Programmable Smart Machines: A Hybrid Neuromorphic approach to General Purpose Computation

Programmable Smart Machines: A Hybrid Neuromorphic approach to General Purpose Computation

Jonathan Appavoo[1], Amos Waterland[2], Katherine Zhao[1], Schuyler Eldridge[1], Ajay Joshi[1], Margo Seltzer[2], and Steve Homer[1]

[1]Boston University {jappavoo,kzhao,schuye,joshi,homer}@bu.edu
[2]Harvard University {apw,margo}@seas.harvard.edu

We describe our research on integrating neuromorphic devices into a new computing system, where the system is *not* approximate by default and its operation automatically improves as a function of its resources and "experience". We focus on communicating key insights and observations and then sketch our research agenda. We use the term *neuromorphic* broadly to mean any hardware specifically designed for machine learning or inspired by biological neural networks.

There appear to be two basic modes of computing: (1) by logic, and (2) by pattern recognition. To date, there has generally been a gulf between these two modes, with the exception of work on approximate computing [2, 4, 1]. We seek to define and exploit a closer bridge between the two. We define Programmable Smart Machines (PSMs) as *hybrid computing systems that behave as programmed but transparently learn and automatically improve their operation*.

The PSM research agenda explores the use of neuromorphic devices in addressing the challenges set forth by Michie in the Nature '68 paper that introduced *memoization* [3]:

> "It would be useful if computers could learn from experience and thus automatically improve the efficiency of their own programs during execution... When I write a clumsy program for a contemporary computer a thousand runs on the machine do not re-educate my handiwork. On every execution, each time-wasting blemish and crudity, each needless test and redundant evaluation, is meticulously reproduced."

Michie observed that computing the output of any function given its input can be done either by calculation (rule) or memory recall (rote), and concluded that:

> "on each given occasion proceed either by rule, or by rote, or by a blend of the two, solely as dictated by the expediency of the moment ... rule versus rote decisions shall be handled by the machine behind the scenes."

To address these challenges, we integrate pattern recognition into the execution model of a general purpose computer. Our goal is a computing system that (1) automatically learns and recalls patterns in its operation to achieve correctness-conserving speedups by eliminating large fractions of its conventional instruction execution stream, and (2) recognizes and exploits increasingly complex patterns as a function of the size of its neuromorphic hardware resources.

Since the PSM research agenda hybridizes conventional execution with learning, prediction, and auto-associative memory recall, our work is well-positioned to exploit efficient high-capacity neuromorphic hardware.

Execution Model

Two key observations give rise to the PSM research agenda: (1) execution of a von Neumann computer naturally produces a vector-valued binary signal from the dynamics of the *architected state* (all memory and registers), and (2) a forward update of the architected state can be done either by rule (execution) or rote (lookup in a cache of previously executed complete state traces).

These observations link our research agenda to neural networks in specific and machine learning and approximate computing in general. Learning, prediction and recall are at the heart of our design for a computing system that automatically improves as a function of its hardware resources. Neural networks are increasingly pervasive in domains such as image and video processing. Both machine learning researchers and biologists have shown that large neural networks can automatically find and recall complex structure from patterns in binary data.

By coding zeros and ones as e.g. black and white, the complete n bits of instantaneous architected state of a computer can be represented as a $1 \times n$ black and white image, illustrated as a single row of pixels in Figure 1. Execution can then be represented as a video of the architected state as it evolves with each instruction execution. This gives a natural way to interface computer execution to neuromorphic devices whose value has already been proven in the domains of binary-valued image and video processing. Neural networks implemented in neuromorphic hardware can be used to learn, recognize and recall patterns in execution as *predictions* over future frames.

Recognizable clips in "execution movies" can be interpreted as repeated computation that maps the state of the system represented by the first frame of the clip to the system state associated with the last frame. As such, if one is able to recognize, learn and predict reoccurring clips in execution movies then one has captured and compressed patterns in execution over all aspects of the program and its data.

In this fashion a neuromorphic mechanism can store knowledge of execution in the form of recognized patterns. The mechanism can provide feedback on the current execution in the form of predicted frames (states of the system). For a PSM to be effective, it must use this feedback to generate future states of the system that are known to ensue from the

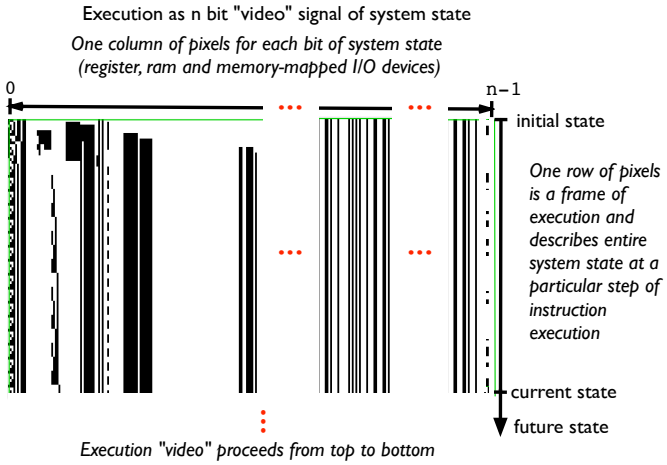

Figure 1: Execution as n-bit "video" signal of system state.

current state. One approach is to view the neuromorphic mechanisms as part of a two stage process. In aggregate the neuromorphic devices encapsulate a massive, first stage, long-term knowledge-base of execution patterns of various scales and complexity. When fed information from the current execution, they generate predicted start states of a clip of computation that it believes will follow based on the patterns it knows. This feedback must be converted into an exact match to a state that will deterministically follow from the current state of the computer or it must be discarded.[1]

For this conversion a second "resolution" stage can be employed. The idea is to use a smaller set of traditional resources to maintain a short term cache of start to end state pairs that is populated based on the neuromorphic predictions. Rather than directly updating the state of the computer with the neuromorphic prediction the prediction is used to generate a cache entry that maps a set of precise start states to a set of resulting end states. To do this the resolution stage allocates traditional cores and memory to determine future states. Given that the prediction defines a complete system state a traditional parallel thread of computation can be used to generate the clip of computation that it represents simply by executing forward. This allows the prediction to be converted into a start and end state pair. This pair is then stored in a cache that is consulted as the computation proceeds. If a match is found the system can directly be advanced to the associated end state. In current and future work we explore more subtle resolution and cache matching procedures that allow us to generalize and reuse cache entries thus eliminating cases in which execution is required. Details of our current resolution process are beyond the scope of this paper (see Waterland et al. [5]).

We see the PSM approach as providing a means for integrating neuromorphic devices into a general purpose computing system that maintains a simple precise and deterministic programming model. We also see PSMs as a way of allowing arbitrary programs to transparently benefit from learning when there is sufficient redundancy in behavior either due to the computation being well structured, or the data that it is operating on, or both. Finally we believe that the PSM model has the advantage that it can allow for "experience" from one system to speed up other systems via a shared hierarchical approach. That is to say while some of the neuromorphic devices could be local to a system, others could be remotely hosted and shared across several machines.

To date we have been using a combination of software machine learning agents to learn and predict each value of the system's state. In particular, we combine the results of two widely-used learning algorithms: logistic regression and linear regression. These two approaches are closely related to single layer neural networks. In the case of logistic regression predictors, each predicted state bit is produced as a linear combination of all bits values plus a bias weight. The weights are updated using a standard gradient descent online update method. Our linear regression predictors also operate on non-overlapping 32-bit receptive fields interpreted as real values. These two approaches are chosen to minimize the software overheads and to permit an online learning model.

Next Steps We will evaluate the use of hardware neural network accelerators in place of software implementations. Our goal is to quantify the implications of using more complex multi-layer networks within the PSM model. Speedup in a PSM is a function of the prediction accuracy and the scales and complexities of the patterns recognized. As has been observed in vision based applications, deeper networks permit improvements on all of these fronts.

To proceed we will first break down our PSM model into a version that is friendly to offline training given that most current hardware neural network accelerators follow this approach. To do this we will assume a model where a hardware accelerator is synthesized to a particular multi-layer perceptron network configuration and then its weights can be loaded or reloaded as needed. As such we will conduct training for each application that we intend to run and store the attendant weights with the application binaries. When we launch an application we will load the neural network accelerators with the associated weights. Given this setup we will evaluate a range of applications, application-specific workloads and a range of network complexities as dictated by a range of hardware resource specifications. To do this we will build upon components from our independent prior work into the design and implementation of a multi-layer preceptron accelerator architecture [1]. We envision an architecture that incorporates network accelerators operating alongside traditional computation units and provides a path to deliver More-than-Moore performance improvements in spite of the harsh landscape of nanoscale CMOS. Interestingly, such an architecture has the potential to leverage *both* precise and approximate regimes of computation for performance improvements. We will migrate portions of the online learning model from our prior software implementation to hardware and investigate increasingly advanced architectures (e.g. HMAX) for the ability to discern more complex patterns.

[1] PSM computing systems also enable an error-bounded form of approximate computing, but we leave this for future work.

References

[1] S. Eldridge, F. Raudies, D. Zou, and A. Joshi, "Neural network-based accelerators for transcendental function approximation," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2014.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, Dec 2012, pp. 449–460.

[3] D. Michie, ""Memo" Functions and Machine Learning," *Nature*, vol. 218, no. 5136, pp. 19–22, Apr. 1968. [Online]. Available: http://dx.doi.org/10.1038/218019a0

[4] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 35–50. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541948

[5] A. Waterland, E. Angelino, R. P. Adams, J. Appavoo, and M. Seltzer, "ASC: Automatically Scalable Computation," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 575–590. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541985