Check for updates

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

Murat Isik*
mci38@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.

CCS CONCEPTS

• **Hardware** → **Neural systems**; • **Computer systems organization** → **Dependable and fault-tolerant systems and networks**.

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 *19th ACM International Conference on Computing Frontiers (CF'22), May 17–19, 2022, Torino, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3528416.3530232

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.

- 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.
- We introduce astrocytes in a deep learning model to achieve a desired degree of tolerance to hardware faults.
- 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 Ca^{2+} release (CICR) is the main mechanism to regulate Ca^{2+} in the healthy brain. CICR is triggered by inosital 1,4,5-triphosphate (IP_3),

*Authors contributed equally to this research.

CF'22, May 17–19, 2022, Torino, Italy

ACM ISBN 978-1-4503-9338-6/22/05.
https://doi.org/10.1145/3528416.3530232

which is produced upon astrocyte activation. To describe the operation of the astrocyte, let $\delta(t-\tau)$ be a spike at time τ from the neuron n_i. 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)

$$\frac{dAG}{dt} = \frac{-AG}{\tau_{AG}} + r_{AG} \cdot \delta(t-\tau), \tag{1}$$

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

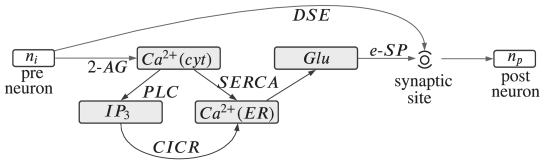


Figure 1: Operation of an astrocyte (gray blocks).

On one pathway, the cytosolic calcium is absorbed by the endoplasmic reticulum (ER) via the Sarco-Endoplasmic-Reticulum Ca^{2+}-ATPase (SERCA) pumps, and on the other pathway, the cytosolic calcium enhances the Phospholipase C (PLC) activation process. This event increases IP_3 production and ER intracellular calcium release via the CICR mechanism.

The intracellular astrocytic calcium dynamics control the glutamate (Glu) release from the astrocyte, which is governed by

$$\frac{dGlu}{dt} = \frac{-Glu}{\tau_{Glu}} + r_{Glu}(t - t_{Ca}), \tag{2}$$

where τ_{Glu} is the rate of decay and r_{Glu} is the rate of production of glutamate, and t_{Ca} is time at which Ca^{2+} crosses the release threshold. The glutamate generates e-SP, the indirect signal to the synaptic site. e-SP is related to Glu using the following ODE

$$\frac{deSP}{dt} = \frac{-eSP}{\tau_{eSP}} + \frac{m_{eSP}}{\tau_{eSP}} Glu(t), \tag{3}$$

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

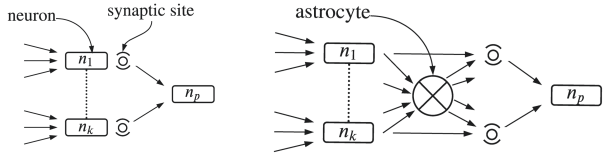
Finally, there exists a direct signaling pathway (DSE) from neuron n_i to the synaptic site. The DSE is given by

$$DSE = -K_{AG} \cdot AG(t), \tag{4}$$

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

$$PR(t) = PR(0) + PR(0)\left(\frac{DSE(t) + eSP(t)}{100}\right) \tag{5}$$

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.



(a) Original network. **(b) Astrocyte-modulated network.**

Figure 2: Inserting an astrocyte in a neural network.

To understand the self-repair mechanism, consider neuron n_i in Fig. 1 fails to fire a spike. Without the astrocyte, the spike firing rate at the synaptic site would decreases. However, because of the astrocyte, 2-AG production reduces (Eq. 1). This increases the DSE (Eq. 4). Therefore, the PR increases (Eq. 5) along with an increase of the spike firing frequency at the synaptic site.

Figure 3 illustrates the self-repair mechanism. The input neuron n_i is excited with Poisson spike events having a mean spike rate of 60Hz. We interrupt the input at around 50 sec. We observe that the firing frequency at the synaptic site connected to n_i drops to 0. This is indicated with the label *output (fault)*. Using astrocyte, the firing frequency can be restored partially as illustrated using the label *output (astrocyte)*.

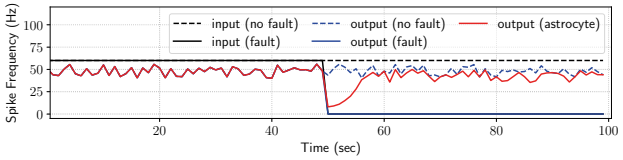


Figure 3: Self-repair mechanism of an astrocyte.

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 $N \times N$ crossbar with N pre-synaptic neurons connected to N post-synaptic neurons (middle sub-figure), and μBrain [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).

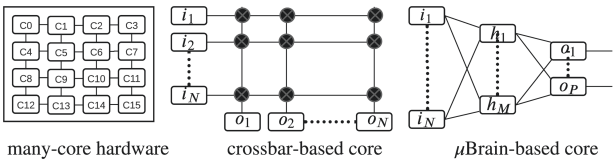


Figure 4: Baseline architecture of a neuromorphic hardware.

Figure 5 illustrates our proposed changes to a baseline crossbar (left sub-figure) and a baseline μBrain (right sub-figure) design.

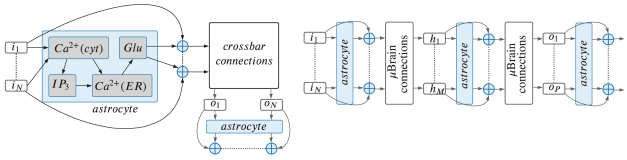


Figure 5: Proposed design of crossbar (left) and μBrain (right).

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.[1] It sorts all neurons of a model based on their

[1] Apart from distance-based heuristic, recently heuristic graph partitioning approaches are also proposed in literature [2, 4, 15].

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.

$G_M(C, E)$ = Inference model with C clusters and E edges

$G_A(C_A, E)$ = Astrocyte-enabled model with C_A clusters and E edges

L = Layers of a core. $L = \{L_x, L_y\}$ (crossbar) and $L = \{L_x, L_y, L_z\}$(μBrain)

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_{min} is lower than a threshold a_{th}, 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_{th} are user defined parameters and they are empirically set to 10,000 and a_o, respectively, where a_o 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.

```
   Input: G_M = (C, E)
   Output: G_A = (C_A, E)
1  for C_k ∈ C do                                   /* For each cluster in C */
2      C_k = {C_k^x, C_k^y, C_k^z};   /* arrange neurons & synapses of C_k into three
         layers for μBrain core. For crossbar mapping, C_k = {C_k^x, C_k^y}. */
3      for C_k^i ∈ C_k do                           /* For each layer in C_k */
4          while (true) do  /* Run until all neurons of the layer are protected
             against randomly injected errors */
5              Insert N_r random errors using ARES and evaluate the minimum accuracy
                 a_min;
6              if a_min < a_th then  /* Min accuracy is less than threshold. */
7                  C_k^i = C_k^i ∪ A;                /* Add an astrocyte. */
8              else
9                  exit;
```

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.
- `Brian 2 [16]`: for astrocyte modeling.
- `ARES [10]`: for fault simulations.
- `Xilinx Vivado`: for FPGA synthesis.

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 use of more 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	0.538 W

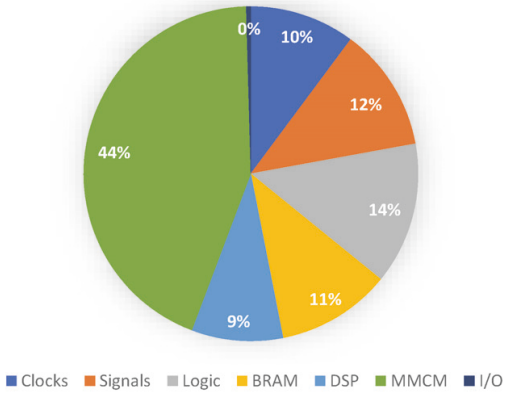


Figure 6: Power consumption of astrocyte, distributed into clocks, signals,BRAMs,DSPs,MMCM, and I/Os.

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.

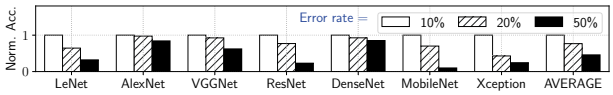


Figure 7: Normalized accuracy for different error rates.

4.3 Design Tradeoffs

Figure 8 shows the area of a μBrain-based design normalized to the replication technique for three error rates – 10%, 20%, and 30%. The accuracy constraint is set as the accuracy without error. This accuracy constraint is achieved for 10% error rate using our baseline

design. So there is no area overhead. For 20% and 50% error rates, more astrocytes are needed to achieve the accuracy constraint. On average, the proposed design requires 28% and 49% higher area for 20% and 50% error rate, respectively.

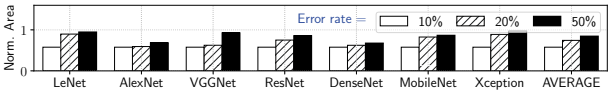


Figure 8: Normalized area for different error rates.

4.4 Model Area

Table 2 reports the design area for each of the evaluated deep learning inference models using 1) model replication technique [9], 2) redundant mapping technique [20], and 3) the proposed design methodology. Design areas are reported for both the μBrain-based core [18] and the crossbar-based core [5]. All results are normalized to the μBrain-based design implementing the LeNet model using the model replication technique. We make three key observations.

Table 2: Design area compared to model replication [9] and redundant mapping [20].

	Model Replication [9]		Redundant Mapping [20]		Proposed Design	
	μBrain	crossbar	μBrain	crossbar	μBrain	crossbar
LeNet	1.0	0.8	–	0.7	0.5	0.4
AlexNet	79.0	68.5	–	54.8	39.2	33.1
VGGNet	62.9	54.6	–	43.7	31.2	26.4
ResNet	1.1	0.9	–	0.8	0.6	0.5
DenseNet	13.5	11.7	–	9.4	6.7	5.7
MobileNet	4.4	3.8	–	3.0	2.2	1.8
Xception	40	34.7	–	27.7	19.9	16.8

First, design area is larger for models with higher number of parameters. This is because models with more parameters require more clusters (cores), which increases the design area. Second, the redundant mapping technique is only applicable to crossbar-based designs. Therefore, results for the μBrain-based design are not provided. Third, for the μBrain-based design, the proposed design methodology results in 50% lower area than the replication technique. For the crossbar-based design, it results in 51.6% lower area than the replication technique and 39.5% lower area than the redundant mapping technique. These improvements are because implementing a few astrocytes in a baseline μBrain and crossbar designs is area-efficient than 1) replicating model clusters, which requires more cores to implement a model, and 2) redundant mapping, which requires larger crossbars to implement each cluster.

4.5 Model Power

Figure 9 reports the power for each evaluated model on a crossbar-based design using the three evaluated approaches. Power numbers 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.

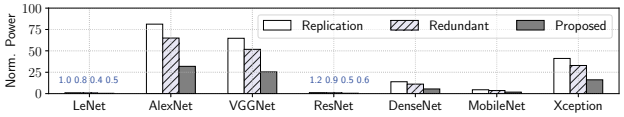


Figure 9: Power consumption.

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 *IJCNN*, 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," *IJCV*, 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. Parpura *et al.*, "Glutamate-mediated astrocyte–neuron signalling," *Nature*, 1994.
[9] F. Ponzina *et al.*, "E2CNNs: 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.*, "Criticality-aware scrubbing mechanism for SRAM-based FPGAs," in *FPL*, 2014.
[12] A. Siddique *et al.*, "Exploring fault-energy trade-offs in approximate DNN hardware accelerators," in *ISQED*, 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.