A LARGE-SCALE SPIKING NEURAL NETWORK ACCELERATOR FOR FPGA SYSTEMS

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

Department of Computing[1], Department of Bioengineering[2]
Imperial College London
email: {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 modelling 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 2.48 times real-time, and achieves a spike delivery rate which is at least 1.45 times faster than a recent GPU accelerator with a benchmark toroidal network.

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 a number of previous attempts to simulate large number of neurons from the scale of 10^6 neurons to 10^9 neurons in various level of biological details [1][2]. 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 based on microprocessor chips [3], GPUs [4], and silicon wafers [5].

This work, building on an earlier proof of principle [6], involves designing a large-scale SNN accelerator targeting FPGA platforms. 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 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.

A prototype node with a single FPGA is implemented to investigate the proposed design with axonal delay support. In short, the novelty and merits demonstrated in this work are:

- Accumulation of synaptic weights in parallel
- Simple and efficient memory data storage and access
- Event-driven and fully pipelined architecture

2. BACKGROUND

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 [7] for modelling 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 (2.1), (2.2) and (2.3).

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I + sN \tag{2.1}$$

$$\dot{u} = a(bv - u) \tag{2.2}$$

$$if\ (v > 30)\ then\ v = c,\ else\ u = u + d \tag{2.3}$$

In the above equations, a, b, c and d are the parameters describing the dynamics of the neuron. I is the incoming postsynaptic current into the neuron due to the spiking from other neurons. 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, the platform iterates in time steps, each step requires calculation of neuron states according to equation above and accumulation

of synapse weight when a neuron fires. The weights are then accumulated in a on-chip memory slot according to the postsynaptic neuron index of the synaptic weight.

FPGA and GPUs are two popular platforms for implementation of SNN accelerators due to their inherent parallel computation architecture. We suggest that FPGA technology is a better choice for 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. The distributed synaptic data means a large amount of data has to be sent across the microprocessor network which complicates the architecture and communication. In contrast, a centralized approach is used in a number of GPU [4] and FPGA [8] implementations where data are stored in a centralized storage of main memory, reducing the communication required between processors.

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. We design a module capable of parallelizing the access, distribution and accumulation of synaptic weight.

3. DESIGN AND IMPLEMENTATION

3.1. Processing in a Simulation Step

Fig.1 shows the overall architecture of the design. The two main modules, neuron module and weight distribution module, correspond to neuron state update and weight accumulation respectively. They are connected together by the weight distribution controller which manages synaptic weight access and scheduling of address streaming. The design is fully pipelined hence processing time is reduced.

The neuron module is a pipelined implementation of the equations in the Izikevich model. 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 from external memory, whereas fired neuron addresses stream out from the module and are buffered in on-chip memory. Several neuron modules can be employed to parallelize neuron state update should the updating becomes bottleneck of the design.

Using a look-up table, the weight distribution controller then translates indices of fired neurons buffered in on-chip memory into memory addresses and data size required one at a time. The controller retrieves synaptic data whenever the fired neuron index buffer is not empty, and is thus event-

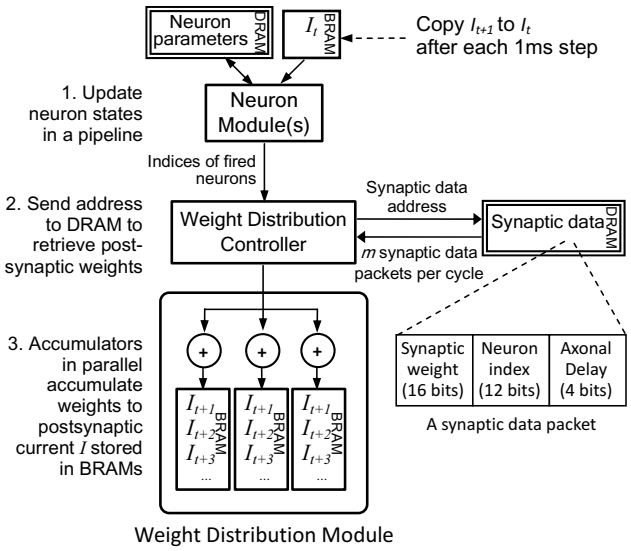

Fig. 1. Overall architecture of accelerator showing the connection between the two main modules (neuron module and weight distribution module), the weight distribution controller and off-chip DRAM (double-bordered).

driven with running time proportional to the firing rate. The synaptic weights retrieved correspond to synapses outgoing from the fired neurons, and for each fired neuron it takes a number of sequential access to the external memory to fetch all the synaptic packets. The controller then continues to the next fired neuron when the data retrieval ends.

In the next stage, the weight distribution module then accumulates the synaptic weights in parallel. The module consists of a number of branches, each of which corresponds to a synapse. The number of branches in the module is equal to the number of synaptic data packets received from DRAM in one cycle. Fig.2 shows the process of weight accumulation with synapse axonal delay. Axonal delay is implemented by using a looping counter which keeps track of the slot for the current time. The counter counts to the maximum allowed delay and then counts back to zero, and effectively utilizing the BRAMs as circular buffers. The delay value is added to the current time to obtain the slot the incoming synaptic weight should store in. The previous value is read from the BRAM, added to the new incoming weight using an adder, and is stored back to the same address. The delay of the input to the write port is to account for the latency of the read and add operations. At the end of a 1ms iteration, the contents of BRAMs storing I_{t+1} is transferred to I_t at the start of the pipeline.

Using the current design, the network size is constrained by the available BRAM size on the FPGA. For a maximum axonal delay 16 ms, the platform is able to simulate a network of size of approximately 100K neurons. For larger network to be simulated, we can employ multiple FPGAs and

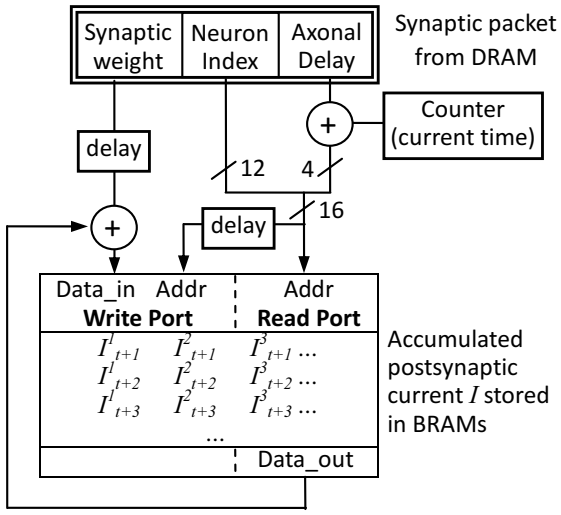

Fig. 2. Architecture of a synaptic branch supporting axonal delay.

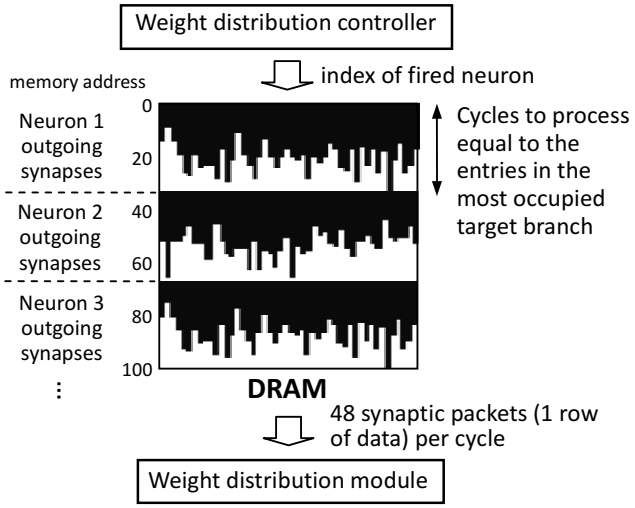

Fig. 3. Memory layout for synaptic data. Regions in black represent occupied memory space storing synaptic data packets. The packets are arranged according to its target branch index in each row.

connect the on-chip memory holding fired neuron indices to the weight distribution controller in the neighbouring FPGA nodes in the form of systolic array. This design requires direct communication bus between FPGAs to ensure speed of communication does not become the bottleneck of the design. Alternatively, it is possible to increase the network size by swapping the buffer holding I to the external memory at the end of each cycle, at the cost of additional communication latency between external memory and the FPGA.

3.2. Memory Layout of Synaptic Data

Synaptic data packets are arranged in consecutive RAM locations with the packet location specially arranged in prior of simulation (Fig.3). Each packet has a total of 32 bits: the synaptic weight occupies 16 bits, the postsynaptic neuron index uses 12 bits and the axonal delay value which is 16 ms at maximum uses the remaining 4 bits. For a row of packets to be sent to the FPGA in each cycle from the DRAM, the packets are arranged according to its destination branch in prior during the accelerator configuration stage, with packets having smaller postsynaptic neuron index on the top. The synaptic data packets is then transferred to the correct branch with no extra routing required when its presynaptic neuron fires. This method contributes to a considerable saving in the packet data size and resources.

The proportion of the empty area in Fig.3 causes overhead for synaptic weight accumulation since the empty packets are not used. The packets assignment to the branches is almost random and the efficiency of the transfer scheme is estimated to be 65% for a 48-branch scheme and 76% for a 24-branch scheme. The higher efficiency property of lower-branch designs would be utilized to boost the performance in future work.

3.3. Platform

This work targets an off-the-shelf computing node provided by Maxeler, with 4 Virtex6 SX475T FPGA (40 nm process) as the main computing element. The platform has 96 GB external DDR3 DRAM with 38.4 GB/s shared 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.

4. RESULTS

We build a prototype of the accelerator at 100MHz to estimate the resource utilization (Table 1) and latency. Since the task is primarily a memory-intensive task and is bound by memory bandwidth, there will not be a large performance gain if higher clock rate is used once the maximum memory bandwidth is reached.

To evaluate the performance, we use a toroidal network [4] with various sizes (p) and various degrees of synaptic

Table 1. Resource utilization of prototype with 48 branches simulating 64K-neuron network

Neuron Modules	1	2
LUTs	199421 (67.0%)	205985 (69.2%)
FFs	135032 (22.6%)	143805 (24.1%)
BRAMs	886 (83.3%)	901 (84.7%)

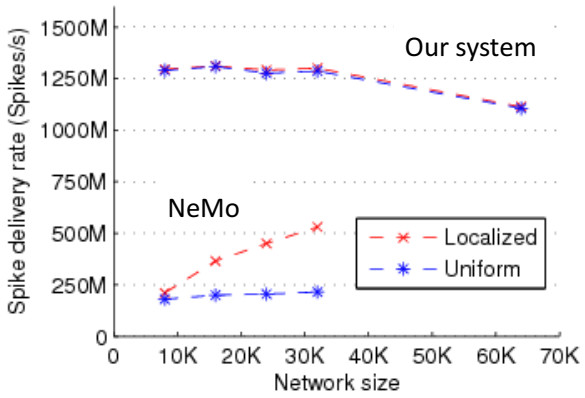

Fig. 4. Estimation of spike delivery rate vs. network size, using 48 branches and 2 neuron modules under localized (σ = 32) and uniform synaptic connectivity conditions.

connectivity sparseness (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. The propagation delay is proportional to the distance between the neurons in the grid, and the maximum propagation delay is 16 ms in this example.

We estimate the spike delivery rate (Fig.4), suggested as a measure of performance [4], 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 and this measure is relatively similar across various conditions. Unlike related work [4], the sparseness of synaptic connection does not affect the performance of our architecture. The accelerator achieves throughput of 1.30G/1.11G spikes/s for spike delivery rate, enabling 5.79 times/2.48 times speedup with respect to real time for 32K/64K neuron simulation. The drop of performance for the 64K network is due to bottleneck at the neuron modules, which can be solved by employing more neuron modules running in parallel.

Our accelerator is 1.45 times (localized connectivity) to 5.27 times (uniform connectivity) faster than the GPU NeMo accelerator (Tesla C1060 65 nm process) in terms of spike delivery rate. Comparing to NeMo however, our current accelerator does not support synaptic plasticity at the moment which requires additional hardware resources.

5. 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 accumulation 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 spike-timing dependent plasticity. 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.

6. REFERENCES

[1] H. Markram, "The blue brain project," *Nature Review Neuroscience*, vol. 7, pp. 153–160, 2006.

[2] R. Ananthanarayanan, S. Esser, H. Simon, and D. Modha, "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*, 2009, pp. 1–12.

[3] M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E. Painkras, and S. Furber, "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *Proc. IEEE International Joint Conference on Neural Networks*, 2008.

[4] A. Fidjeland and M. Shanahan, "Accelerated simulation of spiking neural networks using GPUs," in *Proc. IEEE International Joint Conference on Neural Networks*, July 2010.

[5] J. Schemmel, D. Bruderle, A. Grubl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Conf. Circuits and Systems*, 2010, pp. 1947–1950.

[6] K. Cheung, S. Schultz, and P. Leong, "A parallel spiking neural network simulator," in *Proc. Intl Conf. on Field-Programmable Technology (FPT'09)*, 2009, pp. 247–254.

[7] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[8] S. Moore, P. Fox, S. Marsh, A. Markettos, and A. Mujumdar, "Bluehive–a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *Proc. 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'12)*, 2012, pp. 133–140.