Leveraging the Manycore Architecture of the Loihi Spiking Processor to Perform Quasi-Complete Constraint Satisfaction

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 problem space becomes more complex, finding a solution quickly becomes nearly impossible using traditional computing methods. Thus, in this work we show that constraint satisfaction problems (CSPs) can be solved quickly and efficiently using spiking neural networks. Constraint satisfaction is a general problem solving technique that can be applied to a large number of different applications. To demonstrate the validity of this algorithm, we show successful execution of the Boolean satisfiability problem (SAT) on the Intel Loihi spiking neuromorphic research processor. In many cases, constraint satisfaction problems have solution sets as opposed to single solutions. Therefore, the manycore architecture of the Loihi chip is used to parallelize the solution finding process, leading to a quasi-complete solution set generated at extreme efficiency (dynamic energy as low as 8 micro joules per solution). 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, the proposed SAT algorithm was customized for spiking neural networks 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 capable of generating a solution set. In general, we show that embedded spiking neuromorphic hardware is capable parallelizing the constraint satisfaction problem solving process to yield extreme gains in terms of time, power, and energy.**

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 high speed 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 limitations are of supreme importance.

Embedded spiking neuromorphic hardware [1-3] is a promising technology for the advancement of low power, portable, high efficiency processing. 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, as well as parallelize, constraint satisfaction problems (CSPs).

The presented work 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 (or literals). 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. We utilize the Intel Loihi SNN processor to verify the proposed algorithm experimentally by executing different SAT scenarios. We leverage the manycore architecture of the Loihi chip to implement many SAT solvers in parallel. Each parallel solver executes with different initial conditions; thus each parallel solver has a high likelihood of converging to a different solution. We show this proposed manycore SAT implementation is capable of generating solution sets at extreme energy efficiency. In many applications of the SAT algorithm, solution sets are very attractive as multiple solutions can then be ranked against different criteria to determine the best path forward.

To the best of our knowledge, this work represents one of the first implementations of SAT on embedded neuromorphic

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

hardware, with the exception of [7]. In [7] we show SAT can be implemented using SNN hardware, and show that a single valid solution can be found for a given problem. In this work we have extended and streamlined our algorithm to efficiently determine sets of valid solutions by parallelizing the algorithm. In this work we also present a power, energy, and timing analysis that provides a detailed account of the efficiency gains obtained through parallelization.

A few other research groups have also proposed SNN algorithms for constraint satisfaction [6-10], and a few research groups have gone so far as to implement SAT in non-portable neuromorphic hardware [11,12]. However, with the advent of embedded neuromorphic spiking systems [1-3], we propose parallel SAT-based decision making on extremely efficient and portable devices.

This paper is organized as follows: Section II provides background information on constraint satisfaction, as well as the Loihi processor. Section III provides a high level description of how SAT is implemented on Loihi, and Section IV provides a more detailed walkthrough of the process. Section V contains a results analysis when using the multicore parallel SAT system, which describes the solutions obtained as well as a power, time, and energy analysis. Section VI provides a discussion of the benefits spiking neural networks provide for this type of problem, and Section VII concludes the paper.

II. Background

A. Constraint Satisfaction

SAT problems are typically solved using one of two types of algorithms. The first type is referred to as complete (backtracking based search algorithms) [13,14], and the second type is referred to as incomplete (greedy algorithms) [15,16]. 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 determine a valid solution as quickly as possible, but they cannot prove whether 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 [13] that was introduced in 1960. An alternative algorithm based on the DPLL principles was proposed in 1995, which is called GRASP [14]. The Conflict Driven Clause Learning (CDCL) process was one of the primary contributions of GRASP. This clause learning process drastically narrows the search space and increases the performance of the solver. Several other efficient SAT algorithms such as MiniSat [17], Chaff [18], BerkMin [19], CryptoMiniSAT [20] were also introduced, which all build on the DPLL algorithm.

The local search algorithm [15,16] is one of the most regularly used incomplete SAT algorithms. Different heuristics are used to search a solution space randomly in this case, and the algorithm gradually iterates to the final solution with the help of some objective function. Local search algorithms primarily include the greedy local search (GSAT [15]) and the random walk GSAT (WSAT [16]).

B. The Loihi Spiking Neural Network Processor

The Loihi spiking processor was introduced in [1,2]. We are aware of little research that has been published utilizing the Intel Loihi Processor [1,2,21-24]. It has been shown that the Loihi system is approximately 38 times more power efficient than a GPU based system [21] for real-time DNN inference. Results in [22] show Loihi can perform head direction SLAM with 100× lower power than a CPU. Furthermore, [23] describes an olfaction-inspired learning algorithm that runs on Loihi that can learn patterns 3000× more efficiently when compared to a deep autoencoder or other conventional methods. Work in [24] shows how the Loihi processor can be used to perform high efficiency asset allocation for tactical mission planning. In general, experiments that have been published thus far utilizing the Loihi processor have shown that substantial size, weight, and power gains can be had by moving computation from traditional systems to SNN hardware. With our work we continue this trend, showing that parallel SAT solving can also be performed on spiking neuromorphic hardware.

III. Loihi SAT: Architecture Overview

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.

In this work we made it our goal to develop a general SAT solving technique that can be applied to virtually any problem. 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 solution is considered valid when all clauses are satisfied.

In this work we have implemented a SAT solver that is capable of returning a set of satisfiable solutions. However, the system is unable to tell if all possible solutions have been determined. Therefore, we refer to this system as a quasi-complete SAT solver; it can generate a valid set of solutions, but it cannot determine if this is the complete set of solutions. This degree of capability falls between the incomplete solver (which may find a single solution) and a complete solver (which returns all possible solutions).

The block diagram in Fig. 1 displays the solution finding process used in this work. To implement this algorithm using networks of spiking neurons, the first step was to convert a 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 algorithm described in Fig. 1 is capable of generating a single valid solution. Implementing this algorithm on a Loihi

chip requires three cores, one to implement several layers of spike driven neurons, one to implement a spike generator, and one to implement neurons that possess a stochastic behavior. Fig. 2 shows how the SAT solver algorithm in Fig. 1 can be replicated a maximum of 42 times within a single Loihi chip, as each chip contains 128 cores. The noise generator core in each solver is initialized with different initial conditions. Thus, depending on the number of valid solutions for a given SAT problem, it is likely each solver will iterate to a different valid solution.

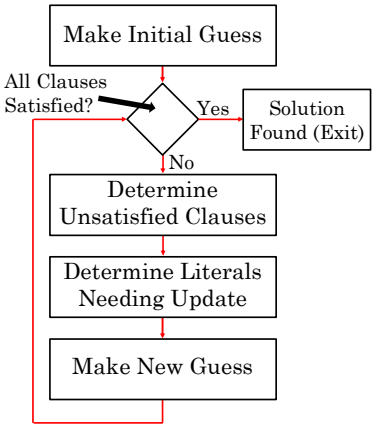

Fig. 1. Block diagram showing how each solver in the parallel SAT system generates a solution.

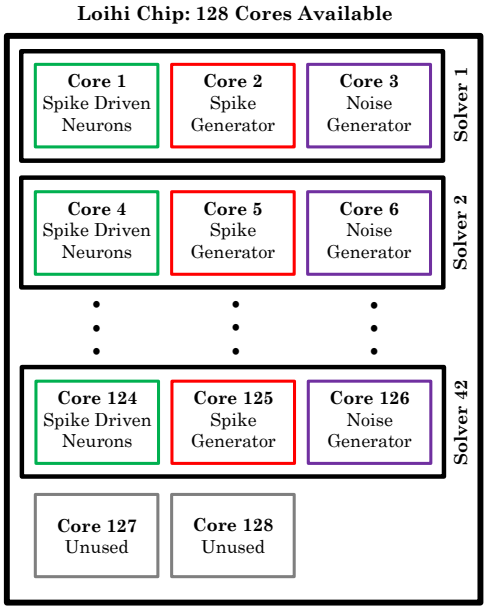

Fig. 2. Overview of how the parallel SAT approach is distributed over the 128 cores present in the Loihi chip.

IV. Loihi SAT: Case Study

The following section provides the reader with a more detailed understanding of how a given solver within the parallel SAT system (see Fig. 2) is implemented in spiking neural network form. In this section, the solution finding process of a graph coloring problem with 6 vertices and 3 colors, requiring 18 literals and 30 clauses is demonstrated.

Fig. 3 displays the a more detailed block diagram for a single solver block, and each component of this solver is described in the following subsections.

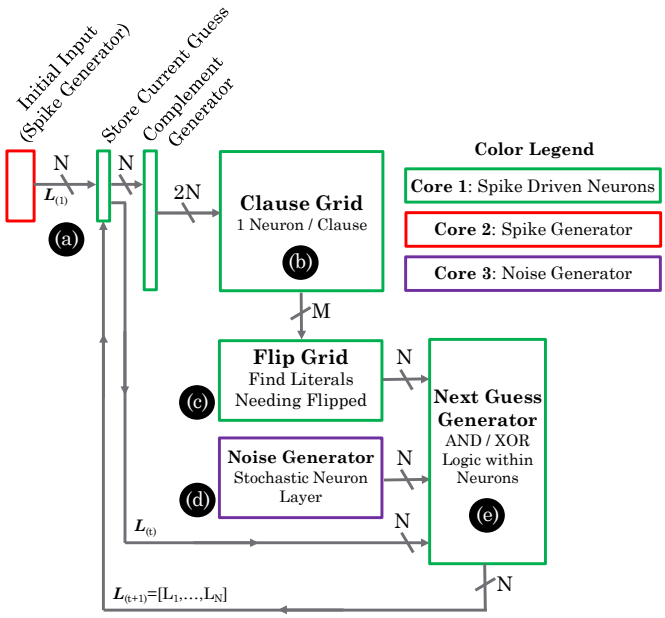

Fig. 3. Block diagram describing the process flow of the SAT solving method on the Loihi system including (a) the input processing block. (b) the *Clause Grid* responsible for tracking unsatisfied clauses, (c) the *Flip Grid* that is responsible for determining which literals need to be flipped to obtain 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 by a spike generator block that simply supplies an array of spikes to start the process. The *Initial Input* neuron layer (see Fig. 3 (a)) consists 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).

The *Store Current Guess* layer is needed only to manage the handoff from the initial guess to the future guesses, as spikes cannot be fed back in to the spike generator used to initialize the process. Each neuron i=1,…,N+1 in this layer will spike any time a spike is present at its input. As a result this neuron layer simply copies the spike pattern input from the previous layer.

To determine which clauses are satisfied and unsatisfied, an input string is required that contains both the literals and their complement. Carrying the literal complement through the neuron layers makes it much easier to program neurons to fire based on the absence of spikes in the literal signal, which is required in some layers in this system. 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 2N+1 is shown in Fig.

4 (b), where the first spike column corresponds to the initial input, and the following spike columns represent the new literal prediction after each SAT cycle.

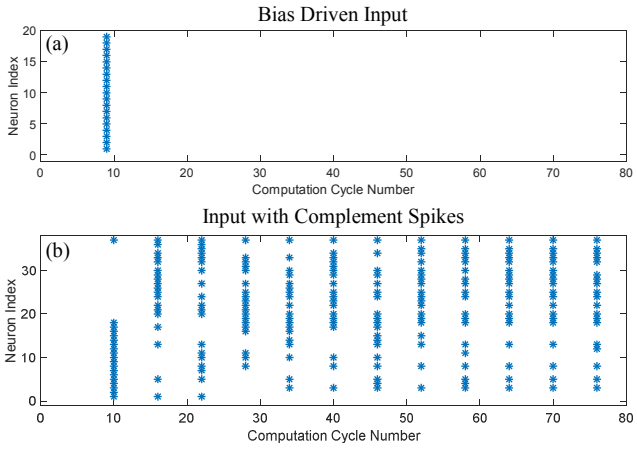

Fig. 4. Plots displaying the (a) the initial input used to drive the feedback spiking process and (b) the signals at the output of the *Complement Generator*.

B. Clause Grid

Once the complement signal is generated, the entire spike array (of length $2N+1$) will propagate to the input of the *Clause Grid* neuron layer. This layer holds 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 set up so that each of its neurons whose weights store an unsatisfied clause, will fire in a given SAT cycle. The spike array that is output from the *Clause Grid* (of length M) is then fed into the *Flip Grid*. Fig. 5 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).

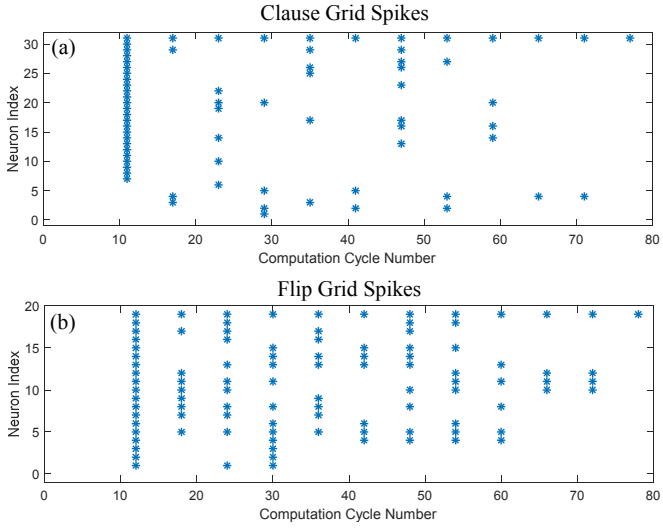

Fig. 5. Plots displaying spiking outputs from (a) the *Clause Grid* and (b) the *Flip Grid*.

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* notify 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.

D. Noise Generator

If all of the invalid literals are flipped during every single SAT cycle, the algorithm will have a high likelihood of getting stuck at local minima. To remedy this, 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 of randomly spiking neurons. Fig. 6 (a) shows the output of the stochastic neuron layer that was used to execute this nondeterministic solution finding process.

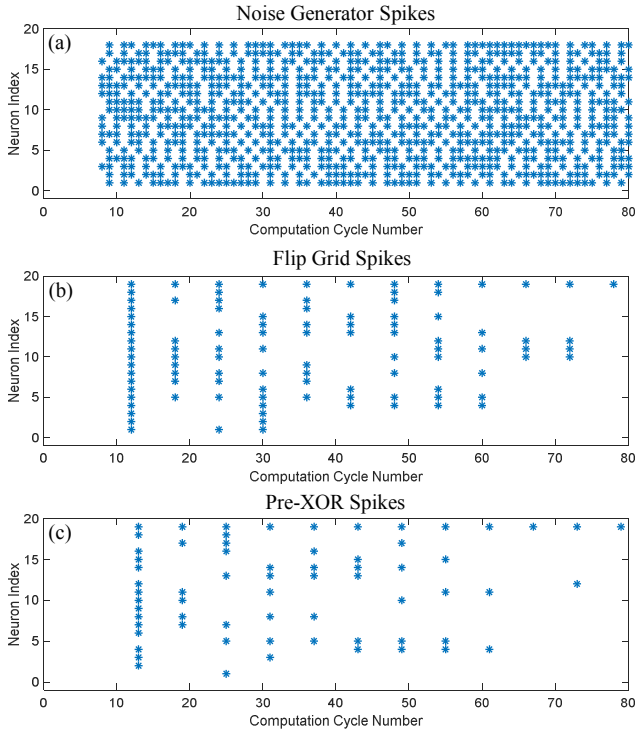

Fig. 6. Plots displaying the spikes observed at (a) the stochastic neuron layer output, (b) the *Flip Grid* output, and (c) the Pre-XOR node.

E. Next Guess Generator

The *Next Guess Generator* 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. The logic required for carrying out the next guess generation is displayed in Fig. 7. However, in this design layers

of spiking neurons are used to perform the functionality of each individual logic gate.

The plots in Fig. 6 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. 7). 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.

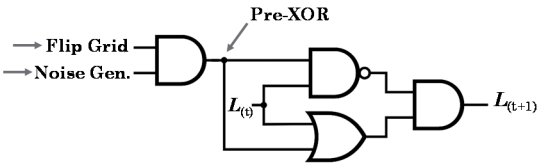

Fig. 7. Equivalent circuit for the logic being performed by the *Next Guess Generator* (implemented in Loihi using neuron layers).

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. 8 display the completion of this process. The output signal representing the $\boldsymbol{L}_{(t+1)}$ signal in Fig. 8 (b) can be determined by completing the XOR of the previous literal guess $\boldsymbol{L}_{(t)}$ in Fig. 8 (a) and the Pre-XOR signal in Fig. 6 (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. 8 (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. 8 (c) immediately after all clauses are satisfied in Fig. 5 (a).

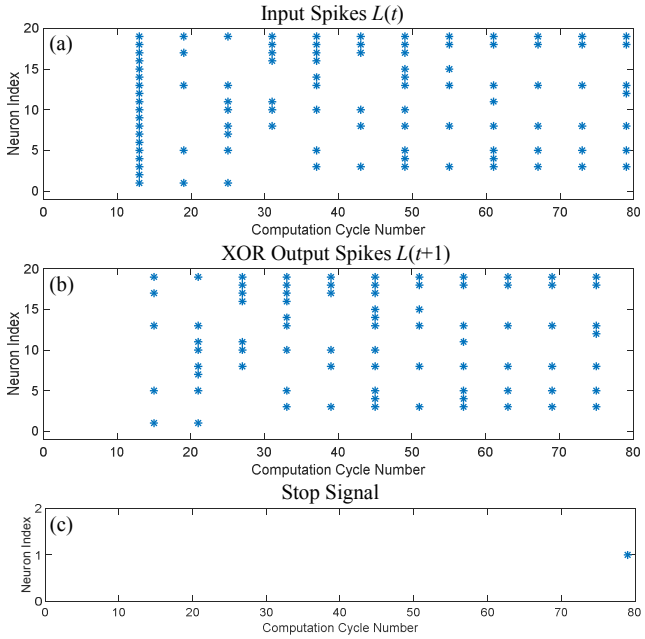

Fig. 8. Plots displaying spikes observed at (a) the *Store Current Guess* output, the *Next Guess Generator* output, and (c) the *Stop Signal* output.

F. Complete System Neuron Count

The total number of neurons required to solve a given problem can be determined by adding all neurons in each layer in the network layout.

The total neurons can be determined using equation (1), where M represents the number of clauses and N represents the number of literals within a given problem. When looking at this equation, it can be observed that increasing the number of literals in a problem is more computationally expensive than increasing the number of clauses.

$$T_N = 14 + M + 11N \quad (1)$$

This equation provides a slightly reduced number of neurons compared to the equation provided in [7], as the input layer of the algorithm was further optimized in this work.

V. Paralellization Study

The process detailed in Section IV explains how a single solution is obtained using the proposed spiking neuron approach for constraint satisfaction. This section describes the results obtained when executing 42 solvers on the Loihi chip (the maximum that would fit on 128 cores) in parallel. Each of these solvers will simultaneously iterate to a solution. Because the stochastic neuron layer was primed with different initial conditions for each solver, each solver should traverse the solution space differently. This will allow the parallel system to return a set of unique solutions as opposed to a single valid solution.

For this work we remotely log in to a server operated by Intel that houses the Loihi cloud system. Loihi programming was performed using the NxSDK version 0.8.0, also developed by Intel. The Loihi cloud contains several Loihi systems, and we utilize the *nahuku08* board for this work which contains 8 Loihi chips. The following experiments only require a single Loihi chip, but the power measurement tools in this version of the NxSDK software are unable to discriminate between active and unutilized chips. Thus, it could be argued that the static power figures presented could be dramatically lowered if it were possible to turn off unutilized parts of the Loihi hardware.

A. Solution Space Examination

Two different graph coloring problem (GCP) scenarios were examined in this study. The 6 vertex problem described previously, as well as a 10 vertex problem. Each of the problem specifics are detailed in Table I.

These two problems are very similar in application, but very different in terms of solution space. The 6 vertex problem only has 30 valid solutions, thus the number of solvers on the Loihi chip exceeds the number of valid solutions. On the other hand, the number of valid solutions to this particular 10 vertex problem greatly exceeds the number of solvers available.

TABLE I. Size and Solution Space of the Graph Coloring Problems Examined in This Study

Problem Size	Colors	Literals	Clauses	Neurons Required	Possible Solutions
6 Vertex	3	18	30	242	30
10 Vertex	5	50	95	659	311,573

Each GCP scenario was executed using the parallel Loihi SAT system, and a description of the solutions obtained is presented in Table II. Each of the two GCPs was executed for three different runtime cases. The shutoff point was set first to 500 Loihi computation cycles, and then this number was increased to 1000. Finally, the system was left running until all solvers were able to find a solution. This came to 1165 and 1475 cycles for the 6 and 10 vertex GCPs respectively. As the number of computation cycles increases, it is more likely that a larger number of unique solutions will be determined. The user of this system can analyze this time and solution number tradeoff to determine which is more important for a given application: finding a few solutions quickly or finding the maximum number of solutions by increasing runtime.

Notice that in the case where all solvers converged to a solution, the number of unique solutions found is still less than 42. This is because some solvers converged to the same solution. Since the 6 vertex problem has fewer possible valid solutions, finding a larger number of unique solutions is more difficult. In this case 42 parallel solvers found 15 unique solutions. However, when the number of valid solutions is abundant as in the 10 vertex problem, 40 unique solutions were found using 42 parallel solvers.

TABLE II. NUMBER OF SOLUTIONS FOUND FOR EACH GRAPH COLORING PROBLEM BASED ON DIFFERENT RUNTIME LIMITS

Problem Size	Max. Cycles	Unique Solutions Found	Avg. Cycles to Solution	Solvers Without Solution
6 Vertex	500	11	259	23
	1000	14	476	2
	1165	15	509	0
10 Vertex	500	25	255	15
	1000	38	410	2
	1475	40	451	0

B. Power, Time, and Energy Analysis

The total time to solution and power during runtime for this SAT system were found using the benchmarking tools contained within the Loihi SDK. This software allows the user to accurately measure the power consumed within a Loihi chip, and precise timing can be obtained that tracks computation time per execution cycle. This performance profiling suite was used to generate the results in this section.

First, power figures were obtained when running each of the two GCP cases (see Table III). The larger number of neurons required to carry out the 10 vertex problem leads to just a slight increase in power. Static and dynamic power were determined by determining the difference in power required when executing 1 solver versus 42 solvers. From these numbers we can approximate the dynamic energy required per solver based on this rate. Then we determined static power by subtracting the dynamic power required for a single solver from the total power required for a single solver.

Overall power data in Table III looks very promising, with a dynamic power of only 3 to 8 mW per solver. Additionally, it should be noted that the static power presented is that of the entire circuit board housing several Loihi chips. It is likely that static power could be greatly reduced if the goal were to develop a custom high efficiency system with only one or two Loihi chips.

TABLE III. STATIC AND DYNAMIC POWER REQUIREMENTS WHEN EXECUTING EACH GCP SCENARIO

Problem Size	Total Power (mW)	Static Power (mW)	Dynamic Power (mW)	Dynamic Power / Solver (mW)
6 Vertex	1420	1274	146.5	3.488
10 Vertex	1592	1269	322.7	7.683

Next, we collected runtime information for each GCP study, and determined energy consumption for each case (see Table IV). As each study executes in the ms time scale, the total energy consumed in each case falls between 3 and 33 mJ. However, dynamic energy per solver is between 7 and 102 µJ. These values demonstrate the efficiency of the spiking neural network architecture within the hardware utilized. The Loihi architecture is extremely efficient for a problem such as this one, where all computations are handled via spike trains through weight matrices of high sparsity.

TABLE IV. RUNTIME AND ENERGY RESULTS FOR EACH GCP SCENARIO EXECUTED ON THE LOIHI SAT SYSTEM

Problem Size	Max. Cycles	Runtime (ms)	Total Energy (mJ)	Static Energy (mJ)	Dynamic Energy (µJ)	Dynamic Energy Per Solver (µJ)
6 Vertex	500	2.240	3.181	2.853	328.133	7.813
	1000	4.248	6.032	5.410	622.280	14.816
	1165	4.887	6.940	6.224	715.886	17.045
10 Vertex	500	4.833	7.694	6.135	1559.527	37.132
	1000	9.338	14.866	11.853	3013.213	71.743
	1475	13.252	21.097	16.821	4276.194	101.814

TABLE V. DETERMINING ENERGY PER UNIQUE SOLUTION FOR EACH GCP SCENARIO EXECUTED ON THE LOIHI SAT SYSTEM

				Power / Solution (mW)		Energy / Solution (µJ)	
Problem Size	Cycles	Runtime (ms)	Solutions Found	Total	Dynamic	Total	Dynamic
6 Vertex	500	2.24	11	129.091	13.318	289.164	29.830
	1000	4.248	14	101.429	10.464	430.869	44.449
	1165	4.887	15	94.667	9.767	462.636	47.726
10 Vertex	500	4.833	25	63.680	12.908	307.765	62.381
	1000	9.338	38	41.895	8.492	391.213	79.295
	1475	13.252	40	39.800	8.068	527.430	106.905

In Table V, we show a summary table that relates power and energy consumtion to the number of unique solutions determined in a given scenario. Results show that dynamic power per solution falls between 8 and 14 mW, while dynamic energy per solution falls between 29 and 107 µJ. The lowest dynamic energy per solution in these GCPs is generated when

runtime is at a minimum, even though this is the case that produces the fewest unique solutions. This further demonstrates the flexibility in the optimization of this system where two modes of operation could be used: low enegy for a small solution set or higher energy for a larger solution set.

VI. Discussion

The results in this paper show that in terms of energy per solution, a parallel spiking SNN based SAT algorithm implemented on manycore neuromorphic hardware is an attractive solution for low power, portable AI. One of the key sources of power consumption in the Loihi system utilized is the static power, which is about a constant 1.2 W. However, this static power could be reduced if the Loihi circuit board were optimized for low power operation, where the number of Loihi chips on a circuit board could be tailored to the size of a specific problem.

Table VI provides a simple comparison of the power, time, and energy benefits of placing parallel solvers on a single Loihi chip. For the 6 vertex case it takes about 3.7 ms to execute a single solver, but only 4.25 ms to execute 42 in parallel. For about a 15 % increase in time and an 11 % increase in power, a 42× increase in computation is observed. Similar impact is observed for the 10 vertex case. Given that computation through spike propagation is so efficient in the Loihi system, low dynamic energy results in substantial savings through parallelizing workflow.

TABLE VI. Time, Power, and Energy Comparison for the Proposed Spiking SAT System When Executing Only 1 Solver Instead of 42 on a Loihi Chip

	6 Vertex		10 Vertex	
	1 Solver	**42 Solvers**	**1 Solver**	**42 Solvers**
Time	3.689 ms	4.248 ms	7.710 ms	9.338 ms
Power	1.277 W	1.420 W	1.277 W	1.592 W
Energy	4.711 mJ	6.032 mJ	9.846 mJ	14.866 mJ

In general, the work in this paper shows that this high efficiency solution set finding technique is feasible. The next step will be to apply this method to larger problems with more dynamics and real world applicability when compared to the GCP research exercise. In many more complex problems, a valid solution set vs. a single solution can be extremely useful. For example, in a scheduling or resource allocation problem, the user may want to work in tandem with the computing hardware to determine the best solution. In this case, a set of valid solutions will be returned and the user can then pick the one that makes the most sense. Generating more than one solution would also be beneficial for these types of planning problems if the first choice was invalidated by unforeseen consequences and a backup plan needed to be determined quickly. Providing multiple solutions allows for a very fast backup plan validity check.

Another common decision making technique that requires multiple solutions would rank all possible solutions according to some mathematically determined objective function [25-26]. The GCP example may not show the immediate necessity for a parallel solution finder, but it has served as a demonstrative example to show that multiple solutions can be obtained. Moving forward, we will apply this system to a variety of problems with solution sets that have a quality rank associated with them. Then we will be able to show that solution quality can be improved when ranking multiple solutions vs. simply moving forward with the first solution returned regardless of rank.

While we perform graph coloring in this work, the data input to this solver is in a highly generalized CNF format. Any problem described in conjunctive normal form can be applied to this system with zero hardware modification. The only foreseeable limit is the number of neurons available on the Loihi chip, and we plan to investigate these problem size limits in future work.

VII. Conclusion

This paper presents spiking neural network based algorithms for performing low energy, portable constraint satisfaction problems. We utilize the Loihi spiking neural network hardware to show this SNN SAT algorithm can be parallelized across a single Loihi chip to gain substantial efficiency. Through the execution of two different GCP scenarios, we show that the proposed SAT algorithm is capable of executing these problems with dynamic energies per solution between 29 and 107 μJ.

In the future, we need to compare our parallel SNN SAT approach to SAT hardware such as that developed using FPGAs [27], or embedded low power hardware that is carrying out constraint satisfaction. This will help us better quantify the impact SNN based hardware has in the AI field.

Acknowledgment

This work was performed with the support of the Air Force Research Laboratory.

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] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, and S. Douglass, "Solving Constraint Satisfaction Problems Using the Loihi Spiking

Neuromorphic Processor," IEEE Design, Automation and Test in Europe, 2020 (Accepted).

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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

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

[24] 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.

[25] C. Yakopcic, T. Atahary, T. M. Taha, A. Beigh, and S. Douglass, "High Speed Approximate Cognitive Domain Ontologies for Asset Allocation based on Isolated Spiking Neurons," IEEE National Aerospace and Electronics Conference (NAECON), pp. 241-246, Dayton, OH, July, 2018.

[26] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, A. Beigh, and S. Douglass, "High Speed Approximate Cognitive Domain Ontologies for Constrained Asset Allocation based on Spiking Neurons," IEEE National Aerospace and Electronics Conference (NAECON), Dayton, OH, July, 2019

[27] 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.