Middle East Technical University
Multimedia Informatics Department

MMI 713

PROJECT PROPOSAL REPORT

Solving N-Queens Problem Using Cell Assemblies on GPU

Ugurcan Cakal

March 25, 2020

Contents

1 Problem Definition

The n-queens problem is a well-known constraint satisfaction problem in the field of artificial intelligence. Kruse et al. define the problem as "The problem consists in the task to place n queens (a piece in the game of chess) of the same color onto an n x n chessboard in such a way that no rank (chess term for row), no file (chess term for column) and no diagonal contains more than one queen"[1, p. 181]. There are plenty of solutions for the problem in literature, some of which pointed out in Section 3.3. Although they aim to present the most computationally elegant solution, no of them target the issue in a way to contribute to brain research. I believe it is crucial to model and explore neuroscientific theories in silico in order to obtain yet unasked questions. Therefore this project aims to solve the n-queen problem using a computational cell assembly model.

In 1949, Hebb named the simplest instance of a representative process as Cell Assembly [2, p. 60]. 50 years later, Huyck has started developing a computational cell assembly model to investigate natural neural functioning[3]. Through this model, it has been possible to study human cognition using computers in a limited manner, as detailed in Section 3.2. The sequential nature of the Von Neumann architecture has been the main obstacle in front of the development of the CA model. Massively parallel computation regime of the human brain requires a parallel architecture to mimic itself efficiently. Thanks to the developments in the field of neuromorphic computing, as proposed in 1990 by Mead[4], it has been possible to use spiking neurons to develop biologically plausible networks. There has been implemented neuromorphic computers like SpiNNaker[5], BrainScaleS[6], TrueNorth[7], Loihi[8] and Neurogrid[9]. Currently, Huyck and his team have been simulating CAs on NEST[10] and emulating on SpiNNaker using PyNN[11] as a middleware[12].

Even though neuromorphic computers are designed to be the best-suited hardware for spiking neural networks simulations, reliable and abundant proofs are required to validate their superiority over the other available architectures. Considering the massively parallel nature of the GPUs and neuromorphic computers, they could be seen as opponent processors. Therefore implementing the same application on both could provide us with a suitable comparison medium. Since the whole idea is beyond the scope of the course, this project does not focus on the neuromorphic computing and comparison part, but only on efficient implementation on GPU. The rest could be considered as future work.

2 Literature Survey

The task proposed in this proposal requires implementing a solution to a constraint satisfaction problem using cell assemblies. A similar approach could be seen in the work done by Kreivenas and Huyck in 2018, a way to implement rules using cell assemblies in [13]. The knowledge base in classical cognitive architectures like ACT-R [14], Soar [15], and EPIC [16] inspire the working principle of the project. The way for solution finding is creating a knowledge base depending on already known rules and extending by deducing new ones at each step. The rules are kept in the form of finite state automata [17] represented by CAs. In order to represent finite state automata in the form of CA, rules are translated to neurons. Through this model, it has been possible to solve the Tower of Hanoi problem using spiking neuron based cell assemblies.

Gabriel Andrés Fonseca Guerra, a Ph.D. student of Dr. Steve Furber, has developed a constraint satisfaction problem solver for sudoku, map coloring, and ising spin system problems using stochastic spiking neural networks on SpiNNaker [18]. This work uses a different approach to the problem; however, one of the few applications targeting the CSPs using spiking neurons.

Apart from the pure neuromorphic computing solutions that are referred above, there is a substantial effort to compute on GPU using spiking neural networks. For example, in [19], an efficient Izhikevich neuron [20] based large-scale SNN simulator that runs on a single GPU has been implemented. In [21], the ongoing efforts towards the parallel simulation of the spiking neural network have reviewed, and the main difficulties have outlined. Moreover, the leading simulators of SNNs like NEST[10], Brian[22], and Neuron[23] are compatible with both GPU and CPU simulations. Although they are not as popular as the above mentioned ones, GPU optimized spiking neural network simulators also exist [24], [25].

Last but not least, Vempaty pointed out a way to model 3-queens problem using finite state automata in 1992 theoretically[26]. This model could be scaled out to more dimensions to obtain an n-queens solution by defining the n. However, this solution would not be a perfect way if n is not constant because it focuses on the implementation of static states. For a variable n solution, another approach is necessary.

Taking the effort to implement rules with artificial neurons, CSP solutions using SNNs, and spiking neuron simulations on GPUs into consideration, this proposal aims to combine all these and produce a novel application.

3 Approach for the Solution

Building a neural cognitive architecture starts from building the analog of the smallest computational unit of the brain. Since the smallest computational unit in the brain is the neuron, a proper neuron model is needed to be chosen at first. Commonly used spiking neuron models are Izhikevich [20], leaky integrate and fire(LIF) [27], and Hodgkin&Huxley [28] model. Even though Izhikevich and Hodgkin&Huxley models are neuro-physically more meaningful, leaky integrate and fire model is more hardware friendly since it requires less computational resources. In this project, a modified version of the LIF model, fatigue leaky integrate and fire(FLIF)[29] model will be used because that Kaplan et al. have proposed FLIF as a neuron model for cell assembly construction.

Huyck states that cell assemblies are sets of neurons that may be spatially distributed but that have high mutual inter-connectivity [12]. In order to build finite state automata out of cell assembly, a binary decision mechanism need to be made. Thus, a CA needs to ignite or stay idle and keep the state until a transitive effect has changed the state. This is called binary CA, and Kreivenas and Huyck use 8 excitatory and 2 inhibitory neurons to form a binary CA for rule implementation [13].

Binary CAs can have inhibitory or excitatory effects over the other binary CAs. Building a finite state machine can be possible through exciting the resulting state and inhibiting causing state. Cell assemblies are theoretically capable of programming a finite state machine because Turing completeness of the computational cell assembly model has been proved in [30].

In the n-queens problem case, there is a finite number of allowed placements for each initiation. Each state in the finite state machine represents an instant view of the chess board. Queen replacements are expressed by standard chess board encoding like A1 B2 C4 D4. In this case, a finite state machine accepting allowed states and rejecting prohibited ones can be built.

To sum up, fatigue leaky integrate and fire neurons constitutes binary cell assemblies; binary cell assemblies act as finite state automata; the finite state automata solves the n-queens problem for a predefined n. The following sections includes detailed explanations of the modules.

3.1 Fatigue Leaky Integrate and Fire Neurons

fLIF neuron model is an idealized model of biological neurons. Similar to biological neurons, an fLIF neuron may have many inputs and one output. At a given timestep, if the weighted cumulative sum of all inputs make the activation level of the neuron passes a certain threshold, then the neuron generates a binary 1 output. While accumulation goes on, leak component reduces the activation level over time if the neuron does not fire. Moreover, The fatigue component models the mechanism by which repeated firing leads to an increase in the threshold level of activation that a neuron must surpass in order to fire.

Let's say:

- $\phi_{\_}$: Firing flag
- $E_{\_}$: Energy Level
- $F_{\_}$: Fatigue level
- θ : Firing threshold

Then the neuron fires if the difference between energy level and fatigue level is above the threshold [29].

$$\phi_b(t) = \begin{cases} 1 \text{ if } E_b(t) - F_b(t) \geq \theta \\ 0 \text{ otherwise} \end{cases}$$

Let's say:

- d : Decay constant
- $\omega_{\_\_}$: Connection weight
- F^r : Recovery constant
- F^c : Fatigue constant

If the neuron does not fire, one step further energy level is calculated by summing up the decreased current energy level and weighted sum of incoming spikes.In addition, if the current fatigue level is greater than recovery constant, it is reduced by the recovery constant. The fatigue level is limited below by 0. Else if the neuron fires, one step further energy level is calculated by only the weighted sum of incoming spikes. Also, the current fatigue level is enhanced by the fatigue constant [30].

$$\begin{aligned}
&\textbf{if } \phi_b(t) = 0 \ : \\
&E_b(t+1) = \frac{1}{d}E_b(t) + \sum_{a=1}^{n} \omega_{ab} \times \phi_a(t) \\
&F_b(t) = max\{0, F_b(t-1) - F^r\}
\end{aligned}$$

$$\begin{aligned}
&\textbf{if } \phi_b(t) = 1 \ : \\
&E_b(t+1) = \sum_{a=1}^{n} \omega_{ab} \times \phi_a(t) \\
&F_b(t) = F_b(t-1) + F^c
\end{aligned}$$

Let's say:

- $\alpha \in [0, 1]$: learning rate
- W_B : constant representing average total synaptic strength.
- W_i : current total synaptic strength
- x_t : pre-synaptic firing flag
- y_t : post-synaptic firing flag

The connection weights are adapted at each step using the Hebbian Learning rule. This means that if two neurons fire together, they wire together [2]. On the other side, if two neurons rarely fire together, then their connection strength weakens. The constant representing average total synaptic strength and current total synaptic strength is used to redistribute the synaptic weights and normalize the process in the case that a synapse dies [31].

$$\Delta\omega_{ij} = \begin{cases} \alpha(1-\omega_{ij})e^{W_B - W_i} & \text{if } x_t = 1, \ y_t = 1 \\ -\alpha\omega_{ij}e^{W_i - W_B} & \text{if } x_t = 1, \ y_t = 0 \end{cases}$$

3.2 Binary Cell Assemblies

Binary cell assembly model consist of 10 neurons, 8 having excitatory connections and 2 inhibitory connections. In a neural network, a group of neuron could be considered as cell assembly provided that they have a high mutual synaptic connectivity. This means that, neurons in a CA need to be connected to the most of the other neurons. Also, the neurons in the CA need to be able to sustain activation in the absence of external input by exciting each other. This situation is called ignition and substitute for binary 1. Figure 1 visualizes a binary CA.

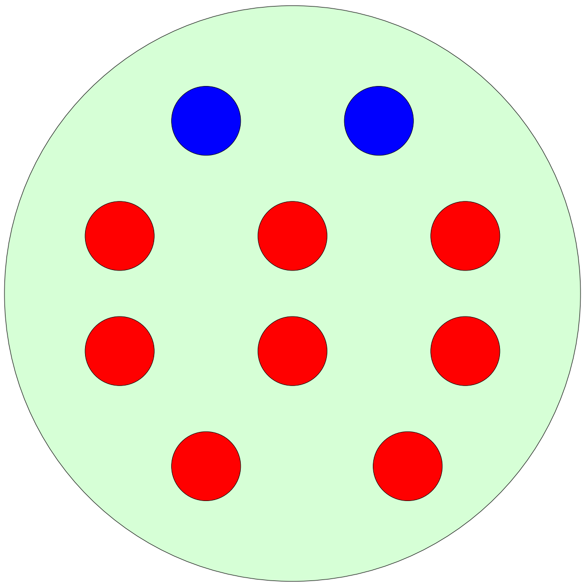

Figure 1: Binary CA (A simplified version of an actual CA in the brain) with blue circles representing inhibitory neurons and red ones representing excitatory neurons

Lets say:

- A : CA
- $\phi_{\_}$: Firing flag
- |A| : number of neurons in A
- X_A : activation threshold for A
- $t_{\_}$: time step
- a : neuron

- $E_{\_}$: Internal energy level
- θ : Firing threshold

For a CA to be ignited, the average spiking activity is needed to be above a certain activation threshold. Also, the ignition needs to stay steady for a measurable time duration[30].

$$
\begin{aligned}
&X_A \leq \frac{\sum_{a \in A} \phi_a(t)}{|A|} \\
&\text{such that } \forall t_j \in [t, t+p] : \\
&\exists a \in A \text{ s.t. } E_a^{internal}(t_j) > \theta
\end{aligned}
$$

3.3 N-Queens Problem

N-queens problem refers the problem of placing n different queens on a nxn chessboard such that no queen attacks any other (A queen attacks any piece in the same row, column or diagonal). Figure 2 show a solution to the 8-queens problem.

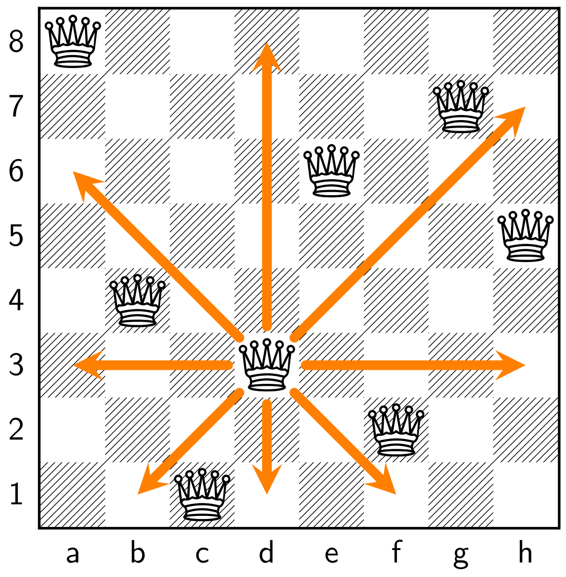

Figure 2: 8-Queens Problem

N-queens problem is a classical AI problem, and there are many different approaches to the problem. For example, Russel and Norvig discuss search algorithms for the solution of the problem [32]. Kruse et al. are focusing more on the implementation of genetic algorithms for the solution. Besides, there are plenty of open-source solutions available in the literature. The typical path is to define the states, take actions, and test the goal.

In order to formally define a 4-queen problem, a constraint satisfaction triplet Variable, Domain, Constraint can be set. The triplet above has simplified by avoiding stating all the values assigned to the variables. The reader should know that if a variable is not assigned to Q, then it is assigned to N or vice versa.

$$
\begin{aligned}
&V = \{A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4, D1, D2, D3, D4\}\\
&D = \{Q, N\}\\
&C = \{\\
&(A1, B2, C3, D4) : \{(Q, Q, Q, Q)\},\\
&(A1, B2, C4, D3) : \{(Q, Q, Q, Q)\},\\
&(A1, B3, C2, D4) : \{(Q, Q, Q, Q)\},\\
&(A1, B4, C2, D3) : \{(Q, Q, Q, Q)\},\\
&(A1, B3, C4, D2) : \{(Q, Q, Q, Q)\},\\
&(A1, B4, C3, D2) : \{(Q, Q, Q, Q)\},\\
&(A2, B1, C3, D4) : \{(Q, Q, Q, Q)\},\\
&(A2, B1, C4, D3) : \{(Q, Q, Q, Q)\},\\
&(A3, B1, C2, D4) : \{(Q, Q, Q, Q)\},\\
&(A4, B1, C2, D3) : \{(Q, Q, Q, Q)\},\\
&(A3, B1, C4, D2) : \{(Q, Q, Q, Q)\},\\
&(A4, B1, C3, D2) : \{(Q, Q, Q, Q)\},\\
&(A2, B3, C1, D4) : \{(Q, Q, Q, Q)\},\\
&(A2, B4, C1, D3) : \{(Q, Q, Q, Q)\},\\
&(A3, B2, C1, D4) : \{(Q, Q, Q, Q)\},\\
&(A4, B2, C1, D3) : \{(Q, Q, Q, Q)\},\\
&(A3, B4, C1, D2) : \{(Q, Q, Q, Q)\},\\
&(A4, B3, C1, D2) : \{(Q, Q, Q, Q)\},\\
&(A2, B3, C4, D1) : \{(Q, Q, Q, Q)\},\\
&(A2, B4, C3, D1) : \{(Q, Q, Q, Q)\},
\end{aligned}
$$

$(A3, B2, C4, D1) : \{(Q, Q, Q, Q)\},$
$(A4, B2, C3, D1) : \{(Q, Q, Q, Q)\},$
$(A3, B4, C2, D1) : \{(Q, Q, Q, Q)\},$
$(A4, B3, C2, D1) : \{(Q, Q, Q, Q)\},$
$\}$

The formulation above characterize the constraint explicitly. However, for the sake of a scalable implementation, the constraint can be specified implicitly by a function. First of all, the chessboard could be seen as a 1-D vector since only one queen on the row or column is allowed. This 1-D vector may further be processed. For example, if we see the variable pattern as a code, such as (A2, B3, C1, D4) = 2314. Then we could define a function to check if the code is composed of unique characters or not since it is the constraint that we need to satisfy. If required, we could calculate the distance from the closest solution.

At first glance, it looks like we could approach to the problem by representing the code using cell asseblies. To do this, we could use a binary coded decimal system as such 2 3 1 4 = 010 011 001 100. We could decide that ignition represents 1 and otherwise 0. And define 3 cell assemblies makes a group.

In case there is a queen on A2, there are multiple solutions like 2134, 2143, 2314, 2341, 2413, 2431. Therefore the constraints could be implemented by inhibitory synapses so (A1,B1,C1,and D1) were mutually inhibitory. Those not inhibited could have smallish excitatory connections. The controller could then turn on a CA (say A2), and it would spread to B4,C1 and D3. However, if the system turn on A1, three other CAs couldn't come on.

4 Acknowledgments

The author would like to thank Dr. Chris Huyck for helpful discussions and valuable suggestions.

5 References

[1] R. Kruse, C. Borgelt, C. Braune, S. Mostaghim, and M. Steinbrecher, *Computational intelligence: a methodological introduction.* Springer, 2016.

[2] D. O. Hebb, *The organization of behavior: a neuropsychological theory.* J. Wiley; Chapman & Hall, 1949.

[3] C. Huyck, "Modelling cell assemblies," in *Proceedings of the International Conference of Artificial Intelligence Las Vegas, Nevada*, 1999.

[4] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct 1990.

[5] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, Dec 2013.

[6] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 1947–1950.

[7] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.

[8] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[9] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[10] M. Gewaltig and M. Diesmann, "NEST (NEural Simulation Tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007, revision #130182.

[11] A. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, vol. 2, p. 11, 2009. [Online]. Available: https://www.frontiersin.org/article/10.3389/neuro.11.011.2008

[12] C. R. Huyck, "A neural cognitive architecture," *Cognitive Systems Research*, vol. 59, pp. 171 – 178, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389041719304899

[13] C. Huyck and D. Kreivenas, "Implementing rules with artificial neurons," in *Artificial Intelligence XXXV*, M. Bramer and M. Petridis, Eds. Cham: Springer International Publishing, 2018, pp. 21–33.

[14] J. R. Anderson and C. Lebiere, "The atomic components of thought." 1998.

[15] J. E. Laird, A. Newell, and P. S. Rosenbloom, "Soar: An architecture for general intelligence." STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1986.

[16] D. E. Kieras, S. D. Wood, and D. E. Meyer, "Predictive engineering models based on the epic architecture for a multimodal high-performance human-computer interaction task," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 4, no. 3, pp. 230–275, 1997.

[17] H. R. Lewis and C. H. Papadimitriou, "Elements of the theory of computation," *ACM SIGACT News*, vol. 29, no. 3, pp. 62–78, 1998.

[18] G. A. Fonseca Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers in Neuroscience*, vol. 11, p. 714, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2017.00714

[19] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. Veidenbaum, "Efficient simulation of large-scale spiking neural networks using cuda graphics processors," in *2009 International Joint Conference on Neural Networks*, June 2009, pp. 2145–2152.

[20] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

[21] R. Brette and D. F. Goodman, "Simulating spiking neural networks on gpu," *Network: Computation in Neural Systems*, vol. 23, no. 4, pp. 167–182, 2012.

[22] D. F. Goodman and R. Brette, “The brian simulator,” *Frontiers in neuroscience*, vol. 3, p. 26, 2009.

[23] N. T. Carnevale and M. L. Hines, *The NEURON book.* Cambridge University Press, 2006.

[24] N. Ahmad, J. B. Isbister, T. S. C. Smithe, and S. M. Stringer, “Spike: A gpu optimised spiking neural network simulator,” *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/11/05/461160

[25] T.-S. Chou, H. J. Kashyap, J. Xing, S. Listopad, E. L. Rounds, M. Beyeler, N. Dutt, and J. L. Krichmar, “Carlsim 4: an open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters,” in *2018 International Joint Conference on Neural Networks (IJCNN).* IEEE, 2018, pp. 1–8.

[26] N. R. Vempaty, “Solving constraint satisfaction problems using finite state automata.” in *AAAI*, 1992, pp. 453–458.

[27] W. Maass and C. M. Bishop, *Pulsed neural networks.* MIT press, 2001.

[28] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764

[29] S. Kaplan, M. Sonntag, and E. Chown, “Tracing recurrent activity in cognitive elements (trace): A model of temporal dynamics in a cell assembly,” *Connection Science*, vol. 3, no. 2, pp. 179–206, 1991.

[30] E. Byrne and C. Huyck, “Processing with cell assemblies,” *Neurocomputing*, vol. 74, no. 1, pp. 76 – 83, 2010, artificial Brains. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231210002572

[31] R. V. Belavkin and C. R. Huyck, “Conflict resolution and learning probability matching in a neural cell-assembly architecture,” *Cognitive Systems Research*, vol. 12, no. 2, pp. 93 – 101, 2011, the 9th International Conference on Cognitive Modeling. Manchester, UK, July 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389041710000598

[32] R. Stuart and N. Peter, “Artificial intelligence: A modern approach,” 2016.