Computing Spike-based Convolutions on GPUs

Jayram Moorkanikara Nageswaran and Nikil Dutt
Center for Embedded Systems
Donald Bren School of Information and Computer Science
University of California, Irvine
jmoorkan,dutt@uci.edu

Yingxue Wang and Tobi Delbrueck
Institute of Neuroinformatics
University of Zurich and ETH Zurich
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
yingxue,tobi@ini.phys.ethz.ch

ISCAS Track: 15.9 Sensory Systems: Spike-based systems

I. Demonstration Experience

This demonstration shows the first implementation of a real-time spike-based convolution processing system which combines a spike based dynamic vision sensor (DVS) with parallel graphics processor unit (GPU) computation. Moving objects with different features (shape and size) are presented to the system. In the first demo, the system responses in real time to recognize and keep track of one user specified object and ignore the others. In the second one, the system concurrently extracts several features, and labels the outputs with different colors. Users will enjoy the real-time response and learn about using spike-based sensors combined with conventional procedural processing.

The spikes generated by a DVS are processed through a spiking neural network which computes spike-based convolutions. The network size (128x128 pixels) and the convolution kernel size (minimum 48x48 pixels) make it impossible to compute them on a standard CPU in real time. The spike-based neural network is implemented on an NVIDIA CUDA GPU to achieve real time performance.

Spiking neural networks capture the brain's event-driven style of data processing. They can also offer low latencies when using spike based sensors as inputs [3] and potentially cheaper computation. But algorithms based on this kind of network require highly parallelized computation which cannot be efficiently implemented with conventional sequential processors. GPUs potentially provide powerful platforms for spike-based computing. Here we map Address-Event-Representation (AER) [3] based spike processing onto a GPU. We interface a 128x128 pixel AER DVS to a spiking neural network implemented on GPU for feature detection. We can achieve a speedup of up to 35x on a single NVIDIA GTX280 card when compared to a CPU only implementation.

II. Demonstration Setup and Requirements

There are no special requirements for the setup. The hardware setup consists of a spike-based dynamic vision sensor with 128x128 pixels [1] and a computer equipped with CUDA [4] enabled GPU card. We will either use a laptop with a medium performance GPU or we will bring a desktop with a high end GPU.

The DVS generates asynchronous digital spike events and sends them through the USB interface to the computer. The main CPU of the computer uses local sockets to send and receives the spikes to and from the GPU host process which uses the GPU to compute the feature extraction. The input and output visualization is done through jAER [2]. The setup is shown in Figure.1.

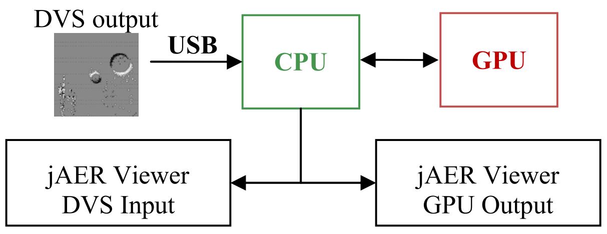


Figure 1: Demonstration setup. jAER [2] is used to interface with DVS chip and to visualize spikes.

References

[1] Lichtsteiner, P., C. Posch and T. Delbruck, "A 128×128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor", IEEE Journal of Solid State Circuits, Feb. 2008, 43(2) 566-576, 2007.

[2] Available: jAER.wiki.sourceforge.net

[3] W. Maass, C.M. Bishop, "Pulsed neural networks", MIT Press Cambridge, MA, USA, 1999

[4] NVIDIA Programming manual Version 2.0, Appendix B. Available: nvidia.com/cuda.

Computing Spike-based Convolutions on GPUs

Jayram Moorkanikara Nageswaran and Nikil Dutt
Center for Embedded Systems
Donald Bren School of Information and Computer Science
University of California, Irvine, USA 92697
jmoorkan,dutt@uci.edu

Yingxue Wang and Tobi Delbrueck
Institute of Neuroinformatics
University of Zurich and ETH Zurich
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
yingxue,tobi@ini.phys.ethz.ch

Abstract— **In spiking neural networks, asynchronous spike events are processed in parallel by neurons. Emulations of such networks are traditionally computed by CPUs or realized using dedicated neuromorphic hardware. In many neuromorphic systems, the Address-Event-Representation (AER) is used for spike communication. In this paper we present the acceleration of AER based spike processing using a Graphics Processing Unit (GPU). In our experiment we interface a 128x128 pixel AER vision sensor to a spiking neural network implemented on a GPU for real-time convolution-based nonlinear feature extraction with convolution kernel sizes ranging from 48x48 to 112x112 pixels. We show parallelism-performance trade-offs on GPUs for single spike per thread, multiple spikes per thread, and multiple objects parallelism techniques. Our implementation can achieve a kernel speedup of up to 35x on a single NVIDIA GTX280 board when compared to a CPU-only implementation.**

I. Introduction

Spiking models are emerging as more biologically plausible than the traditional rate model for neural networks. They better capture the brain's event-driven style of data processing and can also offer lower latencies when using spike based sensors as inputs [1]. Instead of being synchronized by central clocks, this type of network communicates by asynchronous events ('spikes'); meanwhile each neuron continuously, and independently calculates its response according to its inputs. Spiking neural networks can be described as asynchronous systems endowed with highly parallel processing capability.

In most software simulations, neural networks are processed sequentially by a CPU, which means only one neuron is updated at a given time. As a result, the performance degrades quickly with the increase in network size and connectivity. This is especially the case for large connectivity (fan in or out), since sequential processors need to iterate over every connection for each neuron. To speed up the operation, supercomputers or distributed computers are normally used for large-scale neural network simulation. But these solutions incur high cost. Traditional CPU architectures are not designed for parallel processing.

The field of neuromorphic engineering builds hybrid analog/digital VLSI chips to mimic biological architectures present in the nervous system. These include multi-neuron chips, spike-based silicon retina [2], cochlea [3], etc. These systems communicate through the Address-Event-Representation (AER) protocol [4]. In a pure AER approach, each spike is encoded as a digital address, with time representing itself. Most neuromorphic chips typically consist of an array of spiking neurons, which are independent computational units. These neuron arrays operate in parallel and in real-time. Available single or multi-chip AER systems are hard to scale beyond small network configurations [5] and to maintain or modify. Their custom nature also limits outside application.

Improvements in graphics processing units (GPUs) driven by the gaming market provide powerful and inexpensive parallel computing platforms. GPUs were originally developed for graphics rendering and were difficult to program for general scientific applications. But several frameworks [6], led by NVIDIA's CUDA (Compute Unified Device Architecture), allow programmers to more easily harness the parallel processing capability of GPUs with extended semantics for standard C code.

This paper describes our first explorations into efficiently implementing an asynchronous spiking neural network on GPUs for real time processing of the output of an AER sensor. We believe this is the first investigation of AER based spike processing on GPUs. After we introduce the network architecture (Section II), we briefly discuss the GPU architecture (Section III), and analyze different methods to improve the performance of spike processing on GPUs (Section IV). Section V concludes with the main contributions of our approach and future work.

II. Network Architecture

For this initial work we implemented a simple architecture of the neuromorphic hardware system described in the CAVIAR project [5]. In this 3-layer architecture (Figure.1a), a Dynamic Vision Sensor (DVS) [2] is used as the front end, a middle layer of leaky integrate and fire (LIF) neurons computes projective convolutions with predefined kernels [7], and an output winner-take-all (WTA) LIF neuron suppresses neuron activities in the convolution layer to isolate the tracked feature [8]. This architecture illustrates asynchronous processing using a spiking neural network to study the effectiveness of real-time spike processing on GPUs. While this architecture exposes features and difficulties, it will need to be scaled up to a multi-layer architecture for future developments which are capable of more general recognition.

The DVS [2] responds to movement in a scene (temporal contrast), and generates asynchronous AER events. To reduce redundancy and computing time, the events from the DVS are pre-filtered by a refractory spike filter (RSF). The RSF calculates the time difference between the current spike (x,y,t2) and the previous spike (x,y,t1), If the time difference (t2-t1) is less than the refractory period, then the spike is filtered out. The output from the RSF is fed into the middle layer that performs convolution with a defined template weight matrix containing the object features.

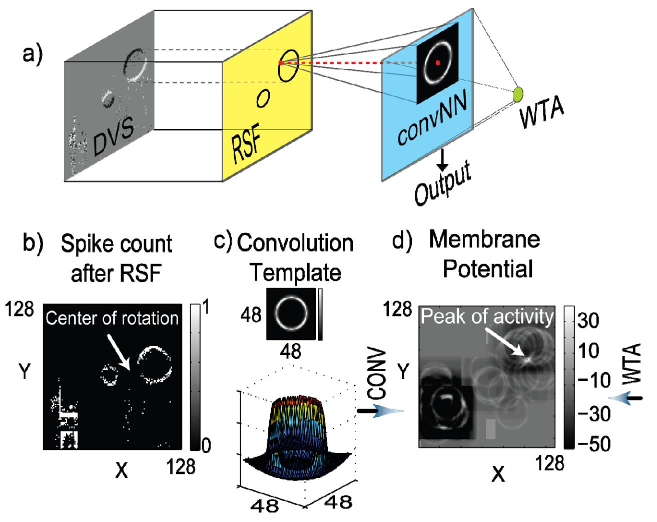
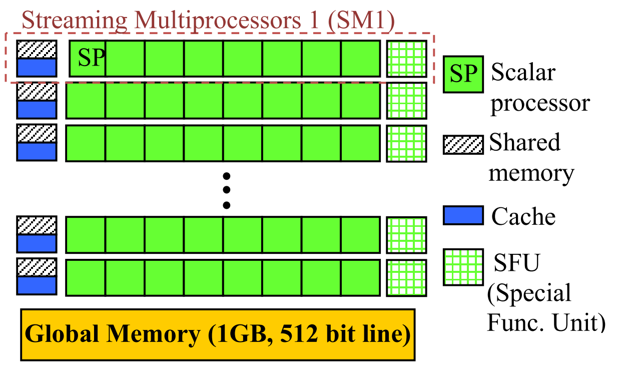


Figure 1. Network architecture (a) Architecture includes three stages: RSF, spiking convolution network (convNN), and WTA; (b-d) Example output from the architecture: (b) RSF output over 10ms, (c) convolution template (the top view and 3D view) and (d) instantaneous membrane potentials of convNN neurons.

Figure.1b-d shows an example to explain the computation at each stage of the network. The scene consists of a large and small circle rotating around a common center of rotation and a static distracter at the bottom left. The ball trajectories form two concentric circles with different diameters. In this example, the distracter leads to a very high spike rate. Figure.1b plots the histogram of the accumulated number of spikes from each pixel over a 10 ms time bin after preprocessing by the RSF. The RSF controls the firing rate per pixel to be less than 100 Hz, so there is at most 1 spike per time bin. These individually timestamped event addresses feed into the 128x128 convolution network. There each event splatts a two dimensional Difference-of-Gaussian template, which defines the circular feature, onto the neuron membranes (Figure.1c), forming a projection field onto the neurons which surround the spiking pixel. The LIF neuron implements a time dependent memory with a leak, which endows the neuron with the ability to detect temporal correlations, rather than simply accumulating input events. The membrane potential (Figure.1d) of each neuron in the convolution layer shows a peak of activity at the center of the large circle because the template feature is most similar to the large ball in the scene. The spikes from the network then compete through a global WTA spiking neuron to suppress the activity of neurons with a lower firing rate [8].

III. GPU ARCHITECTURE

Figure.2 shows a simplified view of the CUDA GPU architecture from NVIDIA [9]. It contains an array of streaming multiprocessors (SMs). Each SM consists of eight floating-point Scalar Processors (SPs), a Special Function Unit (SFU), a multi-threaded instruction unit, a 16KB user-managed shared memory, and 16KB of cache memory.



Streaming Multiprocessors 1 (SM1)
SP
SP Scalar processor
Shared memory
Cache
SFU (Special Func. Unit)
Global Memory (1GB, 512 bit line)

Figure 2. Simplified architectural view of CUDA GPU.

In our experiments we used a single high end GTX280 gaming GPU card that consists of 30 SMs (each operating at 1 GHz), allowing 240 threads to execute concurrently and a potential performance of 300 GFLOPS. Furthermore, each GPU has a hardware thread scheduler that selects a group of threads for execution on each SM. If a thread in the group issues a costly external memory operation, then the thread scheduler switches to a new thread group, allowing a large number of threads (15360 in GTX280) to be active simultaneously. To effectively use the GPU resources, each thread should operate on different scalar data. And to achieve peak memory bandwidth performance, all threads within a group should access the same memory segment of size equal to 128 bytes [9].

IV. GPU MAPPING

We now present the performance improvements on the GPU compared with CPU computation, and trade-offs in mapping the spiking network onto GPUs.

The experimental setup is shown in Figure.3. The main CPU (*thread 1*) is in charge of spike communications by socket connections. Through this thread, the input spikes from the DVS come to the CPU over USB via jAER [10] and the GPU output is sent back to this thread and visualized in jAER. In the CPU-only mode, the main CPU (*thread2*) performs all the computation, including *RSF (*filtering), *ConvNN* (convolution), and *WTA* shown in Figure.1a. In GPU mode, only the *RSF* operation (Figure.1a) is performed on the CPU side. Other parts of the task are assigned to GPU kernel, which contains a block of threads where each thread calculates the membrane potential of one LIF neuron. The static feature template is mapped onto the texture cache available in each SM to speed memory access.

Our GPU performance measurements include the time spent in data transfer between CPU and GPU. The results are averaged over 5 trials. The main CPU was an Intel Core2 Duo (6400) processor running at 2.13 GHz having a single core theoretical performance of 10 GFLOPS. We measured two kinds of speedups: kernel speedup and application speedup. Kernel speedup is the ratio of CPU to GPU execution time for specific functions. The application speedup is the end-to-end application speedup, which includes AER and socket data transfer delays. Comparing the performance of GPU and CPU, a minimum speedup of at least 30x should be feasible for computation dominated applications.

We propose three basic techniques for parallelizing the AER based spike computation, namely: single spike parallelism, multiple spikes parallelism (Figure.4 and Algorithm 1), and multiple objects parallelism. In the remaining section we briefly explain the mechanism, performance, and trade-offs in each of these techniques.

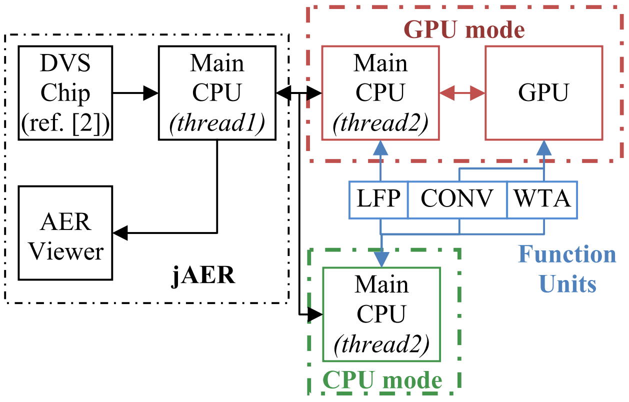


Figure 3. Experimental setup in CPU mode and GPU mode. jAER [10] is used to interface with AER DVS and to visualize AER spikes.

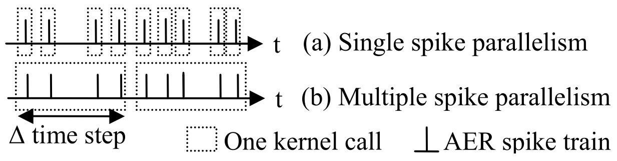

t (a) Single spike parallelism
t (b) Multiple spike parallelism
Δ time step
One kernel call
AER spike train

Figure 4. Illustration of spike parallelism

A. *Single input spike parallelization*

Here the membrane potentials are stored in the global memory, and are updated after each input spike is received (Algorithm 1a). The parallelism is purely in the spatial domain. The number of concurrent GPU threads updating the membrane potentials depends on the size of the template (For a 48x48 template, 2304 neurons are updated for each incoming spike). We incorporate alignment to generate coalesced global memory access. For example, if the 48x48 2D region of neurons that needs to be updated ranges (x,y) from (33,34) to (80,81), we actually operate at a 16-word aligned region starting at (x,y)=(32,34). Thus, threads at locations (32,34), (32,35), and up to (32,80) do not carry out any memory access or computation. Although we might waste some threads operating at the boundary region, the high penalty for unaligned global memory operations are eliminated. Such alignment is necessary since the performance can drop by a factor of 10 due to unaligned or un-coalesced operations. Figure.5 shows the kernel level speedups. The overall speedup obtained by using single input spike parallelism is shown in the lowest curve of Figure.5. For template size from 48x48 to 112x112, the kernel speedup increases from 1.9 to 6.4. This result was not impressive, even though we launch a large number of threads (12544 threads for a 112x112 template). In the test case, about 156,000 spikes are received from the DVS, and each spike launches a kernel. However, the GPU kernel launch overhead is approximately 20us [4], representing about 70% of the total execution time on the GPU. Similar kernel overhead results were measured using the NVIDIA profiler tool. This constrains the overall speedup of the single spike parallelism approach. If future versions of CUDA GPUs can reduce the kernel launch overhead, then single spike parallelism can still benefit from the GPU architecture.

```
Algorithm 1
    sx, sy, st  = input spike location, and time
    nx, ny, nt  = neuron address, and last update time
    v(nx,ny)    = membrane potential at nx,ny
(a) single spike parallelism (executed by each thread)
    1.  // only one spike is given to GPU
    2.  [nx,ny]  = map(threadIdx, blockIdx, sx,sy)
    3   weight   = templateMatrix(nx-sx, ny-sy)
    4.  calculate v(nx,ny)
    5.  fire a spike if (v(nx,ny) > threshold)
(b) multi-spike parallelism (executed by each thread)
    1.  [nx,ny] = map(threadIdx, blockIdx)
    2.  // a group of spike is given to GPU
    3.  for each input spike sx, sy, st
    4.    weight   = templateMatrix(nx-sx, ny-sy)
    5.    calculate v(nx,ny)
    6.    fire a spike if (v(nx,ny) > threshold)
    7.  endfor
```

B. *Multiple input spikes parallelization*

To improve the performance, we group the input spikes based on the temporal proximity of successive spikes. In this case, a time step Δ is defined (Figure.4), and spikes falling into that interval are grouped together and sent to the GPU simultaneously with identical timestamps. This approach reduces kernel launch overhead, CPU-GPU data transfer time, and global memory access. In this scheme every neuron in the 128x128 array is mapped to a GPU thread and the number of threads does not vary with the template size as in section A. Therefore, the membrane potentials can be stored in the fast SM (shared memory), reducing slow global memory access. Algorithm 1b shows an outline of this parallelization approach. As shown in Figure.5, this approach scales well with the template size and time step Δ. Using multiple spike grouping in CPU mode already leads to a 3x speedup. And the maximum GPU over CPU kernel speedup was roughly 22x for a template size of 112x112 with Δ=5 ms. Grouping spikes quantizes time and induces loss of precision in the neuron's membrane potential calculations, which can be measured by the change in the total output spike count. It is negligible as long as 1/Δ is much smaller than the maximum firing rate (100 Hz in our example). Figure.6 shows the impact of Δ on the error in firing count. Spike grouping increases the firing count (the total number of spikes), but does not change the overall quality of feature detection. Similar error curves were obtained for larger template sizes with less than 30% for Δ≤2000us. Similar errors were also observed in CPU mode.

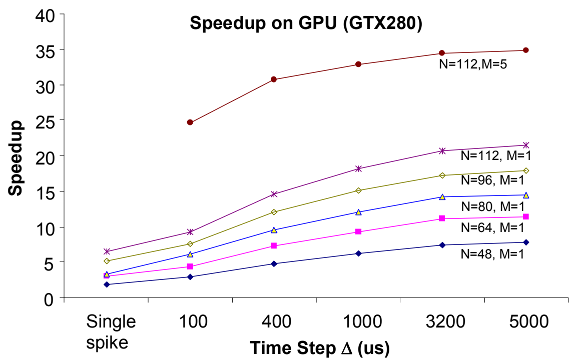


Figure 5. Kernel level speedup by running the application on NVIDIA GPU (GTX280). Δ indicates the time step for grouping, NxN gives the template size, M indicates the number of networks. Error bars are not plotted due to their insignificant variance.

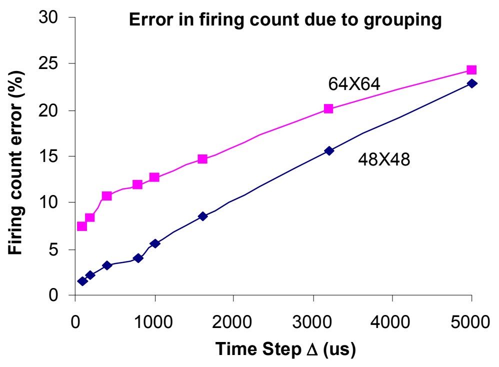


Figure 6. Error in the total firing count for different values of Δ (time step) with template size 48x48 (in blue) and 64x64 (in magenta).

C. *Multiple objects parallelization*

The next level of parallelization is to extract several features simultaneously, which requires running several networks concurrently on the GPU. As a test, we ran up to 5 LIF networks, each using its own template and operated with multiple spike parallelism. The speedup curve with a 112x112 template is shown in Figure.5. The maximum kernel level GPU over CPU speedup was 35x. The speedup starts to saturate beyond Δ = 1 ms, because the GPU has sufficient threads running to achieve maximum performance. The overall application speedup (including socket delays, AER delays, and other parameters) was 13x in GPU mode compared to CPU mode (N=112, M=5).

The GTX280 CUDA-jAER system with a single 64x64 template can process about 200k spikes per second at the application level, computing about one billion connections per second.

V. CONCLUSION

This paper reports the first framework for real-time spike based feature extraction on a GPU using an AER sensor. So far this yields up to about 35x kernel speedup compared to a baseline CPU. Techniques of memory alignment, kernel overhead reduction, and other algorithmic techniques are necessary to achieve this speedup. Our future goal is to incorporate these techniques in a generic framework for real-time simultaneous object tracking and recognition, and also to move towards hierarchical techniques for object recognition using spikes. The set of 4 dedicated convolution chips in [5] can process 64x64 kernels at a rate of 3 M events-per-second, about 4 times faster than what we achieve here, while burning only 1/1000 the power. The price for general purpose, procedural, deterministic digital computation is still evident.

ACKNOWLEDGMENT

We thank the Telluride Neuromorphic Engineering Workshop for the opportunity to initiate this project; and the CAVIAR project for some of the recorded data from the DVS. We also thank NVIDIA for donating the 9800GX2 and GTX280 boards under the Professor Partnership Program.

REFERENCES

[1] W. Maass, C.M. Bishop, "Pulsed neural networks", MIT Press Cambridge, MA, USA, 1999

[2] Lichtsteiner, P., C. Posch and T. Delbruck, "A 128×128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor", IEEE JSSC, 43(2) 566-576, 2008.

[3] V. Chan, S.-C. Liu, and A. van Schaik, "AER EAR: A matched silicon cochlea pair with address event representation interface", IEEE TCAS I: Special Issue on Smart Sensors, 54(1), pgs 48-59 , 2007.

[4] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events", IEEE TCAS II, 47, pp. 416-434, 2000.

[5] R. Serrano-Gotarredona, et al., "CAVIAR: A 45k-Neuron, 5M-Synapse, 12G-connects/sec AER Hardware Sensory-Processing-Learning-Actuating System for High Speed Visual Object Recognition and Tracking", IEEE TNN in press, 2008.

[6] Kayvon Fatahalian and Mike Houston, "A closer look at GPUs", Communications of the ACM, Vol. 51, No. 10, October 2008

[7] R. Serrano-Gotarredona, et al., "On Real-Time AER 2D Convolutions Hardware for Neuromorphic Spike Based Cortical Processing," IEEE TNN, 19(7), pp 1196-1219,in Press. June 2008.

[8] M. Oster, Y. Wang, R. Douglas, and S.-C. Liu, "Quantification of a spike-based winner-take-all VLSI network", IEEE TCAS-I, 55(10) 3160-3169, 2008

[9] NVIDIA Programming manual Version 2.0

[10] Available: jaer.wiki.sourceforge.net