Island Model for Parallel Evolutionary Optimization of Spiking Neuromorphic Computing

Catherine D. Schuman[1], James S. Plank[2], Robert M. Patton[1], Thomas E. Potok[1]
[1]Oak Ridge National Laboratory, Oak Ridge, Tennessee
[2]University of Tennessee, Knoxville, Tennessee

ABSTRACT

Parallel genetic algorithms (PGAs) can be used to accelerate optimization by exploiting large-scale computational resources. In this work, we describe a PGA framework for evolving spiking neural networks (SNNs) for neuromorphic hardware implementation. The PGA framework is based on an islands model with migration. We show that using this framework, better SNNs for neuromorphic systems can be evolved faster.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Genetic algorithms**; • **Hardware** → **Neural systems**;

KEYWORDS

island models, evolutionary optimization, neuromorphic computing, spiking neural networks

1 INTRODUCTION

Parallel genetic algorithms have been a part of the genetic algorithm (GA) landscape for decades as a way to accelerate optimization by exploiting large-scale computational resources [1, 4]. One use case of GAs in recent years is for neural network optimization, also called neuroevolution [2]. An interesting class of neural networks, spiking neural networks (SNNs), have risen in popularity, partly due to their use in low-power neuromorphic systems [8]. GAs provide an attractive option for training SNNs for neuromorphic deployment for a variety of reasons, including the ability to train for a variety of application types, easily operate within hardware constraints, and easily utilize hardware in-the-loop [7]. However, training for SNNs for neuromorphic deployment is computationally difficult and though basic GA approaches can discover solutions to simple problems in a reasonable amount of time, more complex tasks require more complex algorithmic approaches. To help address this issue, in this

work, we build a parallel GA framework based on islands with migration around an existing neuromorphic training framework called EONS. We demonstrate that the approach outperforms the simple parallel technique of islands with no migration.

2 EONS

We implement our parallel GA method around Evolutionary Optimization for Neuromorphic Systems (EONS), a method for designing SNNs for neuromorphic systems. Built on the method described in [7], EONS is a training algorithm integrated into the TENNLab neuromorphic software framework [6], which supports a variety of neuromorphic implementations and applications, such as classification and control tasks. EONS uses a network graph as the genome representation and uses custom reproduction operators, including crossover, merge, and mutation operations. Each EONS executable is compiled with a particular neuromorphic implementation and application. Here, we use the DANNA2 neuromorphic implementation [5], which is a digital spiking implementation that can either be implemented on a field programmable gate array (FPGA) or fabricated in a custom chip implementation. The application that we use is pole balancing problem, a canonical task for neuroevolution [3]. We use the version of the task that does not provide the velocities of the cart and pole to the SNN as input.

3 ISLAND MODEL WITH MIGRATION

Our islands framework implementation is written in C++ and uses MPI for communication. We assume that at the beginning of execution we are allocated C cores and $P = C$ processes are created. One process is the island manager (IM), and the remaining $P - 1$ processes are the island workers (IWs). The IM begins by creating a list of EONS commands to execute, each of which is sent to an IW. Each IW waits until a command is received and then forks a child process that executes the EONS command. The IW collects information about how the EONS process is performing via a pipe. Intermittently, the IW sends a status update about the EONS process's progress to the IM. This message contains the current generation for that IW's EONS process and the current best fitness value. If the IW has not previously sent a network to the IM or if a new best network has been evolved since the last message, the IW sends the new network to be integrated into the IM's migrant pool.

The IW then waits to receive one of the following commands from the IM: **migrant**, indicating that a migrant network will be sent to the IW; **go on**, indicating that the EONS process should continue uninterrupted; and **new EO**, indicating that the IM has sent a new command to be executed, so the current EONS process should be killed. If the IW receives a migrant from the IM, it writes that migrant network to a file, which the EONS process integrates into its population. If the IM directs the IW to create a new EONS

process, the IW kills its current EONS process and then creates a new one.

The IM decides which command to send to the IW based on previous status updates from the IW. First, the IM checks the new status to see if it includes a migrant network. If there is a new migrant, the IM attempts to integrate the migrant into its migrant pool. If the migrant pool is full, the new migrant is added into the pool and the worst performing member of the pool is removed. The IM then decides whether the IW's EONS process should be killed. The IM tracks the fitness scores from the status updates for each island. If the fitness score has stagnated over multiple status updates, then the IM directs the IW to kill the existing EONS process and gives it a new EONS command to execute. Otherwise, the IM selects a migrant from the IM's migrant pool and sends it to the IW. We construct the eligible migrant pool for the IW in order to limit duplicated fitness evaluations. Migrants in the migrant pool are *ineligible* if they came from that IW or they have already been sent as a migrant to that IW. If there are no eligible migrants, we send the "go on" signal to the IW. Otherwise, we randomly select a migrant from the networks in the eligible migrant pool to send to the IW. Then, the IM waits for status updates from other IWs.

4 RESULTS

We ran 10 tests with migration and 10 sets without migration on 16, 64, 256, and 1024 cores. These tests are performed for the DANNA2 neuromorphic implementation on the pole balancing with no velocities task. In this task, the maximum fitness value of 15,000 corresponds to a DANNA2 SNN that is capable of balancing the pole on the cart for five minutes from six starting conditions. For islands without migration, we do not track the performance of each EONS population and replace those that stagnate. We demonstrate that islands with migration performs better than simply running many parallel EONS processes and taking the best network (i.e., islands without migration). Figure 1 shows the results for islands with and without migration on ten tests for each. For each of these tests, the same random number generator seeds were used to create the initial EONS populations on each island, so differences in performance are not a consequence of differences in initial populations. Figure 1 shows that for both islands with and without migration, a higher number of cores results in better resulting networks. This is not surprising because increasing the number of cores effectively increases the overall population size in both cases, resulting in more exploration of the solution space. The differences in performance across the ten runs for each parameter setting are due to the different initial populations. We can also see in Figure 1 that, on average, islands with migration outperform islands without migration. This difference in performance indicates that by allowing for migration between islands, better solutions can be shared across populations. This means that islands with populations that are not performing well will eventually receive better performing migrants from other islands, which introduce more diversity into the island population and lead to faster evolution as discussed in [9].

5 DISCUSSION AND CONCLUSION

We show in this work that islands with migration consistently outperforms islands without migration. That is, better solutions are reached in the same amount of time with the same computational
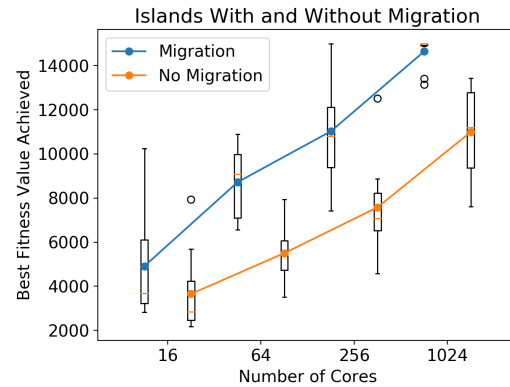
Figure 1: Differences in performance between islands with migration (shown in the first set of boxes, mean value plotted in blue) and islands without migration (shown in the second set of boxes, mean value plotted in orange).

resources if migration is allowed between islands. The impact of this work is that, using the islands with migration framework, we can now train SNNs for neuromorphic systems to better solutions in the same amount of time using the same computational resources. This will enable the neuromorphic community to explore training SNNs for more complex tasks.

6 ACKNOWLEDGEMENTS

REFERENCES

[1] E. Cantú-Paz. 1998. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis* 10, 2 (1998), 141–171.
[2] D. Floreano, P. Dürr, and C. Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (2008), 47–62.
[3] F. Gomez, J. Schmidhuber, and R. Miikkulainen. 2006. Efficient non-linear control through neuroevolution. In *European Conference on Machine Learning*. Springer, 654–662.
[4] Y. Gong, W. Chen, Z. Zhan, J. Zhang, Y. Li, Q. Zhang, and J. Li. 2015. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing* 34 (2015), 286–300.
[5] J.P. Mitchell, M.E. Dean, G.R. Bruer, J.S. Plank, and G.S. Rose. 2018. DANNA 2: Dynamic Adaptive Neural Network Arrays. In *Proceedings of the International Conference on Neuromorphic Systems*. ACM, 10.
[6] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose. 2018. The TENNLab Exploratory Neuromorphic Computing Framework. http://neuromorphic.eecs.utk.edu/pages/research-publications/. (August 2018).
[7] C.D Schuman, J.S. Plank, A. Disney, and J. Reynolds. 2016. An evolutionary optimization framework for neural networks and neuromorphic architectures. In *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 145–154.
[8] C.D. Schuman, T.E. Potok, R.M. Patton, J.D. Birdwell, M.E. Dean, G.S. Rose, and J.S. Plank. 2017. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963* (2017).
[9] D. Whitley, S. Rana, and R.B. Heckendorn. 1998. The Island Model Genetic Algorithm: On Separability, Population Size and Convergence. *Journal of Computing and Information Technology* 7 (1998), 33–47.