A Design Methodology for Fault-Tolerant Computing using Astrocyte Neural Networks

Murat Isik*, mc138@drexel.edu
Drexel University
Philadelphia, PA, USA

Ankita Paul*, ap3737@drexel.edu
Drexel University
Philadelphia, PA, USA

M. Lakshmi Varshika*, lm3486@drexel.edu
Drexel University
Philadelphia, PA, USA

Anup Das
anup.das@drexel.edu
Drexel University
Philadelphia, PA, USA

ABSTRACT

We propose a design methodology to facilitate fault tolerance of deep learning models. First, we implement a many-core fault-tolerant neuromorphic hardware design, where neuron and synapse circuitries in each neuromorphic core are enclosed with astrocyte circuitries, the star-shaped glial cells of the brain that facilitate self-repair by restoring the spike firing frequency of a failed neuron using a closed-loop retrograde feedback signal. Next, we introduce astrocytes in a deep learning model to achieve the required degree of tolerance to hardware faults. Finally, we use a system software to partition the astrocyte-enabled model into clusters and implement them on the proposed fault-tolerant neuromorphic design. We evaluate this design methodology using seven deep learning inference models and show that it is both area- and power-efficient.

KEYWORDS

astrocyte, neuromorphic computing, fault tolerance

ACM Reference Format:
Murat Isik, Ankita Paul, M. Lakshmi Varshika, and Anup Das. 2022. A Design Methodology for Fault-Tolerant Computing using Astrocyte Neural Networks. In CF’22: ACM International Conference on Computing Frontiers, May 17–19, 2022, Turin, Piedmont, Italy. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Modern embedded systems are embracing neuromorphic devices to implement spiking-based deep learning inference applications [3]. A neuromorphic device is designed as a many-core hardware, where each core consists of silicon circuitries to implement neurons and synapses [18]. Although technology scaling has provided a steady increase of performance, increased power densities (hence temperatures) and other scaling effects create an adverse impact on the reliability by increasing the likelihood of transient, intermittent, and permanent faults in the neuron and synapse circuitries [13, 14]. Hardware faults introduce errors in a trained deep learning model implemented on those circuitries, compromising inference quality (assessed using the accuracy metric). Therefore, providing fault tolerance is a critical requirement for neuromorphic devices.

Recent efforts to this end include software solutions such as model replication [9] and error prediction coding [7], and hardware solutions such as approximation [12] and redundant mapping [20]. For FPGA-based neuromorphic designs, fault tolerance can also be addressed using periodic scrubbing [11, 19]. In this work, we propose a complimentary approach to fault tolerance. We exploit the self-repair capability of the brain, which copes with damaged neurons using astrocytes, the star-shaped glial cells of the brain [8]. Astrocytes generate an indirect retrograde feedback signal, which helps to restore the spike firing frequency of a failed neuron [6].

We propose a design methodology for fault-tolerant neuromorphic computing, which consists of the following three components.

1. We propose a many-core neuromorphic design where neurons in each core are enclosed with astrocytes to facilitate self-repair of errors caused by logic and memory faults.
2. We introduce astrocytes in a deep learning model to achieve a desired degree of tolerance to hardware faults.
3. We propose a system software to partition an astrocyte-enabled inference model into clusters and implement them on the proposed fault-tolerant neuromorphic cores of the hardware.

We evaluate our design methodology using seven deep learning inference models. Results show that the proposed design methodology is both area- and power-efficient, yet providing a high degrees of fault tolerance to randomly injected faults.
2 ASTROCYTE NEURAL NETWORKS

Figure 1 illustrates how an astrocyte regulates the neuronal activity at a synaptic site using a closed-loop feedback mechanism.

Astrocyte causes a transient increase of intracellular calcium (Ca$^{2+}$) levels, which serves as the catalyst for self-repair. Ca$^{2+}$-induced CICR (calcium release via the CICR mechanism) is the main mechanism to regulate Ca$^{2+}$ in the healthy brain. CICR is triggered by inosital 1,4,5-triphosphate (IP$_3$), which is produced upon astrocyte activation. To describe the operation of the astrocyte, let $\delta(t-\tau)$ be a spike at time $\tau$ from the neuron $n$. This spike triggers the release of 2-arachidonyl glycerol (2-AG), a type of endocannabinoid responsible for stimulating the cytosolic calcium Ca$^{2+}$ (cyt). The quantity of 2-AG produced is governed by the ordinary differential equation (ODE)

$$\frac{dAG}{dt} = -AG + r_{AG} \cdot \delta(t-\tau),$$

where $AG$ is the quantity of 2-AG, $r_{AG}$ is the rate of decay and $r_{AG}$ is the rate of production of 2-AG.

The intracellular astrocytic calcium dynamics control the glutamate (Glu) release from the astrocyte. This is governed by the ODE

$$\frac{dGlu}{dt} = -Glu + t_{RGlutam} (t-t_{Glutam}) ,$$

where $t_{RGlutam}$ is the rate of decay and $t_{Glutam}$ is the rate of production of glutamate.

The gluta-teme generates e-SP, the indirect signal to the synaptic site. e-SP is related to Glu using the following ODE

$$\frac{deSP}{dt} = -eSP + t_{Res} m_{eSP} Glut(t) ,$$

where $t_{Res}$ is the decay rate of e-SP and $m_{eSP}$ is a scaling factor.

Finally, there exists a direct signaling pathway (DSE) from neuron $n_i$ to the synaptic site. The DSE is given by

$$DSE = -K_{AG} \cdot AG(t) ,$$

where $K_{AG}$ is a constant. Overall, the synaptic transmission probability (PR) at the synaptic site is

$$PR(t) = PR(0) + \frac{DSE(t) + eSP(t)}{100} ,$$

In the brain, each astrocyte encloses multiple synapses connected to a neuron. Figure 2a shows an original network of neurons, while Figure 2b shows these neurons enclosed using an astrocyte.

3 PROPOSED DESIGN METHODOLOGY

3.1 Novel Hardware With Astrocyte Circuitries

Figure 4 shows the architecture of a many-core neuromorphic hardware (left sub-figure). We take the example of two recent designs – DYNAPs [5], where each core consists of an N x N crossbar with N pre-synaptic neurons connected to N post-synaptic neurons (middle sub-figure), and $\mu$Brain [18], where each core consists of neurons that are organized in three layers with $N$ neurons in layer 1, $M$ neurons in layer 2, and $P$ neurons in layer 3 (right sub-figure).

Figure 5 illustrates our proposed changes to a baseline crossbar (left sub-figure) and a baseline $\mu$Brain (right sub-figure) design.
3.2 Software Mapping Framework
A single neuromorphic core can implement only a limited number of neurons and synapses. A 128 × 128 crossbar core consists of 128 input and 128 output neurons, while a µBrain core consists of 256 neurons in layer 1, 64 neurons in layer 2, and 16 neurons in layer 3. We use a distance-based heuristic [18] to partition an inference model into clusters, where each cluster can be implemented on a core of the hardware. It sorts all neurons of a model based on their distances from output neurons. For µBrain (crossbar) mapping, it groups all neurons with distance less than or equal to 2 (1) into clusters considering the resource constraint of a core. In the next iteration, it removes already clustered neurons from the model, recalculates neuron distances, and groups remaining neurons to generate the next set of clusters. The process is repeated until all neurons are clustered. By incorporating hardware constraints, we ensure that a cluster can fit onto the target core architecture.

3.3 Astrocyte-Enabled Inference Model
We introduce the following notations.

\[
\begin{align*}
G_M(C, E) &= \text{Inference model with } C \text{ clusters and } E \text{ edges} \\
G_A(C_A, E) &= \text{Astrocyte-enabled model with } C_A \text{ clusters and } E \text{ edges} \\
L &= \text{Layers of a core. } L = \{L_x, L_y\} \text{ (crossbar) and } L = \{L_x, L_y, L_z\} \text{ (µBrain)}
\end{align*}
\]

Algorithm 1 shows the pseudo-code to insert astrocytes in clusters of an inference model \(G_M\). First, it organizes the neurons of a cluster into two (for crossbar) or three (for µBrain) layers (line 2). Next, for each layer it uses the ARES framework [10] to insert \(N_r\) random errors, one at a time and record the corresponding accuracy (line 5). If the minimum accuracy \(a_{\text{min}}\) is lower than a threshold \(a_{\text{thr}}\), it adds an astrocyte to the layer (lines 6-8). Otherwise, it exits and analyzes the next layer (lines 8-9). In allocating astrocytes to a layer, if more than one astrocytes are needed, then its distributes neurons of the layer equally amongst the astrocytes. \(N_r\) and \(a_{\text{thr}}\) are user-defined parameters and they are empirically set to 10,000 and \(a_{\text{thr}}\), respectively, where \(a_{\text{thr}}\) is the baseline accuracy of the model without error. Finally, the astrocyte-enabled model \(G_A\) is returned.

**Algorithm 1: Inserting astrocytes in clusters of a model.**

```plaintext
Input: \(G_M = (C, E)\)  
Output: \(G_A = (C_A, E)\)  
for \(C_k \in C\) do  
    \(C_k = (C^C_k \cup k, C^E_k)\); /* arrange neurons & synapses of \(C_k\) into three layers for µBrain core. For crossbar mapping, \(C_k = (C^C_k, C^E_k)\). */  
    for \(C^C_k \in C^C_k\) do  
        while (true) do  
            /* Run until all neurons of the layer are protected against randomly injected errors */  
            Insert \(N_r\) random errors using ARES and evaluate the minimum accuracy \(a_{\text{min}}\);  
            if \(a_{\text{min}} < a_{\text{thr}}\) then  
                /* Min accuracy is less than threshold. */  
                \(C_k = C_k \cup k\); /* Add an astrocyte. */  
            else  
                exit  
        end  
    end  
end  
```

4 EVALUATION
Our simulation framework consists of the following.
- QKeras: to train 2-bit quantized deep learning models.
- PyCARL [1]: to generate spiking inference models.

\[
\begin{align*}
\text{Algorithm 1: Inserting astrocytes in clusters of a model.}
\end{align*}
\]

4.1 Astrocyte Area and Power
We implemented the astrocyte design, the baseline µBrain and crossbar designs on Xilinx VCU128 development board (see Table 1). We observe that although an astrocyte circuitry is smaller than the size of a µBrain (336 neurons) and a crossbar (256 neurons), it is in fact, significantly larger and consumes significantly higher power than a single neuron circuitry. Furthermore, an astrocyte circuitry uses more flip-flops (FF), slices, and lookup tables (LUTs) than the two baseline designs. The higher area of the two baseline designs are due to the usage of block RAMs (BRAMs). The power consumption of an astrocyte design is shown in Figure 6, distributed into clocks, signals, logic, DSP, BRAM, MMCM, and I/O.

**Table 1: Implementation of an astrocyte and the baseline µBrain [18] and crossbar [5] designs on Xilinx VCU128.**

|                        | µBrain [18] | Crossbar [5] | Astrocyte |
|------------------------|-------------|--------------|-----------|
| Neurons                | 336         | 256          | -         |
| Synapses               | 17,408      | 16,384       | -         |
| Operating Frequency    | 100MHz      | 100MHz       | 100MHz    |
| BRAM                   | 48          | 32           | 4         |
| DSP                    | 0           | 0            | 4         |
| FF                     | 129         | 86           | 2,368     |
| Slice                  | 117         | 78           | 670       |
| LUT                    | 114         | 76           | 1,345     |
| FPGA Utilization       | 49%         | 40%          | 12%       |
| Power                  | 4.64W       | 4.53W        | 6.538W    |

**Figure 6: Power consumption of astrocyte, distributed into clocks, signals, BRAMs, DSPs, MMCM, and I/Os.**

4.2 Fault Tolerance
Figure 7 plots the accuracy, normalized to the replication technique, of each evaluated model for 10%, 20%, and 50% of parameters in error. These errors are injected randomly using the ARES framework [10] and the reported results are average of 10 runs. With 10% error rate, there are only a few errors per cluster. Therefore, most errors can be masked by astrocytes that are inserted into each model cluster. So, we see no accuracy drop. With higher error rates, the accuracy is lower. This is because of the increase in parameter errors in each cluster. Errors in multiple neurons of an enclosed astrocyte impact its ability to restore the spike frequency, causing a significant amount of accuracy drop. On average, the accuracy is 23% and 54% lower for error rate of 20% and 50%, respectively.

- Brian 2 [16]: for astrocyte modeling.
- ARES [10]: for fault simulations.
- Xilinx Vivado: for FPGA synthesis.
for each core is calculated based on the static power of the design and the activation of the synaptic weights in the core [17]. We make two key observations. First, power is higher for models such as AlexNet, VGGNet, and Xception due to higher number of model parameters. Second, on average, power using the proposed design methodology is 60% lower than replication technique and 50% lower than redundant mapping technique. For μBrain-based design (not shown here for space limitations), power using the proposed methodology is 60% lower than the replication technique.

5 CONCLUSIONS

We propose a design methodology for fault-tolerant neuromorphic computing. First, we propose a novel design, where a core consists of neuron, synapse, and astrocyte circuitries. Each astrocyte encloses multiple neurons to facilitate self-repair of a failed neuron. Next, we insert astrocytes in an inference model to achieve the desired degree of fault tolerance. Finally, we propose a system software framework to map astrocyte-enabled inference model to the proposed fault-tolerant many-core design. We evaluate the proposed design methodology using several deep learning models on the fault-tolerant implementation of two baseline neuromorphic designs. We show that the proposed design methodology is both area and power-efficient, yet providing similar degrees of fault tolerance compared to existing approaches.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation Faculty Early Career Development Award CCF-1942697 (CAREER: Facilitating Dependable Neuromorphic Computing: Vision, Architecture, and Impact on Programmability).

REFERENCES

[1] A. Balaji et al., “PyCARL: A PyNN interface for hardware-software co-simulation of spiking neural network,” in FJCCN, 2020.
[2] A. Balaji et al., “Mapping spiking neural networks to neuromorphic hardware,” TVLSI 2020.
[3] Y. Cao et al., “Spiking deep convolutional neural networks for energy-efficient object recognition,” TCV, 2015.
[4] P. K. Huynh et al., “Implementing spiking neural networks on neuromorphic architectures: A review,” arXiv, 2022.
[5] S. Moradi et al., “A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs),” TBCAS, 2017.
[6] S. Nadkarni et al., “Modeling synaptic transmission of the tripartite synapse,” Physical Biology, 2007.
[7] S. Park et al., “Low-cost prediction-based fault protection strategy,” in CGO, 2020.
[8] V. Pampur et al., “Glutamate-mediated astrocyte-neuron signalling,” Nature, 1994.
[9] F. Ponzi et al., “E2CNN: Ensembles of convolutional neural networks to improve robustness against memory errors in edge computing devices,” TC, 2021.
[10] B. Reagen et al., “Ares: A framework for quantifying the resilience of deep neural networks,” in DAC, 2018.
[11] R. Santos et al., “Critically-aware scrubbing mechanism for SRAM-based FPGAs,” in FIE, 2014.
[12] A. Siddique et al., “Exploring fault-energy trade-offs in approximate DNN hardware accelerators,” in IRED, 2021.
[13] S. Song et al., “A case for lifetime reliability-aware neuromorphic computing,” in MWSCAS, 2020.
[14] S. Song et al., “Improving dependability of neuromorphic computing with non-volatile memory,” in EDCC, 2020.
[15] S. Song et al., “DFSynthesizer: Dataflow-based synthesis of spiking neural networks to neuromorphic hardware,” TECS, 2021.
[16] M. Stimberg et al., “Modeling neuron–glia interactions with the Brian 2 simulator,” in Computational Glioscience, 2019.
[17] T. Titirsha et al., “On the role of system software in energy management of neuromorphic computing,” in CF, 2021.
[18] M. L. Varshika et al., “Design of many-core big little μBrains for energy-efficient embedded neuromorphic computing,” in DATE, 2022.
[19] S. Venkataraman et al., “A bit-interleaved embedded hamming scheme to correct single-bit and multi-bit upsets for SRAM-based FPGAs,” in FPL, 2014.
[20] G. Yuan et al., “Improving DNN fault tolerance using weight pruning and differential crossbar mapping for ReRAM-based edge AI,” in ISQED, 2021.