Four Simulators of the DANNA Neuromorphic Computing Architecture

Adam W. Disney
James S. Plank
Mark Dean

July, 2018

International Conference on Neuromorphic Systems (ICONS 2018)

https://ornlcda.github.io/icons2018/

Copyright ACM. Please find the ACM version of the final camera-ready paper at http://dl.acm.org/citation.cfm?id=3229884

The online home for this paper may be found at: http://neuromorphic.eecs.utk.edu

Citation – Plain Text:

author          A. W. Disney and J. S. Plank and M. Dean
title           Four Simulators of the {DANNA} Neuromorphic Computing Architecture
booktitle       International Conference on Neuromorphic Computing Systems
publisher       ACM
address         Knoxville, TN
month           July
year            2018
doi             10.1145/3229884.3229893
where           https://dl.acm.org/citation.cfm?id=3229893

Bibtex:

@INPROCEEDINGS{dpd:18:fsd,
    author = "A. W. Disney and J. S. Plank and M. Dean",
    title = "Four Simulators of the {DANNA} Neuromorphic Computing Architecture",
    booktitle = "International Conference on Neuromorphic Computing Systems",
    publisher = "ACM",
    address = "Knoxville, TN",
    month = "July",
    year = "2018",
    doi = "10.1145/3229884.3229893",
    where = "https://dl.acm.org/citation.cfm?id=3229893"
}
Four Simulators of the DANNA Neuromorphic Computing Architecture

Adam Disney  
University of Tennessee  
EECS  
adisney1@vols.utk.edu

James S. Plank  
University of Tennessee  
EECS  
jplank@utk.edu

Mark Dean  
University of Tennessee  
EECS  
markdean@utk.edu

ABSTRACT
DANNA is a computing architecture, designed in 2014 to meld features of recurrent, spiking, plastic neuromorphic computing systems with very efficient hardware implementations. Its hardware design and FPGA implementation preceded any software support or simulation. Therefore, the development of simulators for DANNA had multiple, and sometimes conflicting goals. In this paper, we describe four separate simulators for DANNA, including their motivations and details of their implementations. We present a performance evaluation, and conclude with lessons learned. We intend this work to be useful to researchers involved in the co-design of novel hardware and software systems.

KEYWORDS
Neuromorphic, DANNA, Spiking Neural Network

ACM Reference Format:

1 INTRODUCTION
The computing community is currently in an active phase of exploration centered around brain-inspired computing. Neuromorphic computing devices have proliferated, ranging from very large projects such as SpiNNaker[7] and TrueNorth[9] to smaller devices from smaller research groups. This paper focuses on one such device, the Dynamic Adaptive Neural Network Array (DANNA), developed in 2014 by Dean, Schuman and Birdwell [3]. DANNA’s design goal was to build a device similar to a FPGA, but for neuromorphic processing rather than conventional digital logic. It features a rectangular grid of elements that may be programmed as either neurons or synapses in a spiking neuromorphic system.

As with any hardware project, DANNA needs to be accompanied by a variety of software components, the most important of which is a simulator. This is necessary not only for hardware verification, but also for network training, application development and hardware design. Over the past three years, we have developed four separate simulators for DANNA. In this paper, we describe the simulators and how they achieve their various goals. We also assess their performance and their impact on the DANNA project.

2 DANNA
DANNA was designed to implement a neuromorphic computing model with the following features:

- A rectangular grid of elements that can be either neurons or synapses.
- Simple spiking neurons with programmable, digital integrate-and-fire thresholds.
- Synapses with programmable, digital delays to convey spikes with programmable, digital weights from neuron to neuron.
- 16 nearest neighbor connectivity.
- Limited plasticity in the form of long term potentiation (LTP) and depression (LTD).

A majority of the design decisions of DANNA were to leverage features of digital hardware. Shortly following the design of DANNA, it was implemented for Xilinx FPGA’s[1], and a VLSI design was completed [2]. A DANNA chip is currently under fabrication.

While the high-level view of DANNA is relatively simple, its implementation details have a significant effect on how DANNA performs computations. Sometimes these effects are counterintuitive, and undermine the simplicity of the high-level view. These details have been described elsewhere [1–3], but we highlight some of them here.

Clocks: DANNA is clocked, and there are multiple clock granularities. A global clock cycle corresponds to the unit of synaptic delay. For example, if a synapse is programmed to have a delay of 10, the units of that delay are global clock cycles. Inputs to DANNA and outputs from DANNA take the form of packets that are consumed and generated at the speed of the global clock. Within a global clock cycle, there are sixteen clock sub-cycles, each of which is dedicated to neurons reading potential spike information from each of their sixteen neighbors.

The order in which neurons read information can have an non-trivial impact on the operation of the array. For example, suppose that a neuron’s threshold is ten units of charge, and on a given global clock cycle, it receives charge events of 12 and -5 from neighboring synapses. If it reads the 12 before it reads the -5, then it will fire and reset its charge to a base threshold. Moreover, the first synapse will potentiate its weight, and the second will depress. However, if the neuron reads the -5 before it reads the 12, then it will not fire, but instead will hold charge that is three units lower than the threshold; the synapses will neither potentiate nor depress.
To eliminate a bias for either situation, the order in which neurons traverse their connections during a global clock cycle is randomized using a linear feedback shift register that is seeded to a known value when DANNA is booted.

**Thresholds:** DANNA neurons have an eight bit register to hold their charge. To eliminate the need for a comparator, the register is treated as an unsigned value with a fixed threshold of 128. In this way, only the top bit must be checked to determine whether the neuron should fire. To allow for programmable thresholds, the neuron is configured with a default charge level to which the neuron resets after firing. For example to configure a threshold of 2, the neuron’s default charge would be set to 126 so that the neuron needs 2 units of charge to fire. While this feature may be abstracted from the user by software, it affects the underlying operation of the neuron.

For example, suppose the user wants to program a neuron with a threshold of 120. Then the default charge of the neuron is eight. If a neighboring synapse fires any value into the neuron that is less than -8, it has the effect of zeroing the neuron’s charge. In essence, by setting the threshold to 120, the neuron’s smallest charge value becomes -8, whereas a threshold of two would allow the neuron to hold charge values as low as -126.

**Variable Delay:** In order to maintain a high clock rate (1 MHz), a neuron does not always fire on the global clock cycle in which its threshold is exceeded. In particular, if a neuron fires due to charge events in its last two sub-cycles, then that event will be read one cycle later than expected by a neighboring synapse. Combined with the randomization of reading from neighboring elements, the effect is that for any given chain of elements, every neuron in the chain can delay its output by one cycle.

3 THE DANNA SIMULATORS

It should be evident from the description above that writing a cycle-accurate simulator for DANNA is a challenging task. The first goal in our simulator development was to perform hardware verification. However, as part of our overall research project in neuromorphic computing, the role of the DANNA simulator is quite varied. Below, we detail the goals for DANNA simulation that go beyond hardware verification:

**Communication:** As mentioned above, the input and output to DANNA come in the form of packets that are consumed and generated at the speed of the global clock. The hardware realization of DANNA requires a communication component so that DANNA may talk with a host computer. Originally, this component was realized by a Cypress FX3 USB kit; subsequent research has been to replace this functionality with custom hardware [15]. Software simulation of DANNA has been used both to develop the communications modules, and to drive the packet design.

**Training:** To employ DANNA in applications, we train networks using a genetic algorithm called *Evolutionary Optimization for Neuromorphic Systems* (EONS) [14]. Candidate networks are generated by EONS, randomly at first, but then using reproductive operations on previously evaluated networks. Each network must be loaded onto a DANNA device, which is then directed by the application to execute on a training suite of inputs/tasks. Thus, the activities of loading networks, processing input and output packets, and running the device, are performed many, many times. While the DANNA hardware may be employed for this task, it is typically more efficient to use standard computing resources for training. Therefore, the performance of the DANNA simulator is important for the timely completion of EONS.

**Utilization of available computing resources:** The reason that software simulation is more efficient for training than hardware is the fact that general computing resources are abundant, and much easier to manage and leverage than FPGA’s. EONS parallelizes very naturally [13], which again points to using software simulation for training. Many modern computers also contain GPU’s, which offer additional opportunities for speeding up simulation.

**Exploration of large devices:** The DANNA hardware team has been performing active research in scaling DANNA by composing multiple DANNA chips [5, 15]. This places an additional focus on simulation, not only for verification, but on the ability to simulate very large DANNA networks efficiently.

**Exploration of future hardware:** Similarly, the DANNA hardware team has been exploring new functionalities within DANNA, such as neuron leak, and more advanced plasticity mechanisms. These functionalities are best explored in simulation, to evaluate their potential effectiveness, rather than in hardware.

To summarize, simulation achieves many goals within the research project of DANNA. To achieve these goals, the simulators must have three sometimes conflicting properties: cycle accuracy for hardware development and verification, speed for training, and flexibility for exploration. It is for this reason that we have developed four separate simulators for DANNA.

4.1 Clock-based simulator

The clock-based simulator was the first simulator developed for DANNA. Its primary goal was hardware verification; however, until subsequent simulators were developed, it was also employed for training, producing the first DANNA networks that solved applications such as data classification [12] and pole balancing [1]. This simulator is written in about 500 lines of C++ and occupies a very small memory footprint, requiring only 40 bytes for the representation of each grid element.

The simulator executes by simulating the exact mechanics of each DANNA element during each global clock cycle and sub-cycle. It performs a two-phase pass for each cycle: one pass evaluates state from the previous cycle, and one pass generates state for the next cycle. This level of simulation can be inefficient. For example, a neuron performs a check on each of its incoming connections, regardless of whether there are firing events on those connections, or even whether there are programmed elements on those connections. This level of simulation was effective at identifying subtle timing interactions on the hardware, and because the mechanics of the hardware was not abstracted away (as in later simulators), it was easier to trace the reason for hardware/software mismatches when they occurred.

The clock simulator also simulated the I/O mechanics of DANNA, consuming input packets every global cycle, and similarly producing output packets. Again, this trades off efficiency for realism, and
was helpful in developing the hardware communication modules for DANNA.

4.2 Event-based simulator

The Clock simulator allowed us to achieve a much better understanding of the DANNA model, which we then distilled into a more abstract simulation of discrete events. The events are simply placed into a priority queue as they are generated and the priority queue is processed in time order. This second “Event” simulator is composed of 750 lines of C++, and allows input and output events to be processed directly in the simulation, rather than as packets delivered to a simulated device. Since it only processes events, this simulator is more efficient than the Clock simulator when simulating sparse networks, and when events do not occur frequently with respect to the global clock. This simulator was employed to train networks for NeoN, a surveillance robot navigated by the DANNA FPGA [10]. The training for NeoN employed EONS on 18,000 nodes of ORNL’s supercomputer Titan, for 24 hours.

4.3 GPU simulator - single network

As mentioned above in Section 3, the prevalence of GPU’s in modern computing systems led us to explore how we could leverage these resources in DANNA’s simulation. Due to causality issues with parallelizing an event simulation, we chose the clock simulator to be the basis for the GPU simulation. Our first GPU simulator loads a single network onto the GPU and partitions the elements among the GPU cores, which process each global cycle in parallel. The simulator is written in about 730 lines of C++ and CUDA.

A major challenge with GPU programming is forming problem solutions around the constraints of the GPU. The ideal workload is composed of identical work units with little to no interdependence, and which access memory in a contiguous fashion. The clock simulation at first glance exhibits this workload within each of the two passes of a global cycle. However, because elements in the array may be one of two types, this causes branching that heavily affects performance. To eliminate this divergence, cores are divided into groups of neuron workers and synapse workers. Unfortunately, this creates another issue. For best performance, groups of cores should access data that is contiguous in memory, but groups of workers now access specific elements that are most likely not contiguous. For this simulator though, the benefits of eliminating the divergence outweighs the penalty of non-contiguous memory access.

Moreover, each pass over the array ends with a synchronization point that is rather expensive and happens 32 times per global cycle. This is due to the way elements only read a single neighbor before updating state. This precludes any meaningful use of the GPU’s shared memory. The ideal workload should also have a high computation to data access ratio, often called arithmetic intensity. A high arithmetic intensity is often achieved with the use of shared memory. DANNA simulation has a low arithmetic intensity, largely due to these unavoidable synchronization points.

Even with these issues, we anticipated that the GPU simulator would improve performance over the Event simulator when networks are large and events are frequent. The large networks increase the amount of work being done per pass within a global cycle, and therefore increase the instruction count between synchronizations. Frequent events improve performance relative to the Event simulator, because the Event simulator cannot abstract away as many cycles that are not doing activity.

4.4 GPU simulator - multiple networks

Unfortunately, large networks are at present not the norm with DANNA applications. For example, the DANNA network for NeoN was on a 15 X 15 array. Currently, the most common use case for the simulator is to perform EONS on large populations of small networks. To leverage the GPU for this use case, our fourth simulator runs multiple networks simultaneously, each on a single GPU block. Each network is completely independent. This allows the configuration of elements and inputs to be unique per network. The only restriction is that all networks must be designed for the same array size. While this version also needs to synchronize 32 times per global cycle, it is a bit less costly because it is at the block level rather than the entire GPU. Otherwise, this version has much in common with the single network version.

5 PERFORMANCE

5.1 Stress Test

To compare the performance of the simulators, we employ a dense, recurrent network, created by tiling a 4 X 4 grid composed of four neurons and twelve synapses. This network is shown in Figure 1. While this network doesn’t solve any particular application, it serves as a stress test for the simulators’ performance since most networks generated by EONS are less dense and less recurrent.

![Figure 1: 4 X 4 repeating tile for simulator stress testing. N represents neurons and arrows indicate a synapse connection.](image)

The performance between the clock-based and event-based simulators varies on different conditions. The clock-based simulation’s performance is $O(n\log t)$ where $n$ is the size of the array and $t$ is the number of cycles simulated. It is also minorly affected by network activity. The event-based simulation’s performance is $O(e \log e)$ where $e$ is the number of firing events. It is also minorly affected by array size.

To compare the simulators, we employed networks that were 15 X 15 and 80 X 80 in size. We chose the 15 X 15 size, because it has been a common size used by EONS to solve many of our applications. The 80 X 80 array size was chosen because it is the...
Figure 2: 224 runs of a 15 X 15 grid pattern for 10K cycles

Figure 3: 224 runs of a 80 X 80 grid pattern for 10K cycles

largest DANNA array size to fit on a single FPGA to date. We ran 224 tests for each array size for 10,000 cycles. In each test, we varied the number of events generated by randomizing the inputs to the network. Since the clock-based simulator is minorly affected by the number of events processed, this test should reveal a threshold where it becomes better than the event-based simulator for a particular array size. We ran tests on a machine with two Xeon E5-2697 v3 @ 2.60GHz CPUs and a GeForce GTX TITAN GPU. Simulators were compiled with gcc v5.4.0 and CUDA 9.1.

In Figures 2 and 3, the Event, Clock and GPU Single simulators show the timing of each of the 224 runs as a scatter plot. The GPU Multiple version ran all 224 tests in parallel with a single execution. Thus, its timing is a single line representing the total time of execution divided by 224. This represents the average timing required per test to sequentially run all 224 tests as fast as the GPU Multiple version.

Figure 2 shows the 15 X 15 timings. The single network GPU version does horribly on this test, because it barely uses any of the GPU’s available resources, due to the small array size. The multiple network GPU version, on the other hand, does use all available resources, because it runs all tests at once, in effect simulating a 15 X 15 X 224 array. While this appears great for multiple networks on the GPU, one must remember that this is a comparison of the whole GPU to a single core on the CPU.

As for the CPU versions, the Clock-based simulation starts to surpass the Event-based simulation at approximately 500,000 events over the 10,000 cycles. That means nearly a quarter of all array elements must fire every single cycle for Clock to pull ahead. This is highly uncommon in our experience and why the Event-based simulation greatly speeds up EONS training (see below in Section 5.2).

Figure 3 shows the 80 X 80 timings. Now, single networks on the GPU begin to pull ahead because it is closer to fully utilizing the available resources. Similarly, multiple networks on the GPU show a larger improvement over the CPU versions. With more available work, the GPU is better at dealing with the memory accesses, mentioned in section 4.3. For this test, the intersection of Event and Clock performance is approximately 11,000,000 events. This requires roughly a sixth of all array elements to fire every single cycle for Clock to be superior.

Additionally to get a sense of scaling, we ran a single test on an array size of 1000 X 1000. The results are shown in Figure 4. While we currently do not use networks of this size, it is of interest for future work (e.g. for a reservoir computing model). Here the GPU simulation dominates, and will likely be the simulator of choice for very large dense networks.

Figure 5 shows the timings for these EONS runs, using each of the simulators. For each of the epochs, all 350 entries are classified by each of the 1,000 networks to get a fitness score. It takes a total of 105,000 cycles to run all 350 entries through DANNA. With that fitness, EONS makes a new population of 1,000 networks then starts a new epoch. Thus, over the 20 epochs a total of 20,000 networks are created. In this test, the average number of neurons and synapses in these networks was 34.9 and 61.8 respectively. The average number of events generated over the 105,000 cycles was 26,413.

The Event simulation is magnitudes better suited for this particular application than the Clock simulation. There are several reasons the Event simulation does much better. First, EONS is designed to
utilize all available CPU cores by instantiating multiple simulators and dividing the networks among them for fitness testing. Thus, this compares the usage of all CPU cores rather than a single core like the stress test. The testing machine has two Xeon processors with 14 cores each and hyperthreading, thus allowing 56 threads to run simultaneously. Additionally, networks generated by EONS are not nearly as dense or recurrent as the stress test, and they generate far fewer events. The average element usage for these networks is approximately 13%, and they only generate an average of one event every four cycles.

6 LESSONS LEARNED AND CONCLUSIONS
Software simulators have played a critical part of the DANNA platform’s development. We developed the simulators serially, which in the end was a very good thing for the DANNA project:

- The first simulator (Clock) is cumbersome and inefficient for training; however, it aided in the development of the FPGA and VLSI versions of DANNA, plus the communication hardware. In some cases, hardware design was motivated by use-cases discovered by the simulator. Additionally, it was the very detailed simulation model of the Clock-based simulator that allowed us to abstract away details for the subsequent Event-based simulator.
- The second simulator (Event) takes a straightforward priority-queue approach to simulation that maximizes efficiency in the common case of EONS, which features smaller, less dense networks and sparse spiking. The simplicity of this simulator has allowed it to port seamlessly onto the Titan supercomputer at Oak Ridge National Laboratory, and to 1000+ node clusters at the University of Tennessee. Because EONS parallelizes so naturally, the Event simulator does not have to employ technologies like optimistic simulation to leverage parallelism [6, 11].
- The GPU simulators have proven to be a challenge, largely because the computational properties of the simulators do not match the properties of programs that map well to GPU’s. We ended up with two GPU simulators, one focusing on large networks, and one focusing on batching the operations of multiple networks. As detailed in section 5, each GPU simulator has scenarios in which it greatly outperforms the others. However, for the most prevalent use case of EONS on small, sparse networks, using multiple CPU cores, the Event-based simulator is far superior to either GPU simulator.

An intriguing Neuromorphic computing paradigm known as Reservoir Computing [8] may best leverage the single-network GPU simulator. With Reservoir Computing, a very large randomly generated neural network (the reservoir) is used as a pre-processing stage on classification and/or regression tasks. The goal of the reservoir is to transform the data into data that is linearly separable, and thus can be classified by a trained readout layer. Were one to explore DANNA for use as a reservoir, the single-network GPU simulator would be idea.

To conclude, the simulators have filled an important niche in the DANNA project. Due to some of the quirky features of DANNA, it would have been extremely difficult to find nuanced bugs in DANNA’s implementation and communications without their assistance. Additionally the simulators have enabled many applications to be solved with the EONS training and explore the capabilities of the platform in a timely fashion. The lessons learned with the simulators have also guided design decisions for future DANNA
devices. We believe that software simulators are crucial for the success of any neuromorphic system.

REFERENCES


