# Solving Constraint Satisfaction Problems Using the Loihi Spiking Neuromorphic Processor

Chris Yakopcic[1], Nayim Rahman[1], Tanvir Atahary[1], Tarek M. Taha[1], Scott Douglass[2]
[1]*Dept. Of Electrical and Computer Engineering*, *University of Dayton,* Dayton, OH, USA
[2]*Human Effectiveness Directorate, Air Force Research Laboratory, Wright Patterson Air Force Base,* OH, USA
[*]cyakopcic1@udayton.edu

***Abstract*—In many cases, low power autonomous systems need to make decisions extremely efficiently. However, as a potential solution space becomes more complex, finding a solution quickly becomes nearly impossible using traditional computing methods. Thus, in this work we present a constraint satisfaction algorithm based on the principles of spiking neural networks. To demonstrate the validity of this algorithm, we have shown successful execution of the Boolean satisfiability problem (SAT) on the Intel Loihi spiking neuromorphic research processor. Power consumption in this spiking processor is due primarily to the propagation of spikes, which are the key drivers of data movement and processing. Thus, this system is inherently efficient for many types of problems. However, algorithms must be redesigned in a spiking neural network format to achieve the greatest efficiency gains. To the best of our knowledge, the work in this paper exhibits the first implementation of constraint satisfaction on a low power embedded neuromorphic processor. With this result, we aim to show that embedded spiking neuromorphic hardware is capable of executing general problem solving algorithms with great areal and computational efficiency.**

***Keywords—Spiking neural networks, Intel Loihi, neuromorphic hardware, constraint satisfaction, SAT***

## I. Introduction

Autonomous systems are being increasingly utilized in a variety of domains, including both mobile systems (UAVs, cars, and robots) as well as planning systems. In these systems, autonomous decision making must be performed at a rapid rate in many cases. Thus, traditional algorithms like exhaustive search are no longer viable when time is of supreme importance. Similarly, traditional commercial computing systems may be abandoned for more exotic embedded architectures when Size, Weight, and Power (SWaP) limitations are of supreme importance.

Embedded neuromorphic hardware [1-3] is a promising technology for the advancement of low SWaP high efficiency processing, and spiking neuromorphic hardware provides additional advantages. In spiking neural network (SNN) hardware, spikes are used to propagate information between neural network layers and cores. This is an extremely efficient design, as most energy is expended only when information is being processed [1,2]. However, to utilize SNN hardware at maximum efficiency, algorithms must be developed to take advantage of this unique hardware platform. In this work we propose a method for using the Intel Loihi manycore neuromorphic processor to solve constraint satisfaction problems (CSPs).

Work in this paper implements a subset of CSP known as the Boolean Satisfiability problem (SAT), which is a well-known combinatorial problem that determines whether a certain formula represented in Conjunctive Normal Form (CNF) is satisfiable [4]. CNF representation is basically the conjunction (AND) of several short clauses and each clause is a disjunction (OR) of Boolean variables. The entire formula is considered to be satisfiable if all the clauses in that formula are satisfied. This means at least one of the literals in each clause must evaluate to 'true.' SAT problems are considered one of the most fundamental problem types within mathematical logic, reasoning, machine learning, and many other theoretical domains [5], and they have the potential to be extremely computationally expensive [6].

Therefore, in this work we aim to alleviate this computational expense by presenting a method for solving SAT problems using SNNs. Furthermore, we verify this algorithm experimentally by executing a number of SAT scenarios on the Intel Loihi manycore neuromorphic processor. With this work we aim to show the efficacy of SNN hardware as applied to inference engines for reductions in power consumption without the sacrificing generality or problem scope. The proposed implementation will work on any problem that can be written in the CNF format, which allows for its use in a wide range of potential applications.

The Loihi spiking processor was introduced in [1,2]. The 60mm$^2$ chip was developed using Intel's 14nm process. each chip contains 128,000 neurons, and the system is capable of implementing hierarchical connectivity, dendritic compartments, synaptic delays, and programmable synaptic learning rules.

Given the recent release of the Loihi system, we are aware of very little research that has been published utilizing the Intel Loihi Processor [1,2,7-10]. Recently, it was shown that the Loihi system is approximately 38 times more power efficient than a GPU based system [7] for real-time DNN inference. Results in [8] show Loihi can perform head direction SLAM with 100× lower power than a CPU. Furthermore, [9] describes an olfaction-inspired learning algorithm that runs on Loihi that can

978-3-9819263-4-7/DATE20/©2020 EDAA

learn patterns 3000× more efficiently when compared to a deep autoencoder or other conventional methods. Work in [10] shows how the Loihi processor can be used to perform high efficiency asset allocation for tactical mission planning. Overall, work published thus far demonstrates that significant SWaP and efficiency gains can be obtained when moving computation from traditional systems onto the Loihi spiking neuromorphic processor. With our work we aim to continue this trend, showing that SAT can also be performed on SNN hardware.

To the best of our knowledge, this work represents the first implementation of SAT on embedded neuromorphic hardware. Few research groups have also proposed SNN algorithms for constraint satisfaction [6,11-13], and few research groups have also implemented SAT in non-portable neuromorphic hardware [14,15]. However, with the advent of embedded neuromorphic spiking systems [1-3], we propose the possibility exists to implement SAT-based decision making systems on extremely efficient and portable devices.

This paper is organized as follows: Section II describes the constraint satisfaction problem we have targeted, as well as background on how we have implemented our solution. Section III describes in detail how the we use the Loihi processor to perform constraint satisfaction in an SNN form. Section IV describes the experimental results when running the proposed algorithm on the Loihi system and Section V provides a results discussion. Section VI provides a brief conclusion.

## II. Constraint Satisfaction

### A. Background

There are two types of algorithms used to solve SAT problems, complete (backtracking based search algorithms) [16,17] and incomplete (greedy algorithms) [18,19]. If one or more solutions exist, then it is guaranteed that a complete SAT algorithm will be able to determine any and all solutions by searching the entire solution space. If solution does not exist, then a complete SAT algorithm will prove that the problem is unsatisfiable. On the other hand, incomplete SAT algorithms try to satisfy as many clauses as quickly as possible, but they cannot prove that a problem is satisfiable or not. The advantage of the incomplete SAT algorithms is their speed and computational efficiency when compared to complete SAT algorithms.

One of the most commonly used complete SAT algorithms is the Davis Putnam Loveland Logemann (DPLL) algorithm [16], which was first introduced in 1960. An alternative algotihm GRASP [17], was proposed in 1995 based on the DPLL principals. The Conflict Driven Clause Learning (CDCL) learning process was the main contribution of GRASP. This clause learning process can drastically narrow the search space and increase the performance of the solver. Several other efficient SAT algorithms such as MiniSat [20], Chaff [21], BerkMin [22], CryptoMiniSAT [23] were also introduced, which all build on the DPLL algorithm.

In case of incomplete SAT algorithms, the local search algorithm is the most widely utilized [18,19]. This method uses different heuristics to search a solution space randomly, and gradually moves to the final solution with the help of some objective function. Incomplete algorithms consider an optimization problem and try to satisfy the maximum number of clauses in some certain time frame if a completely satisfiable solution cannot be found. Local search algorithms primarily include the greedy local search (GSAT [18]) and the random walk GSAT (WSAT [19]).

### B. Proposed Approach

In this work we aim to show that a spiking neuromorphic processor (specifically the Intel Loihi system) can be used to solve SAT problems using a general algorithm applicable to a variety of problems. Thus, any problem that can be converted to a set of literals and clauses using conjunctive normal form may be applied to the proposed system to obtain a solution. In this work, a literal is a binary variable, and an array of these binary variables represents a possible solution to a constraint satisfaction problem. Additionally, a clause can be thought of as rule that constrains the solution space that may depend on one or more literals. A possible solution is considered a valid solution when all clauses are satisfied.

In this work we have implemented an incomplete SAT solver, which will iterate until a single valid solution is found (or the system is stopped due to the inability to find a solution). The block diagram in Fig. 1 displays the solution finding process used in this work. While incomplete solvers provide less information, they have the ability to return a single solution quicker using less computing resources. Thus, if the user's goal is to obtain a valid solution quickly, the proposed system will not only be appropriate, it will also excel due to the high efficiency data propagation mechanisms innate to the chosen SNN neuromorphic hardware.

To implement this algorithm using networks of spiking neurons. The first step was to convert an incomplete SAT solving method to one that could be solved primarily with vector matrix operations. Thus, signals can be propagated through layers of spiking neurons and problem specific information can be stored in weights between the neuron layers. The next section uses an example problem to demonstrate the specifics of how SAT was ported to the Loihi system.

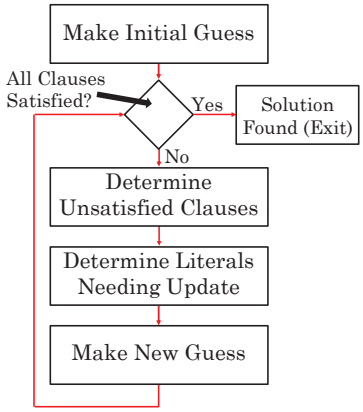


Fig. 1. Block diagram of the proposed SAT solving method.

## III. Loihi Spiking Implementation

The following section describes the process flow of the constraint satisfaction problem solving process as carried out by the Loihi processor. Fig. 2 displays the a more detailed block

diagram that accounts for the order of spike propagation that results in a solution to the SAT problem. The plots in this section correspond to solving a graph coloring problem with 6 vertices and 3 colors, requiring 18 literals and 30 clauses.

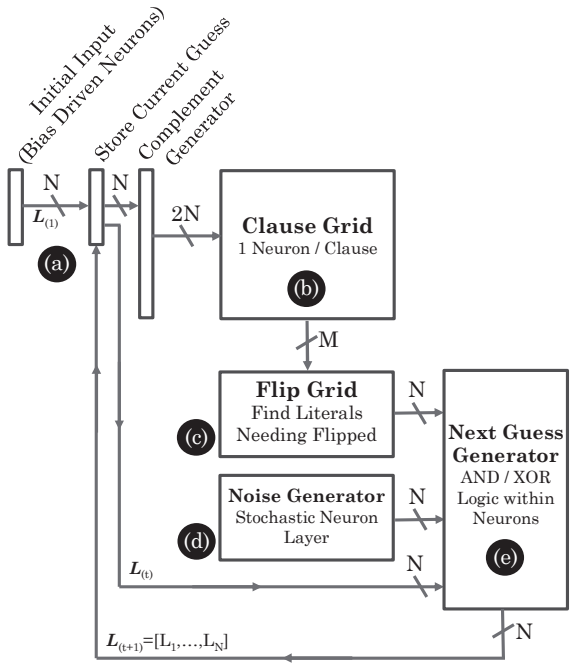


Fig. 2. Block diagram describing the process flow of the SAT solving method on the Loihi system including (a) the input processing block. (b) the *Clause Grid* responsible for tracking unsatisfied clauses, (c) the *Flip Grid* that is responsible for determining which literals need to be flipped to obtain a valid solution, (d) the *Noise Generator* that introduces stochastic behavior into the literal flipping process, and (e) the *Next Guess Generator* that carries out literal flipping based on the *Flip Grid*, the *Noise Generator*, and the last guess.

### A. Input Processing

This algorithm is based on a feedback process where the literals $L_1,\ldots,L_N$ in a given problem update at the beginning of each cycle. However, an input is still needed to start the feedback process. Thus, the Loihi system is initially driven with the *Initial Input* neuron layer consisting of $N$+1 neurons, where $N$ is equal to the number of literals in a given SAT problem (the $N$+1 neuron is a bias required to support data propagation). This initial neuron layer is driven by a constant bias current, and every neuron is set to fire upon the beginning of the initial cycle. However, this layer must only fire once, or else a chaotic spiking pattern will be observed throughout the system. Thus, an additional neuron layer is connected to this initial input layer to suppress any future spikes using negative feedback. This input suppression layer has a strong excitatory connection to itself, and a strong inhibitory connection to the initial input generator. Fig. 3 (a) shows the input signal used to initiate the feedback process.

The *Store Current Guess* layer is needed only to manage the handoff from the initial guess to the future guesses. This layer is not driven by a bias current (like the previous layer). Each neuron $i$=1,…,$N$+1 in this layer is set to spike any time an input spike is present at its input to simply copy the spike pattern from the previous layer.

To perform the required computation to determine satisfied and unsatisfied clauses, an input string is required that contains both the literals and their complement. Thus, the *Complement Generator Layer* is used to both copy the input guess and generate the complement by inverting the current guess, effectively doubling the size of the input and sustaining a single bias. The extended literal string of length 2$N$+1 is shown in Fig. 3 (b). The first spike column in Fig. 3 (b) corresponds to the initial input, and the following spike columns represent the new literal prediction after each SAT cycle.

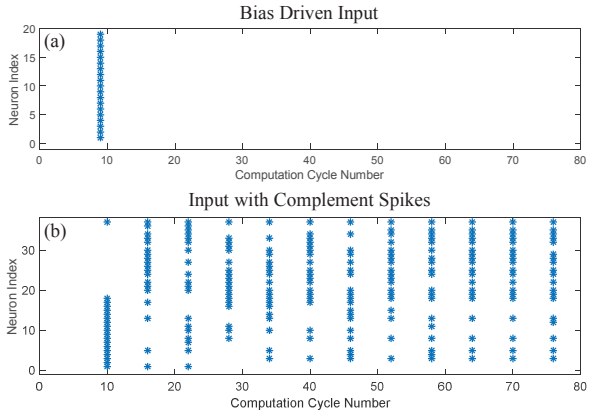


Fig. 3. Plots displaying the (a) the initial input used to drive the feedback spiking process and (b) the signals at the output of the *Complement Generator*.

### B. Clause Grid

Once the complement signal is generated, the entire spike array (of length 2$N$ + 1) will propagate to the input of the *Clause Grid* neuron layer. This layer signifies the first block of significant computation. The *Clause Grid* contains a weight matrix that stores all clauses for a given problem. Each input row in the matrix corresponds to either a literal, or a literal complement. Each column in the *Clause Grid* corresponds to a clause within a given SAT problem. The *Clause Grid* is arranged in such a way that each output neuron will fire that corresponds to an unsatisfied clause. The spike array that is output from the *Clause Grid* (of length $M$) is then fed into the *Flip Grid*. Fig. 4 displays the spikes generated by the (a) *Clause Grid* and (b) the *Flip Grid*. The *Clause Grid* outputs 31 spikes (one for each clause plus a bias) and the *Flip Grid* outputs 19 spikes (one for each literal plus a bias).

### C. Flip Grid

The *Flip Grid* is responsible for determining which literals are contributing to each unsatisfied clause. Thus the spikes that are output from the *Flip Grid* alert the system which literals need to be changed to generate a solution that satisfies all clauses. The *Flip Grid* contains a weight matrix similar to the *Clause Grid*, as it also stores a connection between the clauses and the literals that are utilized in each of these clauses. The *Clause Grid* and *Flip Grid* are the only two neuron layers whose synaptic information is problem specific. Thus, during a problem initialization process these two weight matrices must be programed according to the problem to be solved by the system.

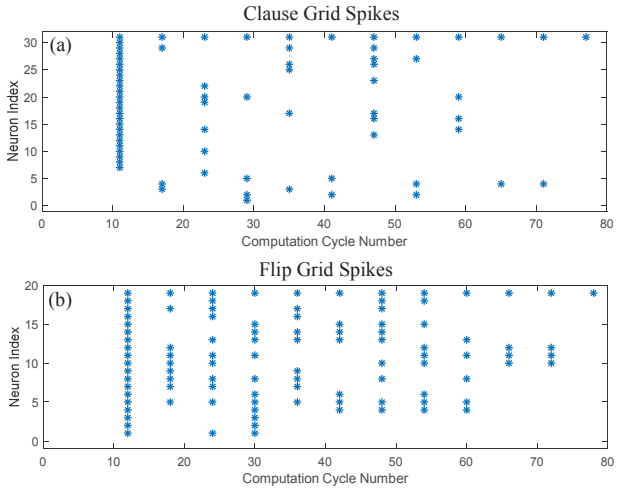


Fig. 4. Plots displaying spiking outputs from (a) the *Clause Grid* and (b) the *Flip Grid*.

### D. Noise Generator

The solution finding algorithm is very likely to get stuck in local minima if all invalid literals are flipped during every SAT cycle. Therefore, a *Noise Generator* was implemented to add some stochastic behavior to the solution finding process. To implement this random behavior required for solution generation, the Loihi stochastic neuron functionality was used to generate an array randomly spiking neurons. Fig. 5 (a) displays the output of the stochastic neuron layer used to implement a nondeterministic solution finding process.

### E. Next Guess Generator

This component in the block diagram is responsible for compiling all three sources of state information (the *Flip Grid* output, the *Noise Generator* output, and the previous guess) to generate a new guess. This is completed using layers of spiking neurons to implement digital logic. The equivalent circuit implemented using these layers of spiking neurons is shown in Fig. 6.

The plots in Fig. 5 display three different spiking signals that exist within the *Next Guess Generator*. These include the *Flip Grid* output spikes, the *Noise Generator* output spikes and the result of sending both of these signals through an AND gate, which was probed at the Pre-XOR node (see Fig. 6). Notice that the continuously spiking stochastic neurons perform a (pseudo) random reduction of the *Flip Grid* spikes. The Pre-XOR output spikes are the result of this reduction, leading to the desired nondeterministic solution finding process.

The last step in the computation cycle in this system is to send the output signal $\boldsymbol{L}_{(t+1)}$, which becomes the new literal vector, back to the *Store Current Guess* layer to start the next SAT cycle. The plots in Fig. 7 display the completion of this process. The output signal representing the $\boldsymbol{L}_{(t+1)}$ signal in Fig. 7 (b) can be determined by completing the XOR of the previous literal guess $\boldsymbol{L}_{(t)}$ in Fig. 7 (a) and the Pre-XOR signal in Fig. 5 (c). Furthermore, the next column of input spikes can be seen to be equivalent to the last column of XOR output spikes, visually demonstrating the feedback process. This feedback process continues until all clauses are satisfied, which causes the Stop Signal in Fig. 7 (c) to fire which has a strong inhibitory connection to all neurons in the Complement Generator, thus destroying the feedback loop. Notice the Stop Signal fires in Fig. 7 (c) immediately after all clauses are satisfied in Fig. 4 (a).

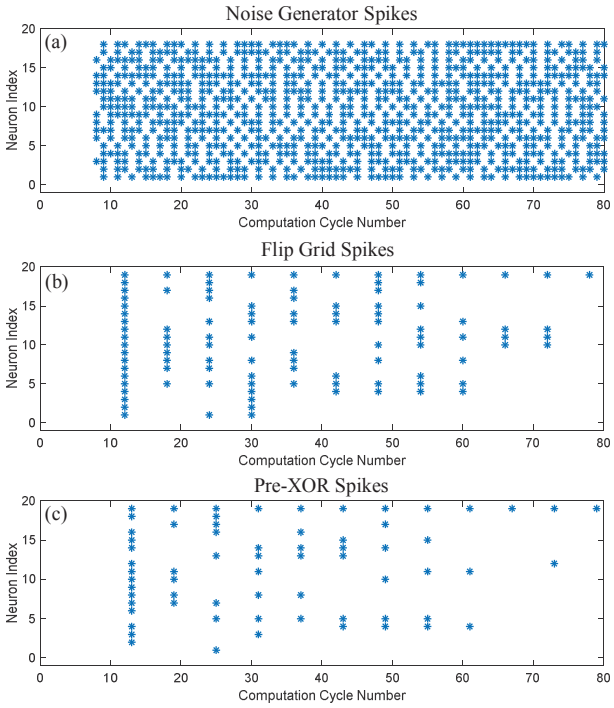


Fig. 5. Plots displaying the spikes observed at (a) the stochastic neuron layer output, (b) the *Flip Grid* output, and (c) the Pre-XOR node.

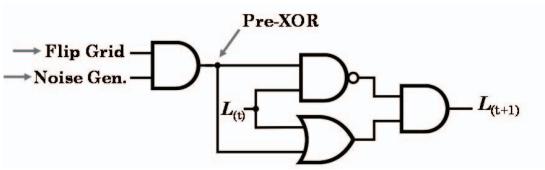


Fig. 6. Equivalent circuit for the logic being performed by the *Next Guess Generator* (implemented in Loihi using neuron layers).

## IV. Experimental Results

Several graph coloring problem (GCP) scenarios were solved on the Loihi hardware using the method proposed in the previous section. In each case a solution was found for this series of SAT problems that had varying difficulty. The results are displayed in Table I. The total number of neurons $T_N$ was determined by adding all neurons in each layer in the network layout. For a given problem, $T_N$ can be extrapolated using equation (1), where $M$ represents the number of clauses and $N$ represents the number of literals within the problem. When looking at this equation, it can be observed that increasing the number of literals in a problem is much more computationally expensive than increasing the number of clauses.

$$T_N = 14 + M + 12N \quad (1)$$

The total time to solution was found using the benchmarking tools contained within the Loihi SDK. Two

examples of the timing data collected during runtime are displayed in Fig. 8. Notice that the runtime per cycle is not constant, but is variable based on the amount of computation required at that cycle.

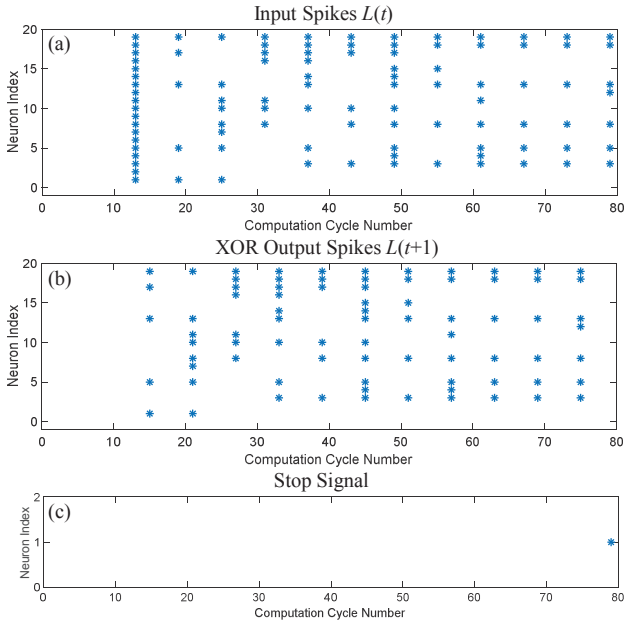


Fig. 7. Plots displaying spikes observed at (a) the *Store Current Guess* output, the *Next Guess Generator* output, and (c) the *Stop Signal* output.

TABLE I. EXPERIMENTAL RESULTS DISPLAYING SUCCESSFUL SOLVING OF THE GRAPH COLORING PROBLEM SCENARIOS

| GC Problem Type | | Constraint Satisfaction on Loihi | | | | | |
|---|---|---|---|---|---|---|---|
| *Vertices* | *Colors* | *Literals* | *Clauses* | *Neurons* | *Solved?* | *Iterations* | *Total Time (ms)* |
| 5 | 3 | 15 | 29 | 223 | *Yes* | 10 | 0.24 |
| 6 | 3 | 18 | 30 | 260 | *Yes* | 12 | 0.78 |
| 10 | 4 | 40 | 78 | 572 | *Yes* | 135 | 9 |
| 10 | 5 | 50 | 95 | 709 | *Yes* | 43 | 2.1 |
| 15 | 6 | 90 | 213 | 1307 | *Yes* | 394 | 15.2 |
| 15 | 7 | 105 | 225 | 1499 | *Yes* | 380 | 14 |

## V. DISCUSSION

With the results in this paper, we believe we have successfully displayed the efficacy of solving SAT problems on embedded spiking neuromorphic hardware, providing a low SWaP alternative for the execution of this problem. To quantify the low power potential, Fig. 9 presents the results of a number of power measurements obtained using the measurement tools within the Loihi SDK. Given that these tools measure the power of the entire circuit board, not the Loihi chip, a large static power is observed in each measurement. To determine the dynamic power required to execute the examples in Table I, the neurons and synapses required to carry out these problems are copied to multiple cores within a single Loihi chip (which are then all executed in parallel). We then derive a simple linear model that predicts the additional power required to solve one more copy of a given SAT problem. Table II displays the static and dynamic power of two different graph coloring problems. Since each of these problems requires 3 cores to solve, dynamic power is equal to three times the scaling constant in each linear model (see Fig. 9). Static power is considered to be the average of the bias term in each of the linear models. This result shows that while the total power consumption is greater than 1 W, the power required to solve these problems is significantly lower. This static power could likely be significantly reduced for a small scale system that possesses fewer hardware components.

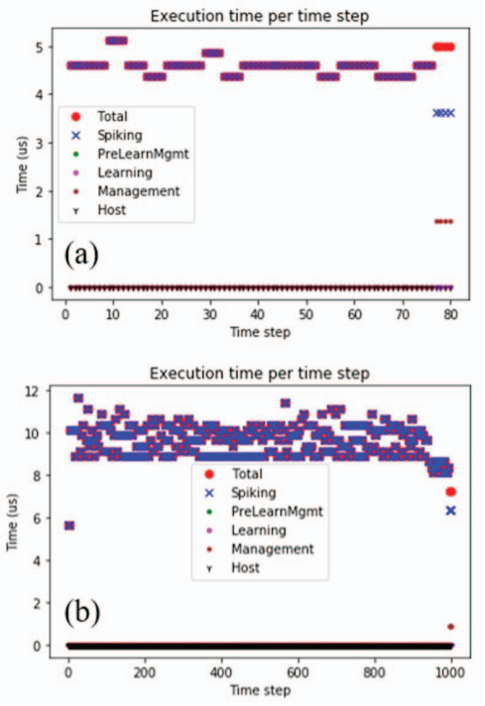


Fig. 8. Plots displaying the executing time at each cycle for (a) a GCP problem with 6 vertices and 3 colors and (b) a GCP problem with 10 vertices and 4 colors.

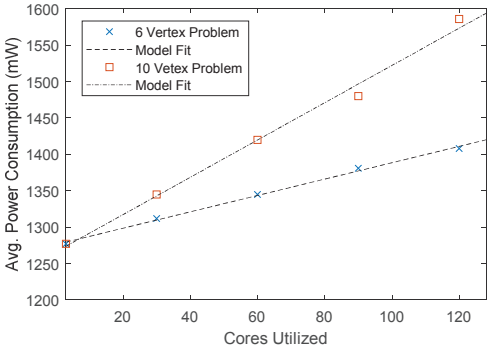


Fig. 9. Plot showing increase in power consumption as two graph coloring problems are repeated in multiple cores on a Loihi chip, where the $y$-intercept represents the static power of the circuit board utilized. Models fitting the data for the 6 and 10 vertex problems: $f_6 = 2.562x + 1266$ and $f_{10}=1.125x + 1276$.

As future work, we are developing methods to objectively compare the performance of this algorithm with the performance of other published alternatives. To do this, we will

have to determine fair methods for approximating the power consumption of alternative SAT hardware such as FPGA implementations [24], or embedded hardware that is carrying out constraint satisfaction.

TABLE II. APPROXIMATE POWER CONSUMTION WHEN EXECUTING THE PROPOSED SAT ALGORITHM USING THE LOIHI SYSTEM

| GCP Number of Vertices | Loihi Static Power | Loihi Dynamic Power |
|---|---|---|
| 6 | 1271 mW | 3.375 mW |
| 10 | 1271 mW | 7.687 mW |

## VI. CONCLUSION

In this paper, we present a method for solving constraint satisfaction problems using spiking neural networks. Using the Loihi spiking neuromorphic processor, we show that this algorithm can be successfully carried out on embedded low power neuromorphic hardware. To the best of our knowledge this is the first implementation of an SNN algorithm on embedded SNN hardware.

We have several directions for future work that go beyond this initial feasibility study. First we aim to perform a detailed power analysis when running algorithms on the Loihi, as well as other systems. This will allow us to better detail the advantages embedded neural hardware provides. Furthermore, the largest scenario performed in this work required about 1500 neurons, which is a relatively low number. In the future we plan to scale up this system and utilize multiple chips to determine the if large scale AI problems can be solved.

## ACKNOWLEDGMENT

This work was performed with the support of the Air Force Research Laboratory. Approved for public release: 88ABW Cleared 11/21/19; 88ABW-2019-5732.

## REFERENCES

[1] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," IEEE Micro, vol. 38, no. 1, pp. 82-99, 2018.

[2] C.-K. Lin, A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery, H. Wang, "Programming Spiking Neural Networks on Intel's Loihi," Computer, vol. 51, no. 3, pp. 52-61, 2018.

[3] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, A. Amir, D. B.-D. Rubin, F. Akopyan, E. McQuinn, W. P. Risk, and D. S. Modha "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," International Joint Conference on Neural Networks (IJCNN), pp. 1-10, Dallas, TX, August 2013.

[4] S.D. Prestwich, CNF encodings, in: Handbook of Satisfiability, pp. 75-97, vol. 185, 2009.

[5] A. Darwiche and K. Pipatsrisawat, Complete Algorithms, in: Handbook of Satisfiability, pp. 99-130, vol. 185, 2009.

[6] J. Binas, G. Indiveri, M. Pfeiffer, Spiking Analog VLSI Neuron Assemblies as Constraint Satisfaction Problem Solvers, IEEE International Symposium on Circuits and Systems (ISCAS), Montreal, QC, Canada, May 2016.

[7] P. Blouw, X. Choo, E. Hunsberger, C. Eliasmith, "Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware," arXiv :1812.01739, submitted 4 Dec. 2018.

[8] G. Tang, A. Shah, K. P. Michmizos, "Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional SLAM," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2019: https://arxiv.org/abs/1903.02504

[9] N. Imam and T. A. Cleland, "Rapid online learning and robust recall in a neuromorphic olfactory circuit," 2019: https://arxiv.org/abs/1903.02504

[10] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, A. Beigh, and S. Douglass, "High Speed Cognitive Domain Ontologies for Asset Allocation Using Loihi Spiking Neurons," International Joint Conference on Neural Networks, pp. 1-8, Budapest, Hungary, July, 2019.

[11] U. Rutishauser, J.-J. Slotine, and R. J. Douglas, "Solving constraint-satisfaction problems with distributed neocortical-like neuronal networks," Neural Comput., vol. 30, no. 5, pp. 1359-1393, May 2018.

[12] K.Corder, J. V. Monaco, Manuel M. Vindiola, "Solving Vertex Cover via Ising Model on a Neuromorphic Processor," IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, Florence, Italy, May, 2018.

[13] Z. Jonke, S. Habenschuss, and W. Maass, "Solving Constraint Satisfaction Problems with Networks of Spiking Neurons," Frontiers in Neuroscience, vol. 10, no. 118, pp. 1-16, March 2016.

[14] G. A. F. Guerra and S. B. Furber, "Using Stochastic Spiking Neural Networks on SpiNNaker to Solve Constraint Satisfaction Problems," Frontiers in Neuroscience, vol. 11, no. 174, pp. 1-13, Dec. 2017.

[15] K. E. Hamilton, C. D. Schuman, S. R. Young, N. Imam, and T. S. Humble, "Neural Networks and Graph Algorithms with Next-Generation Processors," IEEE International Parallel and Distributed Processing Symposium Workshops, pp. 1195-1203, May 2018, Vancouver, BC, Canada.

[16] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," Communications of the ACM, vol. 5, no. 7, pp. 394–397, 1962.

[17] J. P. M. Silva and K. A. Sakallah, "GRASP—A New Search Algorithm for Satisfiability," Proceedings of International Conference on Computer Aided Design, pp. 220–227, San Jose, CA, Nov. 1996.

[18] B. Selman, H. J. Levesque, D. G. Mitchell, "A new method for solving hard satisfiability problems." Tenth National Conference on Artificial Intelligence, pp. 440–446, San Jose, CA, 1992.

[19] B. Selman, H. A. Kautz, and B. Cohen, "Noise Strategies for Improving Local Search", AAAI, pp. 337– 343, 1994.

[20] N. Eén and N. Sörensson, "An Extensible SAT-solver," In: Giunchiglia E., Tacchella A. (eds) Theory and Applications of Satisfiability Testing. SAT 2003. Lecture Notes in Computer Science, vol 2919. Springer, Berlin, Heidelberg.

[21] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," 38th annual Design Automation Conference, pp. 530–535, 2001.

[22] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust Sat-solver," Discrete Applied Mathematics, vol. 155, no. 12, pp. 1549–1561, June 2007.

[23] K. Wu, T. Wang, X. Zhao, and H. Liu, "CryptoMiniSAT solver based algebraic side-channel attack on present," International Conference on Instrumentation, pp. 561–565, 2011.

[24] K. Bousmar, "A Pure Hardware k-SAT Solver for FPGA," IEEE 5th International Congress on Information Science and Technology (CiSt), pp. 481-485, Marrakech, Morocco, Oct. 2018.