**Manuscript ID:** TETCSI-2020-12-0437

**Article Title:** "MigSpike: A Migration Based Algorithms and Architecture for Scalable Robust Neuromorphic Systems"

**To:** IEEE Transactions on Emerging Topics in Computing Editors and Reviewers

**Re:** Response to reviewers

Dear Editor and Reviewers,

Thank you for allowing us to revise our manuscript, with an opportunity to address the reviewers' comments.

We are uploading (1) our point-by-point response to the comments (below) (response to reviewers), and (2) an updated manuscript with yellow highlighting indicating changes.

Best regards,

On behalf of the authors

Khanh N. Dang

# **Legend**

|  |  |
| --- | --- |
| **Style** | **Description** |
| Green text | reviewer's comment |
| Black text | author answer |
| Yellow highlight | updated text in the revised manuscript |
| **Underline, purple, bold** | The position of the higlighted part in the revised document |

# **Associate Editor**

Comments to the Author:

Thank you for submitting your work to the Special Issue. Reviewers found the approach reasonable even if they raised several questions regarding the motivation of your research, the implementation details, the evaluation benchmark, the fault model, and the experimental results. Please revise carefully the paper to address all the reviewers' comments.

**Author's response:** We would like to thank the editor for the recommendation. Please find the following our responses to all the reviewers' comments.

# **Reviewer #1:**

**Reviewer #1, Comment #1:**

Some suggestions:

1. Research motivation needs to be more convincing. It is generally believed that compared with DNN, SNN is naturally more fault-tolerant. As the scale increases, errors will be more; but it is generally believed that when the scale of SNN increases, the capability of fault tolerance will improve, too. Therefore, the motivation of this research needs to be elaborated -- not just using a toy-scale example MNIST to illustrate.

**Author's response:** We would like to thank the reviewer for the important comment. We agree on the point that SNN and large-scale SNN have the natural ability to tolerate faults. This has been shown in Figure 1 in the submitted manuscript as a 1% weight error reduces a small amount of overall accuracy. However, we would like to emphasize that the resiliency of SNN is not perfect as accumulated faults can lead to inaccurate output. If the error rate reaches a certain level, the impact can be problematic.

The main reason for performing the MNIST benchmark is that our neuromorphic hardware system currently does not support convolutional computation. Without a convolutional function, it isn't easy to achieve a reasonable accuracy in the CIFAR-level data set. Our neuromorphic systems are designed for bio-plausible applications; therefore, using the MNIST dataset is suitable for understanding the motivations.

However, as we understand the reviewer's concerns, we did adopt the VGG-16 model for CIFAR-10 in [12] in "software-only" to perform the fault injection. This will help address the motivation. As VGG-16 is more complicated than the MNIST-model, we believe this can convince the reviewers and readers about the need for fault-tolerance features in SNN.

[12] N. Rathi, G. Srinivasan, P. Panda, and K. Roy, “Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation,” in International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=B1xSperKvH:

Source code: [https://github.com/nitin-rathi/hybrid-snn-conversion](about:blank)

The fault injection source code is uploaded here:

[https://github.com/khanhdang/hybrid-snn-conversion-with-faults](about:blank)

**Author's action:**

We modified **Fig.1 on page 1** and its related text for illustrating the faults in VGG-16 for CIFAR-10. As shown in Figure 1 (copied below), the accuracy drops can also be observed in VGG-16 for CIFAR10. The accuracy drops are pretty different in the CIFAR-10 case.

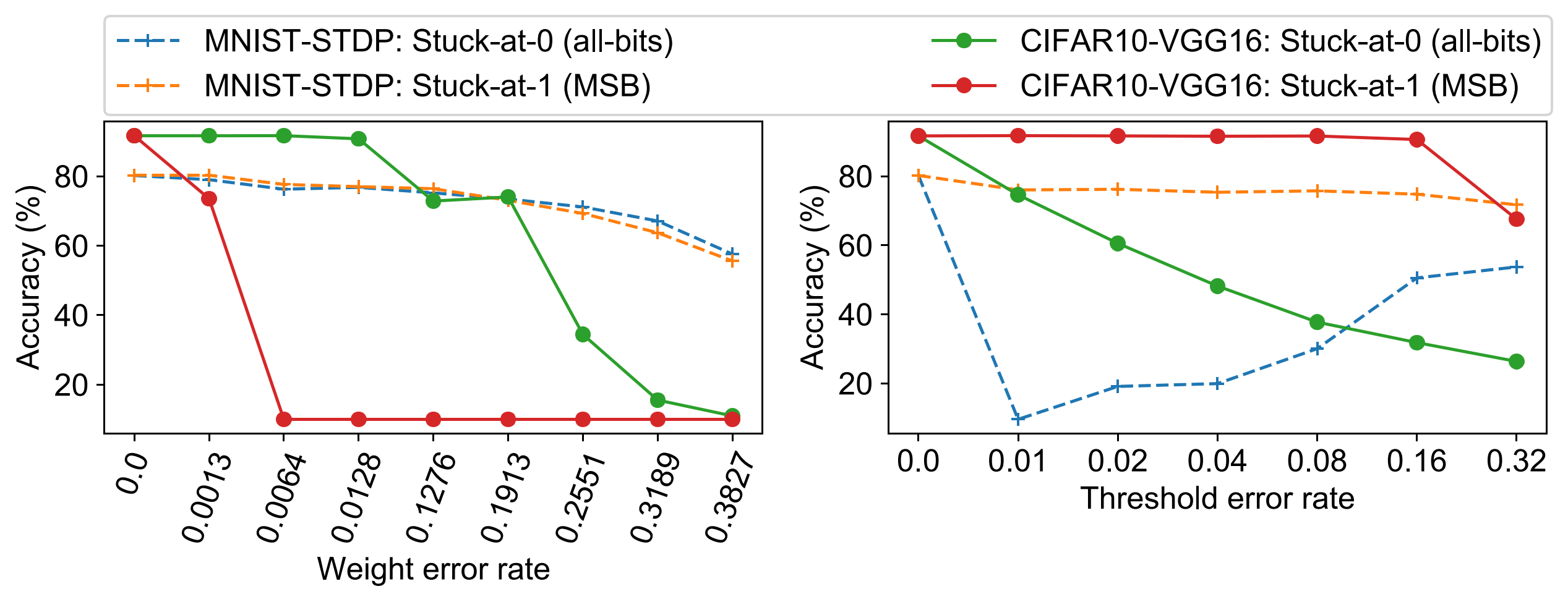


Figure 1: Impact of faults on accuracy of MNIST dataset with unsupervised STDP based SNN size 784:100 and CIFAR-10 with spiking VGG-16.

We also added the text describes the Figure 1 from **lines 47, right column, page 1 to line 30, left column, page 2:**

To understand the impact on SNN, we randomly inserted several faults and tested with the test images in the MNIST and CIFAR10 datasets. For the MNIST dataset, the SNN model is 784:100 with lateral inhibitory connections adopted from [11] and run on BindsNet [8] simulator. This network follows the winner-take-all principle, where a firing neuron inhibits other neurons. The weights are pre-trained using STDP as in [11]. For the CIFAR10 dataset, we adopt the VGG-16 model with the conversion in [12]. Figure 1 illustrates the accuracy drop when inserting faults into the weight SRAM and thresholds. Once we inserted stuck-at-0 or stuck-at-1 faults into the weight memory of the MNIST model, as shown in Figure 1, we can notice that the accuracy drops are ineligible for a small number of faulty weights thanks to the natural fault-resilience of SNNs.

However, when the number of faults increases, the accuracy starts dropping significantly. We can observe a similar behavior with stuck at- 0 in VGG-16 for CIFAR10. However, the stuck-at-1 is critical for VGG-16 as higher weight leads to the domination of some features in the convolution. Once the error rate reaches above 0.0064, the accuracy stuck at 0.1, which means the VGG-16 only outputs one label. On the other hand, a defect in a computation unit is more critical in the MNIST model. A single stuck-at-0 (which is 1% in 100-neurons) on a threshold register can easily make a constant firing neuron, which drops the accuracy significantly to around 10%. This is equal to assigning one label for all testing images of MNIST. Once two or more neurons have stuck-at-0 faults at their threshold registers, two or more neurons start to compete, which increases the overall accuracy; however, the accuracy is still much lower than the non-faulty results. On VGG-16, the stuck-at-0 is a critical problem. The spike can explain this is generated by comparing the membrane potential with a threshold. As the membrane potential accumulates weighted spikes, stuck-at-0 in threshold has similar behaviors as stuck-at-1 in weights. While stuck-at-1 seems not so critical as the accuracy maintained with the error rate in the threshold lesser than 0.16, stuck at-0 in thresholds constantly reduces overall accuracy.

**Reviewer #1, Comment #2:**

1. Some SNN-specific strategies should be highlighted. This paper presents several recovery and migration related algorithms, but which are (is) SNN-specific? Now these technologies look like that they can be used for other distributed processing system's fault tolerance (like DNN) too. More insights are needed.

**Author's response:** This method certainly has the potential to adopt to DNN; however, there are some parts is specific for SNN:

* First, SNN operations rely on low-cost spiking neurons, which are extremely low-cost and can be added as redundancies. In conventional ANN, the computation depends on the *Multiplication-and-Accumulation units* (MAC). Due to the natural complexity of MAC (i.e., floating-point MACs are incredibly high cost), they are not suitable for redundancy-based recovery. Our design uses a LIF neuron, showed in Fig.2, which consisting of several registers, a 16-bit adder (for membrane potential integration), and a 4-bit adder (for refractory counter) two comparators. The design of LIF is a meager cost and can be easily added as spares. Please note that balancing the number of spares and the overall reliability is a **multiple-objective optimization** problem. Solving such a problem is not a trivial task and will be investigated in the future, with multi-objective algorithms such as NSGA-II or SPEA-II. In our design, we currently consider that the spares occupy a certain ratio (20%) as the area overhead is not so critical. In DNN, this however can be a critical issue.
* Second, SNN relies on AER (address-event-representative) to illustrate the firing neuron ID. This is the de facto protocol for the neuromorphic system since Professor Mead, and his team at Caltech designed the first chips. This allows migration in SNN straightforward to handle. After migration, the local firing neuron index is converted to a global AER for transmitting over the system (see Fig. 10). This approach is however not suitable for normal DNN in our experience.

Due to these two critical reasons, we believe the method is well-tailored for SNN. Adapting it into DNN needs more investigations that will be carried out in the future. We also want to point out that our research group only works on SNN currently. Therefore, we do not have a DNN platform to adapt the model. This could be an excellent future research topic for us or other groups.

**Author's action:**

We clarify why this method is more suitable for SNN in the discussion section. We also provide suggestions. The following paragraph is added in **lines 32-41, right column, page 14**:

As the proposed architecture and algorithm based on migrating the neurons, this work might be adapted to other domains (non-spiking NN or distributed computing). However, two key factors allow us to tailor the method for SNN. First, neuron or computing unit of SNN is a meager cost; therefore, adding redundant computing units is not critical as in other computing domains. Second, SNN transmits spikes in AER format where only the global address of the firing neuron is sent. This allows high-flexible mapping as the local firing neuron address is converted into a global one using a LUT, as in Fig. 10.

**Reviewer #1, Comment #3:**

1. This design is only applicable when there is a one-to-one correspondence between application-level neurons and physical neurons? It is explicitly mentioned in the paper that it applies to such "serial systems." But I think the time-division multiplexing of physical neurons is very important to improve the system efficiency, so I want to know whether the current design can be extended to this multiplexing situation.

**Author's response:** Yes, this method works best with "a one-to-one correspondence between application-level neurons and physical neurons." However, there is no problem with "serial systems." Here, we have two assumptions for the serial system:

* **Assumption 1: the computing unit is always healthy in the serial system.** In other words, we only consider faults in the memory units. In this case, the formulation of the problem in Section 4.1 is correct as we initially targeted.
* **Assumption 2: the computing unit can be faulty in the serial system.**  If there are faults in the memory unit, the problem formation is as same as Section 4.1. For defects in the computing units, designers have two options:
  + **Option-1:**  adding physical spare neuron. For instance, add an extra neuron so it can correct. This is similar to *node-level recovery.*
  + **Option-2:** having a spare node, and once the physical neuron fails, all virtual neurons are migrated into the extra node. This is similar to *system-level recovery.* However, it does not have the high granularity as in a one-to-one system.

In both assumptions, our MigSpike platform can deal with a serial system without having significant modification. As the serial system might have one physical neuron per node, the MigSpike needs to run twice: one for weight memory and one for the physical neuron. In the parallel system, it only needs to run once.

In summary, there is no problem with our method in the *serial system.* Designers only need to consider the position of the faults. If it is in the memory unit, it is similar to our method. If it is in the computing unit, it needs a light adjustment. In a one-to-one system, we treat both cases similarly.

**Author's action:**

We clarified how serial system could be adapted to the formulation and algorithms in **lines 20-26, right column, page 5**:

In serial systems [3][37], designers might need to treat memory units and computation units independently. For memory units, it is similar to the above formulation. For faults in computing units, as the physical neuron can be one per node, adding redundancies is in memory and biological neurons.

**Reviewer #1, Comment #4:**

4. This paper only uses MNIST as the data set for evaluation. SNN training algorithm is still in its infancy, but even so, it is not very suitable to take such a toy test as the evaluation benchmark for a hardware design. Existing SNN training studies have supported CIFAR-level test set and spiking-CNN. It is suggested that the authors improve the work in this regard.

**Author's response:** We would like to thank the reviewer for the critical comment. However, we do not support convolutional SNN in the current hardware. Therefore, we do not evaluate with such a model. Unlike the customarily connected model, convolution needs a specific way of loading the inputs and the filter. Also, pooling (max or average) is required.

Convolution function, max-pooling in SNN need more investigations before being implemented in hardware. For instance, convolution requires less weight (as a filter); however, the data to store is enormous. We can also unroll the convolution to compute in a full-connected system; however, the weights are problematic. The second challenge is the pooling (max or average). In ANN, the values are taken immediately for the pooling function. In SNN, we might need to store the whole spike train. Therefore, reducing the latency and memory need to the pooling function is also need.

In [Ref-1], our group initiated one of the first designs on max-pooling for future SCNN.

[Ref-1] "A lightweight Max-Pooling method and architecture for Deep Spiking Convolutional Neural Networks", DA Nguyen, XT Tran, **Khanh N. Dang**, F Iacopi 2020 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 209-212

The algorithm is very suitable to work with CNN, but we can not evaluate it as we do not have a stuitable hardware design. Also, the fully connected model is used as a classifier in the CNN; this work can be used for the part directly.

We already illustrated in **Reviewer #1, Comment #1,** the results on "software only."

Although we can perform on CNN (i.e., VGG-16 with CIFAR-10) in software, we would like to thank the reviewer for this important suggestion. We will investigate and report about CNN in our future publications.

**Author's action:**

We discussed the SCNN in the discussion section in **lines 42-48, right column, page 14**:

We have not evaluated the convolution SNN in the mapping solution in the evaluation section as our hardware architecture does not support convolution and pooling layers. Despite the lack of support for hardware spiking CNNs, the mapping method is still suitable for being applied to CNNs as the problem formulation can be used. The details on tolerating faults in spiking CNNs will be investigated in the future work of MigSpike.

We mentioned SCNN as one of future works in **lines 13-14, left column, page 15**:

Moreover, adapting the algorithm into spiking CNN and a better dataset is another future work.

**Reviewer #1, Comment #5:**

Some minors:

1. The hardware comparison made in Table 2 is of little significance (because of different processes/functions), and removing this does not affect the theme.

2. Some details of the algorithms (such as those related to genetic algorithms) can be moved to the appendix.

3. It is recommended to supplement the mapping related literature, such as: Optimized Mapping Spiking Neural Networks onto Network-on-Chip. ICA3PP 2016.

**Author's response:** We would like to thank the reviewer for these comments. We addressed one by one in the following part

**Author's action:** We revised the manuscript as follows:

1. The hardware comparison made in Table 2 is of little significance (because of different processes/functions), and removing this does not affect the theme.

We agree that the comparison only shows the complexity of the SNN itself and does not reflect the mapping method. We have removed table 2 as the reviewer suggested.

2. Some details of the algorithms (such as those related to genetic algorithms) can be moved to the appendix

We moved the details of GA into **appendix A in lines 16-59, left column, page 15**:

We edited in **lines 9-10, right column, page 9**:

The formulation of GA is shown in Appendix A.

3. It is recommended to supplement the mapping-related literature, such as Optimized Mapping Spiking Neural Networks onto Network-on-Chip. ICA3PP 2016.

Thank you for suggesting important work on mapping SNN into NoC. We cited the paper as reference [33] in **lines 18-21, right column, page 3**:

Since the targeted system is an NoC-based multi-cores one, we can use both the SNN mapping methods [3], [29], [30], [31], [32], [33] and conventional multi-core NoC mapping methods [14] for placing neurons.

[32] Y. Ji, Y. Zhang, H. Liu, and W. Zheng, "Optimized mapping spiking neural networks onto network-on-chip," in International Conference on Algorithms and Architectures for Parallel Processing. Springer, 2016, pp. 38–52.

# **Reviewer #2:**

**Reviewer #2, Comment #1:**

Weaknesses

1. During the system level remapping of neurons, does the proposed neuron migration algorithm consider the possible increase in congestion (and subsdequent increase in latency) as a part of the average spike transmission cost (FCost). Could the author please provide more information (function) on the FCost using in Section 5.3?

**Author's response:** We would like to thank the reviewer for the important comment.

For the max-flow min-cut adaptation, it is not possible to consider the congestion (or latency) in the algorithm as the algorithm only tries to find the best solution with the lowest migration cost.

For the Genetic Algorithm, we did consider the FCost as the second rank selection in S4. In the first submission (you can find in **lines 4-5, right column, page 9**), we wrote:

The communication cost Fcost is also computed for the selection step S4. In S4, it ranks the best B solutions out of K, and if they have similar Mcost values, their Fcost is considered.

In other words, we rank by Mcost first, and if there are two more solutions with the same Mcost, we consider Fcost.

Please note that reducing bot Mcost and Fcost simultaneously converts the problem into a multiple-objective optimization. Our GA is only single-objective optimization. We are currently working toward a multiple-objective optimization approach.

In **Section 5.3 (page 11)**, we use FCost as an evaluation category of all mapping algorithms. As we mentioned about, MFMC does not consider FCost, and GA uses Fcost as the second selection method.

**Reviewer #2, Comment #2:**

2. In the results section the authors compare the execution time of the MFMC algorithm with the runtime algorithms like 1hop and N-hop greedy algorithms. However, the execution time of the proposed runtime algorithm (MFMC) is also compared with the design/program time algorithm proposed in [31].

The design time algorithm is executed on all the neurons in the design. However, the proposed MFMC algorithm only works on remapping the faulty neurons. Hence, is it fair to assume that the 2 optimization algorithms are solving 2 very different problems? Would it then be fair to compare the run-time of the two algorithms given MFMC does not tackle the initial mapping of neurons and synapses?

**Author's response:**  We did compare with Load Balancing and Kernighhan-Lin in [32] (reference [31] in the previous submission) for illustrating the complexity of the algorithm.

That is true that the algorithm in [31] does not target the same problem. We agree with the reviewers that it is not fair for comparison. In the revised manuscript, we remove the work of [31] from the Table.

**Author's action:** We remove the comparison to the algorithms in [31] in the Table and the related text.

**Reviewer #2, Comment #3:**

3. Please review the paper for grammatical errors.

**Author's response:** We thank the reviewer for this comment. We will carefully check the grammatical error in the paper.

**Author's action:** We reviewed and edited the grammatical errors and highlighted them in the revised manuscript. The Table below is the list of some corrected parts.

|  |  |  |
| --- | --- | --- |
| **position** | **original text** | **correction** |
| **page 1, left column** | lacks | lacks of |
| **page 3, left column** | solved using | solved by using |
| **page 3, right column** | the conventional  multi-core mapping such as ilp or pso proves their | the conventional  multi-core mapping solutions such as ilp or pso prove their |
| **page 5, right column** | map | maps |
| **page 5, right column** | limit | limits |
| **page 6, left column** | it built | it builds |
| **page 7, right column** | which lead | which leads |
| **page 7, right column** | migrate | migrates |
| **page 2, left column** | forth | fourth |
| **page 2, right column** | an genetic algorithm | a genetic algorithm |
| **page 4, left column** | luster | cluster |
| **page 11, left column** | 2x2 2x2 | 2x2 |
| **page 8, left column** | surrounded | are surrounded |
| **page 9, left column** | compute | computes |
|  | kept | keeps |
| **page 10, left column** | is kept as it is | are kept as it are |
| **page 11, right column** | are feed | are fed |
| **page 14, left column** | support | supports |

# Reviewer #3

**Reviewer #3, Comment #1:**

The paper is well written and well organized, but there are some typos in the article, e.g.:

1- Page 3, Line 34, Column 1: Forth --> Fourth

2- Page 3, Line 19, Column 2: An genetic --> A genetic

3- Page 4, Line 56, Column 2: luster --> cluster

4- Page 5, Line 44, Column 1: fault neurons --> faulty neurons

5- Page 12, Line 49, Column 1: 2x2 2x2 --> 2x2

6- …

**Author's response:** We thank the reviewer for the correction.

**Author's action:** We updated the manuscript by fixing these typos. The edited parts are highlighted in the revised manuscript. We also carefully check the manuscript to correct further typos.

**Reviewer #3, Comment #2:**

Q1: Is there any reason that you uniformly distribute spare neurons in the nodes? For example, what happens if we have more spare neurons in the central nodes comparing the border nodes?

**Author's response:** We thank the reviewer for the comment on the distribution of spare neurons.

The main reason to have uniformly distributed spare neurons is that we assume the fault distribution to be uniform. In other words, we expect the fault rate are similar among clusters.

If the fault rate is uniform and we put more spare neurons in the central nodes, it will likely have system-level recovery rather than a node-level one. This is because non-central nodes have fewer extra neurons, and they need to migrate their neurons to other nodes.

If the fault rate is center-oriented, placing more spare neurons is more efficient.

**Author's action:** We have updated the reason why we need to adding spare neurons uniformly. We also explain the idea of adding non-uniform extra neurons.

**From page 5, left column line 54 to page 5, right column, line 7:**

In node-level recovery, the number of spare neurons in each cluster is an important parameter. Here, we assume the fault rates of all clusters are similar; therefore, we will plan to add the same number of spare neurons in each node. Non-uniform fault rates (for example, central nodes have more faults) will need {system-level recovery} more frequently as the nodes lack spares for recovery. On the other hand, if the fault rate can be predicted, we can distribute the spares based on the expected fault rates.

**Reviewer #3, Comment #3:**

Q2: How did you implement the proposed algorithms? Did you use any parallelization techniques (e.g. data or thread-level parallelization)?

**Author's response:** The algorithms are implemented in Java (software). We do not perform parallel or thread-level parallelization in Java. However, the program can be paralleled by the compiler or OS. The algorithm in [31] is in Python.

We understand that GA can be partially parallelized (i.e., computing the cost of each individual, crossover, and mutation). However, there is no guarantee that the neuromorphic system can perform the GA in parallel. For instance, an embedded CPU to configure and execute the GA serially.

**Author's action:**  we clarify the detail on **page 12, right column line 54-55**:

The algorithms are written in Java without thread parallelism.

**Reviewer #3, Comment #4:**

Q3: Do you support multiple fault occurrences? Is there any conflict in your proposed neuron migration method to find candidate neurons for migration when multiple faults occur in different nodes?

**Author's Response:** Yes, the method supports multiple faults occurs. In the evaluation section, we spare 20% of neurons unused. Then, we randomly inserted k faults. In Figures 12-16, we have k= 5%, 10%, 15% and 20% of the total number of neurons. In other words, we only test with multiple faults.

As the algorithm executes with multiple defects, there is no conflict happening. In both MCFC and GA, the position of the fault must be known before mapping.

We would like to thank all reviewers for their excellent comments, which help improve the manuscript significantly. This is the end of the answer file. The paper with highlighted parts is in the following pages.