. 113–120, 2012.

A major challenge of devising an efficient SNN accelerator concerns the memory storage and access patterns. When simulating a cortical neural network, the main memory holds primarily the synaptic data since one neuron typically receives 1000 to 10000 synapses from other neurons. As a result, SNN calculation is a memory-intensive task and the efficiency of an SNN accelerator heavily depends on the time spent on accessing neuronal parameters and synaptic weight data. We design a memory storage and access scheme for synaptic data using a simple yet effective method.

This accelerator also makes use of the fact that only a small percentage of neurons are firing (sparse firing) at any given time. Our event-driven hardware architecture accesses synaptic weight only when its presynaptic neuron fires, thus reduce the required running time for weight distribution to be linear to the firing rate.

A prototype node with a single FPGA is implemented to investigate the proposed design, where the FPGA is configured as a stream processor. This accelerator is designed such that multiple FPGAs can be easily concatenated to produce a parallel FPGA network efficiently, allowing a large network to be simulated in real time.

In short, the novelty and merits demonstrated in this work are:

- A module capable of efficiently distributing synaptic weights in parallel
- A simple and efficient memory storage and access scheme
- An event-driven and fully pipelined approach for simulating SNN

2 Background

2.1 Previous Work

The history of employing electronics to implement spiking neurons dates back to 1989 when Carver Mead used analog VLSI to implement neuromorphic devices [9]. Since then FPGA technology has been used to implement neuron-parallel networks. Although FPGA-based soft processors produce high throughput and is biologically plausible [10], the network size is constrained by the hardware resources available and hence they are not suitable for large-scale simulation.

A few projects have demonstrated the feasibility of implementing large-scale SNN accelerator on GPUs. Our work is comparable to a GPU accelerator [5] since the systems have similar functionality and target network size. Such accelerators optimize for local connectivity of neurons and show good performance over other GPU and CPU implementations.

We suggest that FPGA technology is a better choice for implementing SNN accelerator than microprocessors and GPUs. SpiNNaker, A microprocessor-based architecture, adopts a distributed approach such that a single microprocessor is responsible for processing a number of neurons. Their system implements a ‘virtual network’ which can be customized. The synaptic data are distributed such that a large amount of data has to be sent across the microprocessor network which complicates communication.

A centralized approach is used in a number of GPU [4] and FPGA [11] implementations where data are stored in a centralized storage of main memory. Although GPUs have a high memory bandwidth to off-chip memory which is an advantage in memory-intensive SNN calculation, their on-chip memory to store frequently accessed

data is small in size. Contemporary FPGAs have several megabytes of on-chip memory, which we have utilized to store accumulated postsynaptic current. The main memory can then be used mainly for accessing neuronal parameters and synaptic weights.

2.2 General Procedure of SNN Simulation

SNN is a type of neural network widely adopted in studying neural computation using spikes, an abstracted form of action potential, as the means of communication between neurons. This work adopts the Izhikevich model [8] for modeling neurons which is computationally efficient, while sufficiently precise to produce various dynamics. In each time step, the membrane potential variable v and recovery variable u are changed according to equation (1a), (1b) and (1c).

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I + sN \tag{1a}$$

$$\dot{u} = a(bv - u) \tag{1b}$$

$$if\,(v > 30)\ \text{then}\ v = c\,,\text{else}\ u = u + d \tag{1c}$$

In the above equations, a, b, c and d are the parameters describing the dynamics of the neuron while v and u are the membrane potential and membrane recovery variable of the neuron respectively. I is the incoming postsynaptic current into the neuron. We used a delta function in the current implementation (i.e. adding the synaptic weight directly to v). N is a normally distributed random number to mimic random fluctuation in synapse, and s represents the magnitude of the fluctuation.

To simulate a network of Izhikevich neurons:

Step 1. For each time step all the neurons are updated according to equation above;
Step 2. If a neuron fires, the list of synapse data (i.e. postsynaptic neuron index, synaptic weight, axonal delay) of this neuron are accessed from memory
Step 3. Synaptic current cache I accumulates synaptic weight to a slot according to the postsynaptic neuron index

We design a module capable of parallelizing the access, distribution and accumulation of synaptic weight described in step 2 and 3.

3 Design

Fig. 1 shows the overall architecture of the design. The two main modules, neuron module and weight distribution module, correspond to neuron state update (step 1) and weight accumulation (step 2 and 3) respectively, and are connected together by a FIFO storing indices of fired neurons and a controller. The design is fully pipelined hence processing time is reduced. This design has a number of novel features. First, our weight distribution module parallelizes access, distribution and accumulation of synaptic values concurrently. Second, synaptic data are arranged in consecutive RAM

locations, thus a single access returns data for a number of synapses. Third, a simple and efficient event-driven architecture is implemented using FIFO queues.

The neuron module is a pipelined implementation of equation (1). It takes the postsynaptic current *I* accumulated from the previous time step as input and outputs indices of the fired neurons. The equation parameters are streamed into the module every cycle. The neuron model used in neuron module can easily be changed to other neuron models (e.g. integrate-and-fire). Several neuron modules can be employed to parallelize neuron state update should the updating becomes bottleneck of the design.

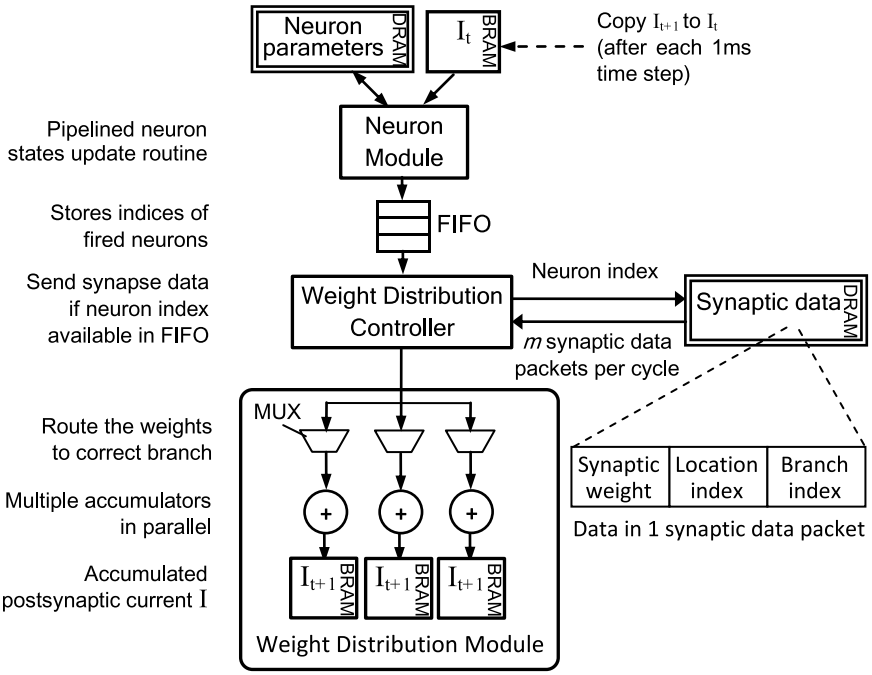

Fig. 1. Overall architecture of accelerator showing the connection between the two main modules (neuron module and weight distribution module), the weight distribution controller and off-chip DRAM (double-bordered)

The weight distribution controller is the intermediate component between the two main modules in the design. It translates indices of fired neurons queued in the input FIFO into memory addresses and data size for accessing the required synaptic data. The accessed synapses have the fired neuron as their common presynaptic neuron (i.e. a fan-out design). The controller calls the weight distribution module on demand for each fired neuron index in the FIFO, such that the processing time which is dominated by weight distribution would be proportional to the firing rate of the network.

The weight distribution module processes the synaptic weights in parallel. The module consists of a number of branches, each of which corresponds to one synapse and most of them are actively accumulating synaptic weights when the synaptic data packets arrive. The number of branches in the module is equal to the number of

synaptic data packets received from DRAM in one cycle. Each branch consists of a multiplexor, an adder and one or more blocks of on-chip Block RAM (BRAM) to accumulate the postsynaptic current *I*.

The accumulated current *I* is stored in BRAM for the weight distribution module to access *I* efficiently. The neurons in the network are assigned to a location in one of the BRAMs in the branches. In GPU-based accelerators, it takes a significant amount of bandwidth to send data back-and-forth between the accelerator and the main memory to access *I*, thus FPGAs have an advantage in reducing memory bandwidth. Using the current design, the network size is constrained by the available BRAM size on the FPGA, but the architecture can be further developed such that *I* is stored in main memory and only a fraction of *I* is stored in BRAM at any point in time.

It is easy to scale up the current design to enable a large network to be simulated in parallel on a number of FPGAs. A multi-node implementation can be achieved by connecting the FIFOs into a systolic array using communication bus between FPGAs where indices of fired neurons are transmitted across FPGAs at a fast rate.

4 Implementation

This work targets an off-the-shelf computing node provided by Maxeler, with 4 Virtex6 SX475T FPGA (40nm process) as the main computing element. The platform has 96GB off-chip DDR3 DRAM with 38.4 GB/s memory bandwidth, and synchronous on-chip BRAM of 4.6MB in each FPGA. For the current implementation only one FPGA is used, but the work can be extended to use all the 4 FPGAs. The platform is configured as a stream processor thus allows us to implement a fully pipelined design. The vendor also provides high-level design tools which translate Java description of hardware resource and functions into low level VHDL, thus simplifying the implementation work. The platform has designated data paths connecting all the FPGAs in a ring to provide high-speed inter-FPGA connections, thus suits our need to implement a multi-FPGA network previously described.

4.1 Weight Distribution Module

Fig. 2 illustrates the architecture for the weight distribution module. It has three incoming synapse data per cycle and the same number of branches. The module receives the synaptic weight and postsynaptic destination for a number of cycles, and distributes the data to the BRAMs through the MUXs as shown in Fig. 2. The synaptic data are organized in memory such that no two synaptic data will be routed to the same branch. A read is first performed in the BRAM, and is added to the new incoming weight and written back to the same address.

In the current design, approximately 1M slots can be accommodated in the BRAM, thus effectively allowing a network with 500K neurons to be simulated on one FPGA (memory blocks need to store both $\underline{I_t}$ and I_{t+1}). To incorporate propagation delay which is currently not implemented, the maximum size of the network will be equal to 500K divided by the maximum delay in the network. Thus I can be considered to swap to DRAM if the network size of maximum delay is large, which costs extra resource usage to build the memory interface.

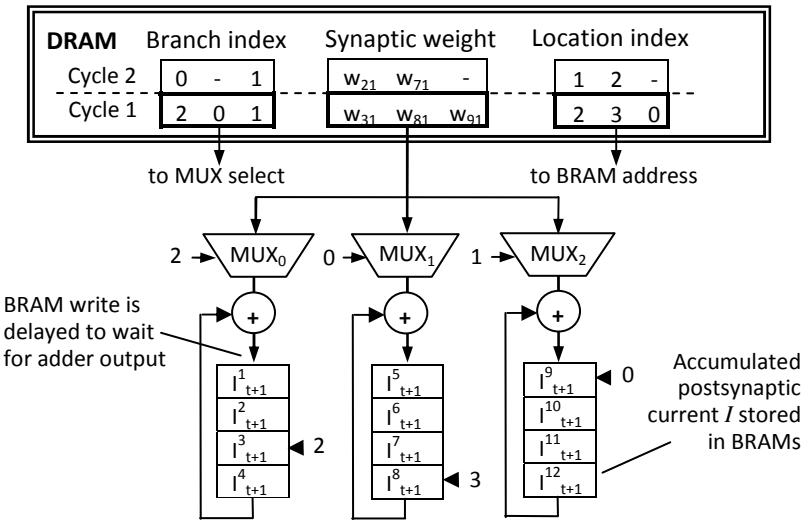

Fig. 2. Weight distribution module with 3 input synapse data per cycle. In this example neuron 1 connects to neurons 2, 3, 7, 8 and 9 in the network, and neuron 1 is fired in the last time step. The graph shows the configuration to accumulate w_{31}, w_{81} and w_{91} in the first cycle.

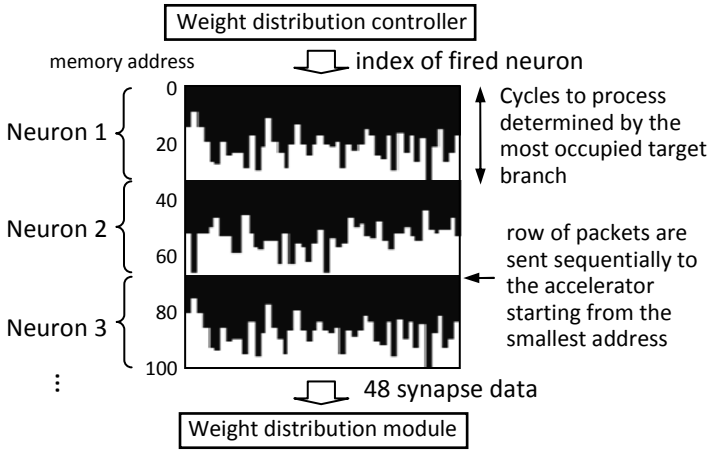

Fig. 3. Memory layout for synaptic data. Black areas represent occupied memory space storing synaptic data packets. In the x-axis of the graph, the packets are arranged according to its target branch index instead of its actual location for easy understanding.

4.2 Memory Layout of Synaptic Data

The synaptic memory space consists of synaptic data packets stored in consecutive addresses in the DRAM (Fig. 3). Each packet has a total of 32 bits, where synaptic weight occupies 16 bits, branch index uses 6 bits (up to 64 branches) and location index uses 10 bits (up to 1024 BRAM locations). A map is used to translate the neuron index to the I location when the synaptic data are loaded into DRAM. Neurons are randomly assigned a location in the BRAMs in the current implementation.

The synaptic data are arranged in DRAM such that they occupy consecutive memory space, with a constraint that no two neurons target the same branch in a single fetch. This process is done on the fly when the data streams into the memory during initialization. The proportion of the empty area in Fig. 3 causes overhead when accumulating synaptic weights. So one possible optimization is to shuffle the position of neurons in the BRAMs such that neurons in branches that are heavily accessed are swap to branches with less workload, and thus effectively reduce the overhead.

5 Results

We build a prototype of the accelerator at 100MHz to estimate the resource utilization (Table 1) and latency. Over half of the resources are used to support memory interface and other peripheries. The accelerator itself uses only a small amount of resources.

To evaluate the performance, we use a toroidal network [5] with various sizes (p) and various degrees of sparseness of synaptic connectivity (σ). The network has $1024 \times p$ neurons, each neuron making 1000 synaptic connections to its neighbouring neurons. The network fires at approximately 7Hz under all conditions.

Table 1. Resource utilization of accelerator prototype with 48 branches

Neuron Modules	LUTs	FFs	BRAMs	DSPs
1	73593 (24.7%)	81896 (13.8%)	252 (23.7%)	10 (0.5%)
2	80212 (27.0%)	90669 (15.2%)	268 (25.1%)	20 (1%)

We estimate the spike delivery rate, suggested as a measure of performance [5] (Fig. 4, left), using a 48-branch and 2-neuron-module design using one FPGA. The measure is used since the firing rate and synapses vary for different networks. Unlike related work [5], the sparseness of synaptic connection does not affect the performance of our architecture. The accelerator achieves throughput of 1.39G/1.17G spikes/s for spike delivery rate, enabling 6 times/2.5 times speedup with respect to real time for 32K/64K neuron simulation. The performance is bound by memory bandwidth, thus there will not be a large performance gain if higher clock rate is used.

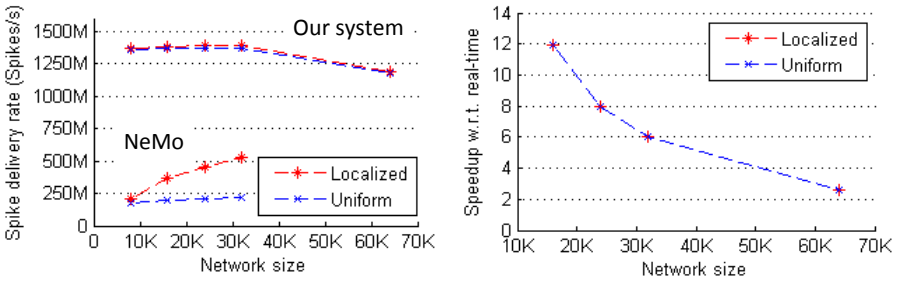

Fig. 4. Estimation of spike delivery rate (left) and speedup with respect to real-time (right) vs network size, using 48 branches and 2 neuron modules under localized (σ = 32) and uniform synaptic connectivity conditions

Our accelerator is 1.4 times (localized connectivity) to 5.5 times (uniform connectivity) faster than the GPU NeMo accelerator (Tesla C1060 65nm process) in terms of spike delivery rate. However, our current accelerator does not support axonal delay and synaptic plasticity which require additional hardware resources.

6 Conclusion

This paper describes a spiking neural network accelerator capable of supporting large-scale simulation using an FPGA-based system. We propose a scheme to efficiently parallelize the access and processing of synaptic weights in SNN. The accelerator shows high spike delivery throughput and outperforms a GPU accelerator.

To achieve a more biologically realistic simulation, the current work needs to be extended to support axonal delay, spike-timing dependent plasticity and postsynaptic potential kernel. Future improvements of the current design include optimizing the memory to reduce overhead, increasing the parallelism of the design by using more hardware resources, and enhancing the efficiency of the memory bandwidth.

Acknowledgments. The research leading to these results has received funding from European Union Seventh Framework Programme under grant agreement number 287804, 248976 and 257906. The support by the Croucher Foundation, UK EPSRC, HiPEAC NoE, Maxeler University Program, and Xilinx is gratefully acknowledged.

References

1. Izhikevich, E.M., Edelman, G.M.: Large-scale model of mammalian thalamocortical systems. PNAS 105, 3593–3598 (2008)
2. Markram, H.: The Blue Brain Project. Nat. Rev. Neurosci. 7, 153–160 (2006)
3. Ananthanarayanan, R., Esser, S.K., Simon, H.D., Modha, D.S.: The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. In: Proc. Conf. High Performance Computing Networking, Storage and Analysis, pp. 1–12. ACM (2009)
4. Khan, M.M., Lester, D.R., Plana, L.A., Rast, A., Jin, X., Painkras, E., Furber, S.B.: SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor. In: Proc. IEEE International Joint Conference on Neural Networks (2008)
5. Fidjeland, A.K., Shanahan, M.P.: Accelerated simulation of spiking neural networks using GPUs. In: Proc. IEEE International Joint Conference on Neural Networks (July 2010)
6. Schemmel, J., Bruderle, D., Grubl, A., Hock, M., Meier, K., Millner, S.: A wafer-scale neuromorphic hardware system for large-scale neural modeling. In: Proc. IEEE Int. Conf. Circuits and Systems, pp. 1947–1950 (2010)
7. Cheung, K., Schultz, S.R., Leong, P.H.W.: A parallel spiking neural network simulator. In: Proc. Int'l Conf. on Field-Programmable Technology (FPT), pp. 247–254 (2009)
8. Izhikevich, E.M.: Simple model of spiking neurons. IEEE Transactions on Neural Networks 14(6), 1569–1572 (2003)
9. Mead, C.: Analog VLSI and Neural Systems. Addison-Wesley (1989)
10. Maguire, L.P., McGinnity, T.M., Glackin, B., Ghani, A., Belatreche, A., Harkin, J.: Challenges for large-scale implementations of spiking neural networks on FPGAs. Neurocomputing 71(1-3), 13–29 (2007)
11. Moore, S.W., Fox, P.J., Marsh, S.J.T., Markettos, A.T., Mujumdar, A.: Bluehive – A Field-Programable Custom Computing Machine for Extreme-Scale Real-Time Neural Network Simulation. In: FCCM, pp. 133–140 (2012)