Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

HeterGenMap: An Evolutionary Mapping Framework for Heterogeneous NoC-based Neuromorphic Systems

Khanh N. Dang[1] , Nguyen Anh Vu Doan[2] , Ngo-Doanh Nguyen[1] (Member, IEEE), and Abderazek Ben Abdallah[1], (Senior Member, IEEE),

[1]Graduate School of Computer Science and Engineering The University of Aizu, Aizu-Wakamatsu 965-8580, Fukushima, Japan (e-mail: {khanh, m5262108, benab}@u-aizu.ac.jp)

[2]Infineon Technologies AG, Germany (e-mail: anhvu.doan@infineon.com)

Corresponding author: Khanh N. Dang (e-mail: khanh@ u-aizu.ac.jp).

ABSTRACT While task mapping for multi-core systems is known as an NP-hard problem, mapping for neuromorphic systems even scale it up due to a high number of neurons per core and a high number of core per system. Moreover, mapping for neuromorphic systems also has several challenges such as heterogeneous computing core or communication fabrics, and potential defects in neurons or routing units. Therefore, this paper presents a genetic algorithm framework named *HeterGenMap* which is a Genetic Algorithm framework for mapping multiple-layer Spiking Neural Network systems to solve the aforementioned problems. The results show that *HeterGenMap* improves the overall communication cost by 11.04-26.77% in comparison to the linear mapping. Moreover, under link faulty scenarios, neuron defects, or multi-chip designs, *HeterGenMap* can reduce the communication cost by 3.41-31.34%, 7.01%-41.51%, and 34.21-45.56% in comparison to the linear approach, respectively. The validation in hardware also demonstrated that *HeterGenMap* reduces the inference time by 63.10-77.87% from the linear mapping.

INDEX TERMS Fault-tolerance, Spiking Neural Network, Neuromorphic System, Network-on-Chip, Max Flow, Migration

I. INTRODUCTION

Spiking Neural Networks (SNN), considered as the third generation of neural networks, mimics the operations of biological brains using spike-based computations. The SNN model is an arrangement of the replicated neurons to simulate natural neural networks existing in biological brains [1]. The backbone element of SNNs is the replicated version of neurons that receive input spikes, compute, and issue possible output spikes. In recent years, numerous works have investigated how to integrate a large number of neurons on a single chip while providing efficient and accurate learning [2]–[9].

Software implementations, thanks to their flexibility have demonstrated the ability to emulate the operation of biological brains at low speeds [7], [8]. Meanwhile, implementing SNNs hardware has been a solution to accelerate the performance thanks to their parallel structures and to gain energy and memory efficiency [2]–[6]. The conventional system usually consists of multiple clusters of neurons connected via an on-chip communication fabric [3], [9]. Scalability can be obtained by using a multi-chip system and off-chip interconnect [2], [3].

While providing hardware to accelerate the performance of the SNN is a promising idea, one of the major challenges is how to efficiently map the computing models into the hardware. As we know, mapping tasks into many/multi-core systems is an NP-hard problem and it can also be applied to Neuromorphic Computing. The general idea of task mapping is to allocate a given set of tasks (in SNN: neuron's computation) into a given set of computing units (in Neuromorphic chips/systems: hardware engine and memory). For an SNN model with n neurons, there are $n!$ possible mapping combinations. Mapping approaches for SNNs can go through a clustering step first, which moves the mapping to the cluster level [10]. However, it is still not ideal with heuristic solutions ($c!$ possible solutions for c computing cores) since task mapping is considered as an NP-hard problem. The equivalent problem of task mapping is the quadratic assignment problem [11].

Beside mapping toward lower communication traffic or reducing the connections, there are two notable problems for

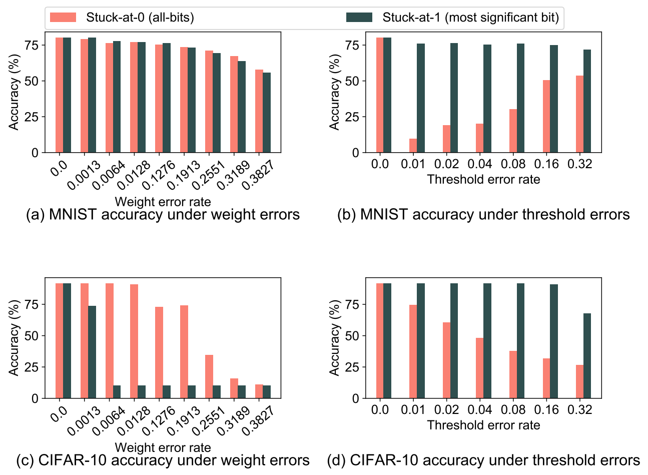


FIGURE 1: The illustration of the impact of defective weights or neurons on the overall accuracy.

large-scale neuromorphic systems that need to be addressed. First, the occurrences of defects within the system could be critical even though the neuromorphic system can tolerate some level of defects [12]–[15]. As being analyzed in [12] the flip-bit position and the defective layer can have different impacts on the overall accuracy. In Fig. 1, we analyzed the impact of defective weight or defective neurons on the overall accuracy of two benchmarks: MNIST with STDP learning and CIFAR10 using VGG16 and ANN to SNN conversion. In summary, the overall accuracy drop can be significant if the errors are not addressed. Because of the reliability issue, mapping for SNN should also consider the ability to deal with defective situations that alter the communication fabric. In this work, we consider the alternated communication fabric as *heterogeneous* communication since the connection is no longer uniform within the system. The second problem is to have a multi-chip system where each chip alone consists of a plural set of neurons. To extend to a large-scale model, it is common to have multiple chips connected to form a larger number of neurons [2], [3]. However, there is a difference between intra-chip and inter-chip communication where inter-chip wires usually consume more power, and have a higher latency than the intra-chip ones [16]. For this matter, it will be more efficient to have more frequently communicating neurons within the chip [17].

Although several mapping approaches for SNN have been proposed [10], [18], some limitations still exist:

1) *A conventional mapping usually goes through clustering first which does not support* ***heterogeneous*** *mapping*. Indeed, it is assumed that every core consists of the same number of identical neurons. However, in large-scale systems, a heterogeneous number of neurons per core could occur due to alternative designs or defects. Pre-clustering would not work well in this situation as the number of tasks and the number of compute units are different.
2) *Communication is also considered homogeneous among mapping algorithms for multi-core systems.* Although this can simplify the mapping process, neuromorphic systems usually come with a multiple-chip design. Therefore, communication is no longer uniform in all cases because it could vary between chips and between on-chip and off-chip communication fabrics. Moreover, defects in the communication units can lead to prohibited routing paths. This alters the routing path which makes some routing paths longer than minimal routing paths.
3) *Most SNN mapping approaches do not consider the mapping as an open and extendable optimization problem.* As mentioned earlier, the mapping procedure is an NP-hard problem, and tackling it is a tedious and time-consuming task. Therefore, it is necessary to develop an algorithm that can be flexible so that the designers can adapt to different targets or configurations.

To solve the aforementioned issues, this work introduces *HeterGenMap* - a genetic algorithm framework for mapping heterogeneous NoC-based neuromorphic systems. The following are the contributions of this work:

1) A heterogeneous mapping method for neuromorphic systems that allows a different number of neurons per core and even a different number of cores per chip. Here, instead of dividing neurons into clusters of similar sizes (each cluster fits into one core in NoC), we propose dividing the neurons by groups. Here, members of a group are exchangeable and a cluster (not fixed-sized) consists of a set of sub-sets of groups. This allows us to reduce the complexity of the mapping problem while still supporting heterogeneous mapping.
2) A model that supports diversifying the communication model which allows the mapping to consider: (1) multi-chips mapping with different routing costs between chips and (2) fault-tolerant mapping with the knowledge of faulty links/routers.
3) As a Genetic Algorithm approach, *HeterGenMap* introduces crossover, mutation, and fitness functions tailored for neuromorphic computing. Additionally, we present several approaches for crossover/mutations and how to integrate them into the GA process. By using a GA, the framework also supports extension abilities such as (1) switching the cost function or (2) using existing/known solutions in the evolution process.

Table 1 shows a comparison between this work and the state-of-the-art in mapping for neuromorphic computing. In summary, our work will tackle the aforementioned problems which, to the best of our knowledge, remain unsolved. The platform is also open-sourced and can be accessed at https://github.com/khanhdang/HeterGenMap.

The organization of this paper is as follows. Section II presents the related works. Next, the proposed method is introduced in Section III and the evaluation is provided in Section IV. In Section V, we discuss the outlook about this

TABLE 1: Comparison between this work and state-of-the-art works.

Method	Description	Clustering	Hetero-geneous mapping	Fault-tolerance	Multi-chip	Reusing existing solutions
NEUTRAMS [19]	Kernighan-Lin partitioning strategy to reduce the inter-core communication	✓	✓	✗	✗	✗
PSOPART [20]	Using PSO to partition	✓	✗	✗	✗	✗
SpiNeMap [21]	Cluster the neurons and place the clusters to minimize the communication/power	✓	✗	✗	✗	✗
MigSpike [13]	Max-flow-min-cut to migrate neurons after fault occurred	✗	✓	✓	✗	✗
R-NASH [22]	Genetic Algorithm for initial mapping	✓	✓	✗	✗	✗
TrueNorth [3]	Converting to the shortest wire-length in VLSI	✗	✓	✓	✓	✗
HeterGenMap (this work)	Using GA to map with awareness of heterogeneity in neurons and communicating fabric	Grouping by the similarity in communication	✓	✓	✓	✓

work, and Section VI concludes the paper.

II. RELATED WORK

In this section, we first summarize the related works on neuromorphic systems. Then, we will survey mapping for large-scale neuromorphic systems.

A. NEUROMORPHIC SYSTEMS

There are several considerations in the design of a neuromorphic system; however, we would like to focus on three major points: (1) communication, (2) neuron architecture, and (3) synapses. For communication, point-to-point or bus-based fabrics cannot scale up to a large system (i.e., a million neurons and a billion synapses). Therefore, adopting an on-chip and off-chip network has been a consensus among state-of-the-art works [2], [3], [5], [7], [23]. In general, 2D Mesh-based Network-on-Chip [2], [3] is the popular choice thank its scalability for both on-chip and off-chip communicating. In [22], 3D Mesh topology is used with the help of Through-Silicon-Via technology. For better multi-casting support, [5] used tree-like topology. In [24], the authors proposed a Dynamic Segmented Bus architecture that improves the power and latency in comparison to Mesh-based architecture.

In SNN design, due to its low complexity, Integrated-and-Fire (IF) and it variations have been the popular choice [2], [3], [5], [7], [23]. While full digital neurons [2], [3], [23] can perform more complex computation with the trade-off of larger area cost, mixed-analog-digital neurons [5], [7] offer lower cost and closer operation to the neuron model. [23] offer an Izhikevic neuron model in the design; however, the actual benefits of this type of neuron are not well demonstrated which still leaves IF-like neurons to stay dominant. To support more complex neuron computation, SpiNNaker [4] uses one million homogeneous ARM968 processors for emulation. Each ARM processor can simulate a thousand neurons allowing the system to save up to a billion neurons. SpiNNaker uses a fully packetized communication infrastructure built on a folded two-dimensional toroidal mesh where each node can communicate with six neighbors. The SpiNNaker system is also based on the AER format with table-based multi-casting support.

With such large-scale neuromorphic systems, the number of neurons, synapses, and even chips could be massive. Therefore, one of the major challenges is to allocate the position of the neurons and synapses properly. The next section will discuss the mapping problem for neuromorphic computing.

B. MAPPING

As a multi-core system, we can use a conventional mapping method for placing the neuron [25]. The conventional multi-core mapping method has prove their efficiency, mapping for NN is highly complicated due to a large number of neurons. To solve this issue, SpiNeMap [21] performs clustering before mapping. In the clustering process, neurons are divided into clusters which can reduce the mapping complexity. On the other hand, neurons can just be mapped linearly [26] which can simplify the mapping process but it leads to a sub-optimal result. In [3], the authors offer several mapping algorithms for the placing process and consider it as a wire-length minimization problem in VLSI placement. In [27], the authors used Lagrange multipliers to reduce the run-time complexity. NEUTRAMS [19] uses the Kernighan-Lin partitioning strategy to split the neurons into groups. In [28], the authors divide the adjacent layers into a so-called sub-network. Then, the sub-network will be partitioned to reduce the complexity of the algorithm. However, the results still stay with small scale-NoC size. With large-scale NoC, work in [29] proposed non-meta-heuristic algorithms with can map much larger-scale neural networks (up to billions of neurons) into NoC.

Although these mapping can work on neuromorphic systems, we notice there is no mapping method on fault-tolerance ones. Conventional multi-core NoC mapping methods [30] can be used for placing neurons. However, the mapping process in SNN is unique due to its connectivity. Instead of reducing the whole cost of the communication path, it is better to reduce the maximum communication path as they may affect performance. In [13], the authors use max-flow-min-cut to migrate the neurons if a fault occurs. Although this works best for run-time maintenance, we also observe it is better to remap if the fault appears after the manufacturing process. Moreover, their method does not support fault in communication at all.

Despite there are numerous works on mapping neuromorphic computing; there are several limitations on mapping *heterogeneous* systems as we illustrated in the introduction. To solve the mentioned limitations, this work will present a genetic algorithm framework.

III. PROPOSED APPROACH

In this section, we present the proposed genetic algorithm framework for mapping SNN. First, we show an overview of the potential system and formulate the problem. Then, the genetic algorithm approach is presented. We also discuss different cost functions and the extension on fault tolerance and multi-chip design. Finally, the complexity of the method is analyzed.

A. SYSTEM OVERVIEW

Figure 2 shows an overview of a Mesh topology (2D or 3D) NoC-based SNN system. Neurons (N) are clustered into computing cores (C). Each neuron (N) is attached to its synapses (S). Here, hardware neurons are implemented in *parallel* model where each physical neuron performs a neuron's computing task. In a *serial* neuron system such as [3], there is one physical neuron that will be used for multiple neurons' computation along multiple synapse memories. The communication from one core to other cores is handled by the Network-on-Chip router (R). If a 3D Mesh NoC is used, spikes to another layer are sent via an Inter-layer Vias (V) that connects adjacent silicon layers. Although this system is based on our previous works on 3D-NoC-based neuromorphic systems [9], [13], the mapping approach here is general and could be applied to other systems as well.

Additionally, neuromorphic systems can be implemented into multiple-chip design thanks to the inter-chip link [2], [3]. Figure 2 illustrates the expansion of Mesh-based NoC into adjacent chips as the inter-chip links work as router-to-router links.

In this paper's evaluation, we mainly consider the mesh topology as it is the most popular one among neuromorphic chips [2], [3], [13]. However, there are different mapping topologies such as tree-based [5] or folded torus [4]. Adapting to these topologies only requires modifications in the cost functions as will be detailed in Section III-D.

B. PROBLEM FORMULATION

The mapping problem for SNN is illustrated in Fig. 3. The development starts by completing the SNN model. Then, the mapping process helps generate the location of each neuron

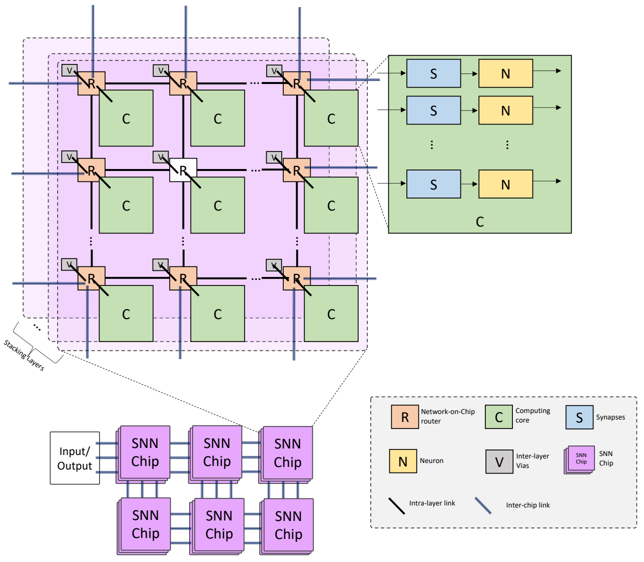


FIGURE 2: A system overview of Mesh topology NoC-based SNN. The system can consist of multiple chips connecting via inter-chip links. Within the single chip, Mesh Network-on-Chip is utilized as the communication infrastructure.

in the neuromorphic hardware where the system consists of a fixed set of cores and each core can host a fixed number of neurons. There are two types of communication: (1) *intra-core communication* where the neurons within the core exchange the spikes; and (2) *inter-core communication* where the neurons in distant cores exchange the spikes.

While the *intra-core communication* is usually fast and without noticeable latency; the *inter-core communication* is the major problem for NoC-based neuromorphic systems. As we illustrated in Fig. 3, the neuron's spikes must travel several routing units (routers) to be delivered to the destination. If the mapping process is suboptimal, it leads to long latency routing paths which give two major consequences: (1) there are bottle-necks in communication in which the SNN chip must wait for the spikes for communication, and (2) higher power consumption due to the routing process. Because of these reasons, in SNN mapping, inter-core communication will play an important role in the system's performance.

With this mapping formulation, there are two options: *neuron-wise mapping* and *group-wise mapping*. In the following section, we will discuss the two options.

1) Neuron-wise mapping

As mentioned earlier, the task mapping problem for multi-core systems can be transferred into an equivalent problem named quadratic assignment problem [11]. Let us assume N neurons per system to be mapped into C cores. A solution s of the problem can be presented as a binary matrix

$$s_{neuron} = \begin{bmatrix} \epsilon_{1,1} & \epsilon_{1,2} & \dots & \epsilon_{1,C} \\ \vdots & \ddots & & \\ \epsilon_{N,1} & \epsilon_{N,2} & \dots & \epsilon_{N,C} \end{bmatrix} \tag{1}$$

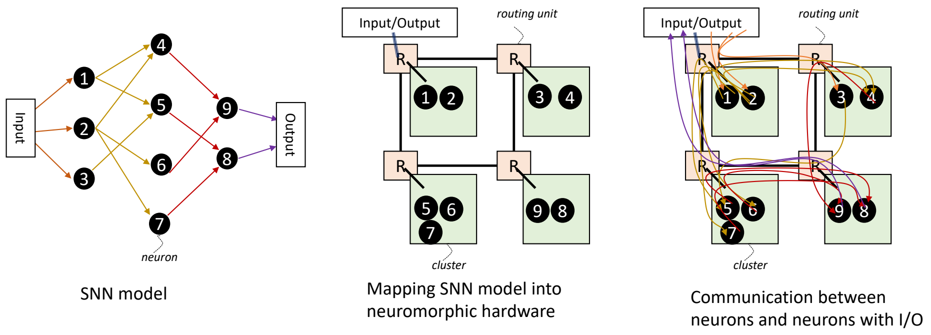

Input
1
2
3
4
5
6
7
8
9
Output
neuron
SNN model
Input/Output
routing unit
R
cluster
Mapping SNN model into neuromorphic hardware
Communication between neurons and neurons with I/O

FIGURE 3: An overview of the mapping problem.

where $\epsilon_{i,j}$ represents the mapping of neuron i to core j as follows:

$$\epsilon_{i,j} = \begin{cases} 1 \text{ if neuron } i \text{ is mapped into core } j \\ 0 \text{ if neuron } i \text{ is NOT mapped into core } j \end{cases} \quad (2)$$

In this formulation, we consider each neuron as one task in the system and need to map them accordingly. Although a *neuron-wise mapping* can lead to fine-grain solutions, we can easily realize it has high complexity since it must map each neuron individually. To solve this problem, SpiNeMap [21] grouped the neurons into clusters where each cluster could be mapped into a hardware cluster or core. However, in a heterogeneous system, each hardware cluster could have a different number of neurons which prevents this clustering idea from being feasible. Moreover, defects could occur in large-scale systems (defects in neuron logic or memory cells) which also alter the number of available neurons in each cluster.

2) Group-wise mapping

Given the challenges and limitations posed by neuron-wise mapping, we adopt instead a *group-wise mapping* approach with the following assumptions:

- A group of neurons does not have a specific size for fitting into a cluster/core.
- A group of neurons must have a high similarity of communication so that the neurons can be exchangeable. For instance, neurons in the same layer of a multi-layer perceptron are grouped since they share the same inputs and outputs. By doing so, we map for its representative group instead of mapping for each single neuron.
- A group will have represent connections (incoming and outgoing) which is the union of all members' connections in the group. Here, we eliminated the sparsity to reduce the complexity of the model. With sparsely connected models, we can later decide the position of each neuron based on its connections rather than its group's connections.

Here, a group is a task that operates in parallel in several locations. Neurons are therefore a parallel routine in the task. By having a representative class, the number of variables becomes much smaller. After mapping a group, we can place the neurons as the second mapping problem.

The mapping problem can be reduced to:

$$s_{group} = \begin{bmatrix} \psi_{1,1} & \psi_{1,2} & \dots & \psi_{1,C} \\ \vdots & \ddots & & \\ \psi_{G,1} & \psi_{G,2} & \dots & \psi_{G,C} \end{bmatrix} \quad (3)$$

where G is the number of groups and $\psi_{i,j}$ represents the number of neurons of group i that are mapped to core j.

Certainly, one neuron should not be mapped into multiple cores and the number of neurons mapped to a core should not exceed the core capacity. The constraints for the mapping are:

$$\forall i \in \{1 \dots G\} \sum_{j \in \{1 \dots C\}} \psi_{i,j} = Group\_Cap_i \quad (4)$$

$$\forall j \in \{1 \dots C\} \sum_{i \in \{1 \dots G\}} \psi_{i,j} \leq Core\_Cap_j \quad (5)$$

where $Core\_Cap_j$ is the capacity for core j and $Group\_Cap_i$ is the capacity for group i.

As mentioned in the introduction, we will consider in this work the capacity of each core separately as there is no guarantee that they are identical. For mapping neurons separately, we can set the group size to one, $G = N$, and $\forall i \in \{1 \dots G\} Group\_Cap_i = 1$.

3) Final formulation

Finally, we can formalize the problem as follows:

$$\begin{aligned} &\min \text{ or } \max F_{cost}(s) \\ \text{s.t. } &\forall i \in \{1 \dots G\} \sum_{j \in \{1 \dots C\}} \psi_{i,j} = Group\_Cap_i \\ &\forall j \in \{1 \dots C\} \sum_{i \in \{1 \dots G\}} \psi_{i,j} \leq Core\_Cap_j \end{aligned} \quad (6)$$

The cost function (F_{cost}) will be discussed later as we design the framework. We will also discuss how this model can be extended with different types of cost functions.

With the final formulation, we can now solve the problem using Genetic Algorithm as in the following section.

C. GENETIC ALGORITHM FRAMEWORK

This section presents the proposed genetic algorithm framework as shown in Algorithm 1. First, the inputs such as the SNN model, the NoC topology, and the capacity in each core are loaded. In the beginning, the framework randomizes $K - E$ mapping solutions (line 1). Then, we could add E existing/known solutions as part of the initial solution (line 2). Please note that this step is optional and we can perform with $E = 0$. After having K mapping solutions, it starts G generations of improvement. In each generation, the algorithm first removes the incorrect mappings by checking Equation 4 and 5. This ensures the solutions taken into the crossover and mutation are correct. Then the cost function such as the communication cost is calculated (line 5). The details about the cost function are shown in Section III-D. B best pairs are then selected out of K mapping solutions by using a tournament selection (line 6). Please note that the pair must not be identical and a solution can be in more than one pair.

Algorithm 1 Proposed Genetic Algorithm for Neurons Mapping

```
Input:  SNN model, NoC topology, Core_Cap
Output: $s_{group}$
    Initialisation :
 1: Randomize to get K-E mapping solutions
 2: Add E existing solutions to form K initial solutions
 3: for generation $g$ in 1 to G do
 4:     Remove the wrong solutions using Equation 4 and 5
 5:     Calculate the cost function for each solution
 6:     Select the $B$ best pairs using tournament selection
 7:     Crossover solutions to have a new population
 8:     Fixing the post-crossover population (see Algorithm 2)
 9:     Mutate the population
10:     Select $K$ best solutions to create the new population
11: end for
12: Calculate the cost function for each solution of the population
13: Select the best solution ($s_{group}$) based on the cost function
14: return $s_{group}$
```

After having the B best pairs, a crossover operation is performed to obtain B new offspring. Then, we merge them to the original B solutions to create $B \times 2$ solutions using the crossover.

After the crossover generates $B \times 2$ solutions, the GA eventually repairs the offspring since the crossover can lead to infeasible solutions (see Section III-C3 for details). Next, the mutation operation is performed, under a certain probability.

After the mutation, the repair operation is again performed as the process can lead to infeasible results. Then, we check whether we satisfy the communication cost in the specification. If the communication cost is good enough, we can end the GA. The GA is also completed after G generations.

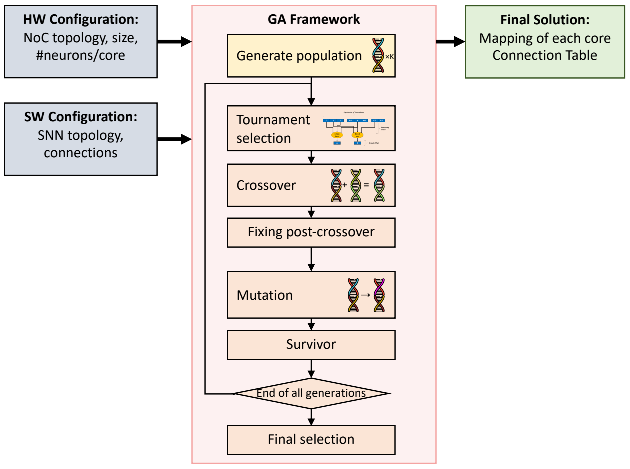

HW Configuration: NoC topology, size, #neurons/core
SW Configuration: SNN topology, connections
GA Framework
Generate population
×K
Tournament selection
Crossover
Fixing post-crossover
Mutation
Survivor
End of all generations
Final selection
Final Solution: Mapping of each core Connection Table

FIGURE 4: The *HeterGenMap* framework.

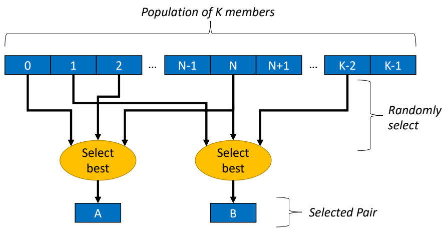


FIGURE 5: Overview of tournament selection.

At the end of the algorithm, the best solution in terms of cost function is chosen to generate the configuration.

In the following subsections, we will discuss the tournament selection, crossover, mutation, and the method to fix wrong offspring due to crossover and mutation.

1) Tournament Selection

To understand how to select a member of the population, this section discusses the tournament selection. The overview of tournament selection is shown in Figure 5. In this selection, the GA platform selects randomly a subset of population members. Among the subsets, the best member is chosen based on the fitness function (minimum or maximum). By repeating this selection, we can obtain any B best pairs for crossover.

2) Crossover

The first key operation is crossover. In this work, we proposed two crossover approaches: (1) Mixing and (2) Single Point. Assuming we have two parent solutions:

$$s_A = \begin{bmatrix} \psi^A_{1,1} & \psi^A_{1,2} & \cdots & \psi^A_{1,C} \\ \vdots & \ddots & & \\ \psi^A_{G,1} & \psi^A_{G,2} & \cdots & \psi^A_{G,C} \end{bmatrix} \tag{7}$$

and

$$s_B = \begin{bmatrix} \psi^B_{1,1} & \psi^B_{1,2} & \cdots & \psi^B_{1,C} \\ \vdots & \ddots & & \\ \psi^B_{G,1} & \psi^B_{G,2} & \cdots & \psi^B_{G,C} \end{bmatrix} \tag{8}$$

Mixing crossover

With the mixing crossover, we assume there are two probabilities for solutions A and B as p_A and p_B ($p_A + p_B = 1.0$). The offspring solution is obtained as follows:

$$s_C = \begin{bmatrix} \psi^C_{1,1} & \psi^C_{1,2} & \cdots & \psi^C_{1,C} \\ \vdots & \ddots & & \\ \psi^C_{G,1} & \psi^C_{G,2} & \cdots & \psi^C_{G,C} \end{bmatrix} \tag{9}$$

where each element of the matrix $\psi^C i,j$ is:

$$\psi^C i,j = round(p_A \times \psi^A i,j + p_B \times \psi^B i,j) \tag{10}$$

Because of this rounding equation, the constraints in Eq. 4 and 5 are respected. However, in some cases, the values may be not satisfied due to the rounding process (missing one or having an extra one). To solve this issue, we must go through repair operation as detailed in Section III-C3.

Single Point crossover

With the single point crossover, the crossover point P will be between 1 and N. Here, the offspring from the two solutions are:

$$c_1 = \begin{bmatrix} \psi^A_{1,1} & \psi^A_{1,2} & \cdots & \psi^A_{1,C} \\ \vdots & \ddots & & \\ \psi^A_{P,1} & \psi^A_{P,2} & \cdots & \psi^A_{P,C} \\ \psi^B_{P+1,1} & \psi^B_{P+1,2} & \cdots & \psi^B_{P+1,C} \\ \vdots & \ddots & & \\ \psi^B_{G,1} & \psi^B_{G,2} & \cdots & \psi^B_{G,C} \end{bmatrix} \tag{11}$$

and

$$c_2 = \begin{bmatrix} \psi^B_{1,1} & \psi^B_{1,2} & \cdots & \psi^B_{1,C} \\ \vdots & \ddots & & \\ \psi^B_{P,1} & \psi^B_{P,2} & \cdots & \psi^B_{P,C} \\ \psi^A_{P+1,1} & \psi^A_{P+1,2} & \cdots & \psi^A_{P+1,C} \\ \vdots & \ddots & & \\ \psi^A_{G,1} & \psi^A_{G,2} & \cdots & \psi^A_{G,C} \end{bmatrix} \tag{12}$$

In short, the child c_1 has the upper part from ψ_A and the bottom part from ψ_B. Meanwhile, the child c_2 has the upper part from ψ_B and the bottom part from ψ_A. Note that the produced offspring might not satisfy the constraints in Equation 4 and 5. To avoid them being eliminated in $S3$, we propose a repairing method in the following section.

Note that it is possible to easily perform crossover using points in the core dimension. Also, multi-points and random crossover are possible without any modification.

3) Fixing the offspring

After crossover or mutation, the offspring solution can violate the constraints in two ways:

- Assigning more the capacity of the core with crossover in neuron dimension.
- Assigning a neuron in multiple cores or no core with crossover in core dimension.

Here, the repair is done by exchanging the position of the neuron/core, as shown in Algorithm 2. It first finds the list of cores that have more or less than their capacity. Then, it randomizes the moving between these cores (from the over-populated, referred to as *overlist* in the algorithm, to the under-populated, referred to as *underlist*).

Algorithm 2 Offspring fixing algorithm

```
Input:  $s_{group}$
Output: $s_{group}$
    Initialisation :
 1: under-list = []
 2: over-list = []
 3: for core $c$ in 1 to C do
        $residual = \sum_{i=1...N} \psi_{i,c} - Cap_c$
 4:     if residual > 0 then
 5:         push [c, residual] to over-list
 6:     else
 7:         push [c, residual] to under-list
 8:     end if
 9: end for
10: for [c1, r1] in overlist do
11:     for i in 1 to r1 do
12:         randomized [c2, r2] with r2>0 in underlist
13:         move one neuron from core c1 to core c2
14:         [c2, r2] = [c2, r2-1]
15:         [c1, r1] = [c1, r1-1]
16:     end for
17: end for
18: return $s_{group}$
```

Example: To understand how the cross-over and fixing of the offspring work, we illustrate them in Fig 6. In this example, the NoC is a 2D Mesh 3×3 with 4 neurons/core. The SNN architecture is feed-forward with the configuration of three groups of [12, 12, 10]. Here, we assume the two solutions s_A and s_B as in the Fig 6 and the mixing crossover give us the result s_C using the Eq. 10. Due to the natural feature of the rounding process, the new solution s_C has the total configuration of [11, 11, 10] which gives a residual of [1, 1, 0] to be fixed. By using the fixing process in Algorithm 2, we can obtain the final solution s_C as in Fig 6.

4) Mutation

The mutation process is done by exchanging the neuron mapping. Here, we distinguish the mutation into two types:

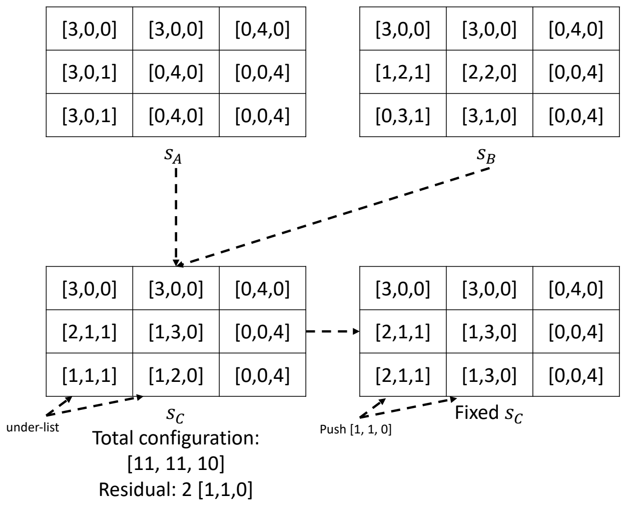


FIGURE 6: Example of crossover and fixing the offspring.

- *Exchange neurons between cores*: in this type of mutation, we only switch a certain number of neurons between cores. The other neurons stay at the original core
- *Swap all neurons between cores*: in this type of mutation, we swap all neurons between two cores. The internal structures of the cores are unchanged.

During the GA process, the mutation method will be alternated in a fixed ratio (i.e. 50% for exchanging neurons between cores and 50% for exchanging all neurons between cores) in order to have different distributions over the search space. Designers can also add extra mutation methods.

Exchange neurons between cores:

For instance, assuming the solution s has $\psi_{a,b} > 0$ and $\psi_{c,d} > 0$ (a, b, c and d are randomly picked), the crossover process will be as follows:

$$\begin{aligned} e &= \min(\psi_{a,b}, \psi_{c,d}) \\ \psi_{a,b} &= \psi_{a,b} - e \\ \psi_{a,d} &= \psi_{a,d} + e \\ \psi_{c,d} &= \psi_{c,d} - e \\ \psi_{c,b} &= \psi_{c,b} + e \end{aligned} \tag{13}$$

In short, we exchange the neuron between core b and core d. The exchanged neuron belongs to class a and c (note: class can be neuron or layer/group).

The original solution s_{orig} will be represented as

$$s_{orig} = \begin{bmatrix} \psi_{1,1} & \cdots & \psi_{1,b} & & \cdots & & \psi_{1,C} \\ \vdots & \ddots & & & & & \\ \psi_{a,1} & \cdots & \psi_{a,b} & \cdots & \psi_{a,d} & \cdots & \psi_{a,C} \\ \vdots & \ddots & & & & & \\ \psi_{c,1} & \cdots & \psi_{c,b} & \cdots & \psi_{c,d} & \cdots & \psi_{c,C} \\ \vdots & \ddots & & & & & \\ \psi_{G,1} & \cdots & \psi_{G,b} & & \cdots & & \psi_{G,C} \end{bmatrix}, \tag{14}$$

and the mutated solution s_{mut} will be:

$$s_{mut} = \begin{bmatrix} \psi_{1,1} & \cdots & \psi_{1,b} & & \cdots & & \psi_{1,C} \\ \vdots & \ddots & & & & & \\ \psi_{a,1} & \cdots & \psi_{a,b} - e & \cdots & \psi_{a,d} + e & \cdots & \psi_{a,C} \\ \vdots & \ddots & & & & & \\ \psi_{c,1} & \cdots & \psi_{c,b} + e & \cdots & \psi_{c,d} - e & \cdots & \psi_{c,C} \\ \vdots & \ddots & & & & & \\ \psi_{G,1} & \cdots & \psi_{G,b} & & \cdots & & \psi_{G,C} \end{bmatrix} \tag{15}$$

Swap all neurons between cores

In this type, we follow this equation:

$$\begin{aligned} e_{row} &= [\psi_{x,1}, \psi_{x,2}, ..., \psi_{x,C}] \\ [\psi_{x,1}, \psi_{x,2}, ..., \psi_{x,C}] &= [\psi_{y,1}, \psi_{y,2}, ..., \psi_{y,C}] \\ [\psi_{y,1}, \psi_{y,2}, ..., \psi_{y,C}] &= e_{row} \end{aligned} \tag{16}$$

The original solution will be represented as

$$s_{original} = \begin{bmatrix} \psi_{1,1} & \psi_{1,2} & \cdots & \psi_{1,C} \\ \vdots & \ddots & & \\ \psi_{x,1} & \psi_{x,2} & \cdots & \psi_{x,C} \\ \vdots & \ddots & & \\ \psi_{y,1} & \psi_{y,2} & \cdots & \psi_{y,C} \\ \vdots & \ddots & & \\ \psi_{G,1} & \psi_{G,2} & \cdots & \psi_{G,C} \end{bmatrix}, \tag{17}$$

and the mutated solution will be:

$$s_{mutated} = \begin{bmatrix} \psi_{1,1} & \psi_{1,2} & \cdots & \psi_{1,C} \\ \vdots & \ddots & & \\ \psi_{y,1} & \psi_{y,2} & \cdots & \psi_{y,C} \\ \vdots & \ddots & & \\ \psi_{x,1} & \psi_{x,2} & \cdots & \psi_{x,C} \\ \vdots & \ddots & & \\ \psi_{G,1} & \psi_{G,2} & \cdots & \psi_{G,C} \end{bmatrix} \tag{18}$$

Example: Fig. 7 shows the example of two mutation methods. First, the swapping neurons between cores are illustrated in Fig. 7(a) where two cores $[1, 0]$ and $[1, 2]$ are swapped. Here, the configurations of the groups ($[0, 0, 4]$ and $[3, 0, 1]$) are kept but allocated to different positions. Previously, in the $s_{original}$, the two cores $[1, 0]$ and $[1, 2]$ have configurations of $[0, 0, 4]$ and $[3, 0, 1]$, respectively. With the mutation, the core $[1, 0]$ has the configuration of $[3, 0, 1]$ and the core $[1, 2]$ now has the configuration of $[0, 0, 4]$. As we can observe, the overall configuration of the system stays unchanged and we don't need any fixing method.

On the other hand, Fig. 7(b) shows the method of exchanging neurons between cores. In this example, we have two cores $[1, 0]$ and $[2, 0]$ with the same configuration of $[3, 0, 1]$. From Eq. 13, we can conduct the common value e is 1. Then,

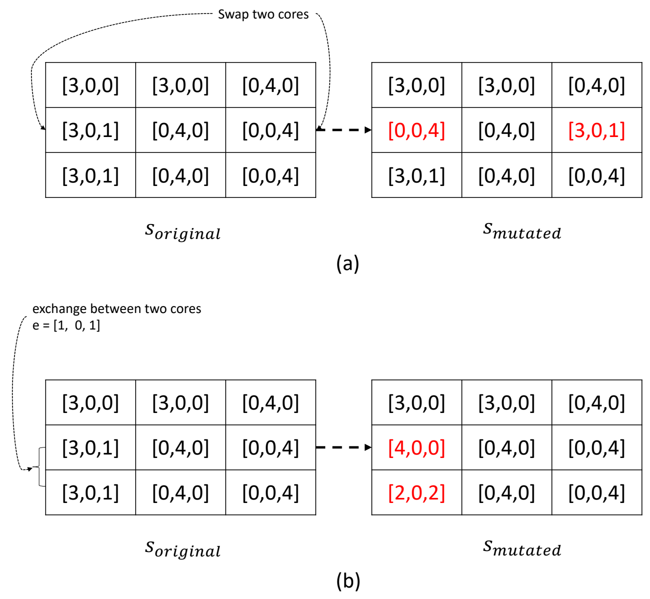


FIGURE 7: Example of mutation: (a) swapping all neurons between cores; (b) exchanging neurons between cores.

we perform a mutation of the two cores to move one neuron from the first group from the core $[2,0]$ to the core $[1,0]$. In the opposite direction, one neuron of the last group is moved from the core $[1,0]$ to the core $[2,0]$. The result of the mutation also satisfies all the constraints.

5) Survivor selection

After the crossover and mutation operations are completed, the new population size exceeds the predefined population size. Before moving to the next iteration, we put it through a survivor selection function where the members with the best fitness are retained. In this work, we use *tournament selection* algorithm in the DEAP framework [31]. The selection is done by having a random subset of the population selected for a tournament. The best member out of the tournament is put into the new population.

D. COST FUNCTION

In this section, we discuss designing the cost function corresponding to the objective of the framework.

For initial mapping, the common objective is to minimize the overall communication of the spike. To do so, we can use the communication cost F_{comm}:

$$F_{comm} = \sum_{j=1}^{C}\sum_{i=1}^{G} d_{ij} \times c_{ij} \tag{19}$$

where d_{ij} and c_{ij} are the Manhattan distance and the connection status between neuron i and j, respectively. Since the data transfer is in a multi-cast manner at each node, c_{ij} is the connection between two PEs. For this type of F_{cost}, we can have two options:

$$c_{ij} = \begin{cases} 1 \text{ if group } i \text{ send spikes to group } j \\ 0 \text{ otherwise.} \end{cases} \tag{20}$$

From the communication, we can also convert to a similar cost function based on so-called average hops (Avg_{hops}, which represent the average distance of hops that spikes travel inside the NoC.

$$Avg_{hops} = \frac{F_{cost}}{\#synapses} \tag{21}$$

With the genetic algorithm-based framework, *HeterGenMap* can solve by optimizing the cost function. In the following section, we will discuss how to deal with *heterogeneous mapping*.

E. HETEROGENEOUS MAPPING: USING AS MAPPING FOR DEFECTIVE AND/OR MULTI-CHIP SYSTEM

In MigSpike [13], a system under defects is remapped into a new mapping solution for dealing with defects in neurons. In this work, the cost of migration (moving neurons between cores) is the major target. As a result, the solution may have long traversal paths between connected neurons. Therefore, the communication cost can be high in this situation. To solve this issue, we reuse the GA framework for dealing with defects. Unlike in other mapping approaches, the capacity of each core is no longer uniform. In other words, each core will have a different capacity. In the **FT** (fault-tolerance) mode, *Core_Cap* will be adjusted according to the number of defects as follows:

$$Core\_Cap_j = \begin{cases} Core\_Cap_j - n\_defects_j \text{ if fault-tolerance} \\ Core\_Cap_j \text{ otherwise.} \end{cases} \tag{22}$$

where $n\_defects_j$ is the number of defects at core j.

On the other hand, defects on routing units (links, NoC routers, off-chip wires) can also lead to disconnected paths. As a consequence, we can end up with non-minimal routing, as shown in Figure 8. Another issue is the multi-chip design as illustrated in Figure 9. While on-chip links have a similar latency, multi-chip design requires inter-chip wires. This type of interconnect medium may require a different latency which needs to be modeled accordingly. For instance, in SpiNNaker [4], the latency of communication between ARM cores is much lower than the latency from one ARM core to another ARM core located on a faraway PCB (Printed Circuit Board). Hierarchical architectures such as H-NoC [32] have different latencies between clusters and among the clusters.

Due to the aforementioned issues, communication and computation can be heterogeneous in the context of neuromorphic computing. Because of this, state-of-the-art works such as [3], [19], [20] may not be suitable for this problem. To solve this problem with our GA framework, instead of using the Manhattan distance equation, we build a table of costs between pairs of neurons. Without heterogeneous connections, the value of each member of the table is identical to

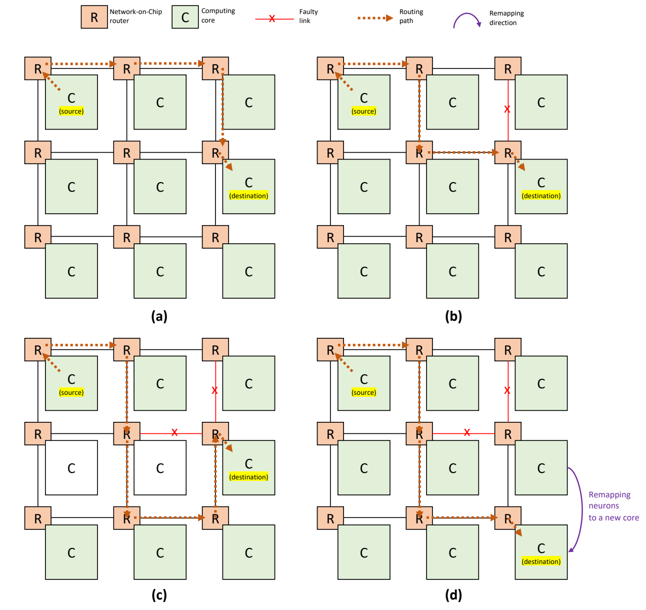


FIGURE 8: NoC routing and fault-tolerance feature: (a) normal situation with minimal routing; (b) defective links without impact on minimal routing; (c) defective links with impact on minimal routing; (d) mapping solution must consider defective parts into consideration.

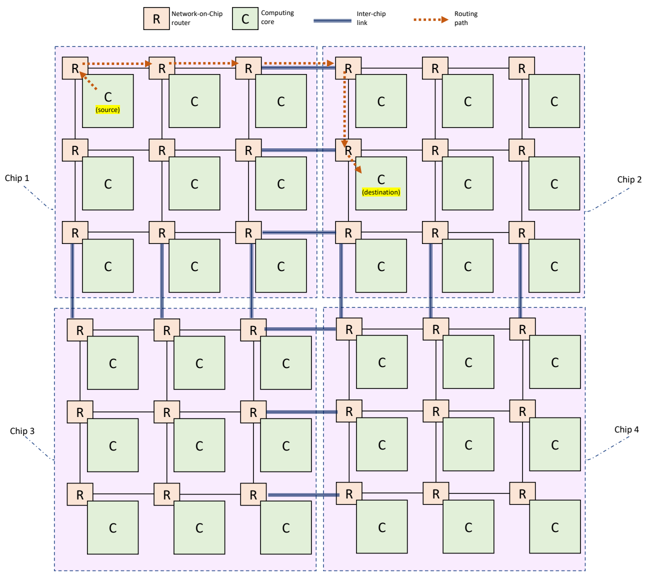


FIGURE 9: NoC routing with multi-chip design. Besides router-to-router on-chip link, inter-chip wires are introduced to scale up the system.

the Manhattan distance. In the case of heterogeneous systems, we find the shortest path using the Dijkstra algorithm.

1) Supporting fault-tolerance and multi-chip system

As mentioned earlier, the communication cost usually considers the distances uniformly. As in Eq. 19, the Manhattan distance d_{ij} is used. Following is the distance within a 3D-NoC:

$$d_{ij} = |x_i - x_j| + |y_i - y_j| + |z_i - z_j| \tag{23}$$

where x_i, y_i, z_i are the x-, y-, and z-coordinate of the neuron i, respectively and x_j, y_j, z_j are the x-, y-, and z-coordinate of the neuron j, respectively. However, we can observe two challenges:

- Once there is a defect in the minimal routing paths between i and j, the actual distance of spike traversal is larger than the Manhattan distance.
- As it is popular in neuromorphic systems, multiple chips can be attached on the same PCBs or multiple PCBs connected in the mainframe are used. Therefore, communication is no longer only on-chip but also consists of off-chip communication. Because off-chip communications usually have a higher latency than on-chip ones, it cannot be simplified to being similar to on-chip latency.

Therefore, in this work, we propose a method for supporting fault-tolerance and multi-chip systems. The distance is no longer considered as in Eq 23 but is based on the actual system's variations. To do so, we build a table of distances between two neurons' IDs and the distance will be looked up during the computation.

2) Routing in Network-on-Chip

Since the spike latencies are very critical in SNNs, we must first cover the routing in the Network-on-Chip. In this work, we will consider Mesh topology as the main target. However, the change in topology can be utilized by the change in the routing mechanism. Generally, the routing for NoC is usually done by a simple XY or XYZ routing [3], [22].

Figure 8 illustrates the NoC routing and fault-tolerance feature. Here, we illustrate with a 3×3 NoC using Mesh topology. We assume that there is a connection between two neurons from the core (source) and the core (destination) as in Figure 8(a). Here, we use XY routing (X first) to route the packet and we have the routing path. Once an edge in the routing path is defective as in Figure 8(b), we certainly need to adapt the XY routing to have a different routing path which is still minimal. This is an ideal case that has a small impact on the latency of the spikes as the routing path is still minimal. We must note that defective links could create congestion and introduce extra latency even if minimal routing paths are guaranteed.

Figure 8(c) shows a different scenario where two defective links make the minimal routing infeasible. There are two possible solutions:

- Using *non-minimal routing* to route the spikes as in Figure 8(c). Although non-minimal routing can lead to a live-lock situation, since spikes are routed separately, this behavior can be alleviated.
- Remap the system to enforce minimal routing as in Figure 8(d). Here, the neurons are remapped into the nearby core which allows a minimal routing path.

These two aforementioned approaches can be used to deal with permanent faults during the mapping process. Unlike traditional applications where the communication can be relaxing; SNNs operation is based on spikes which should be delivered on time. Therefore, the latency of the packets is extremely important.

IV. EVALUATION

A. EVALUATION METHODOLOGY

In this section, we will evaluate the proposed genetic algorithm framework under different configurations of Network-on-Chip. The genetic algorithm platform is written in Python using the DEAP library [31]. The mapping approach is performed using Python 3.0 run on Core i7 11700k. 32 GB memory and RXT 3060. The RTL model is written in Verilog HDL and simulated using Modelsim. The configurations for this evaluation can be seen at Table2 and 4.

First, we evaluate the 2D Mesh and then the 3D Mesh topology in a large range of sizes. Then, we evaluate the mapping under fault situation: (1) fault in communication; (2) fault in computing unit and memory; and (3) fault in communication, computing unit, and memory. In each part, we evaluate the different cost functions (three options as in Section III-D). For each mapping, we will use synthetic and real benchmarks.

To benchmark the proposed GA platform, we use two synthetic configurations (S#1 and S#2) and two realistic applications (MNIST and CIFAR-10 classification) as in shown Table 2.

To understand the efficiency of the proposed method, we compare this work to the linear mapping approach in terms of communication costs. We also later compare our work with other conventional approaches but without heterogeneous mapping. We also would like to mention that, to the best of our knowledge, there is no existing heterogeneous mapping approach (fault-tolerance or multi-chip) for the neuromorphic systems.

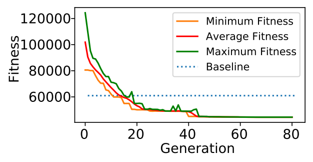
B. GENETIC ALGORITHM PERFORMANCE

In this section, we discuss the proposed Genetic Algorithm performance. First, the convergence of GA is evaluated. Then, we show the impact of reusing an existing solution as a member of the initial solution. Fault-tolerant link mapping and multi-chip mapping are also evaluated.

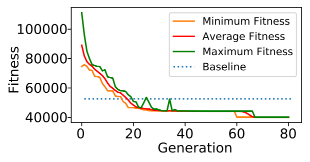
1) Convergence of GA

For the first mapping evaluation, we will use both 2D and 3D Mesh topologies. Figures 10 and 11 show the results of GA convergence on synthetic and realistic benchmarks, respectively. With small configurations, the GA can converge around 40 to 60 generations. When increasing the size of the NoC/SNN, it takes up to 200 generations until it saturates. Compared to the baseline (using linear XYZ mapping), our result improves 11.04-26.77% of the communication cost (fitness).

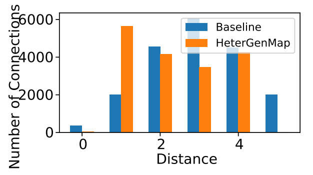
We also illustrate the distribution of the routing distances within the neuromorphic system. In comparison to the baseline model, *HeterGenMap* has a smaller number of connections at long distances. Furthermore, we can observe that the baseline has a longer maximum distance. For instance, with S#1 in 2D Mesh, as shown in Figure 10(c), the maximum distance of baseline is 5 while our proposed *HeterGenMap* is 4. We can observe a similar behavior in the S#2 and VGG16-CIFAR configurations.



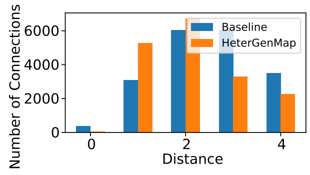
(a) Evolution of **S#1** in 2D Mesh



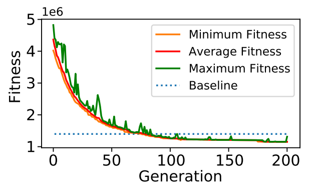
(b) Evolution of **S#1** in 3D Mesh



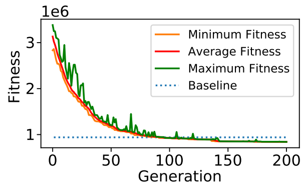
(c) Distribution of connection length of the final solution for **S#1** in 2D Mesh



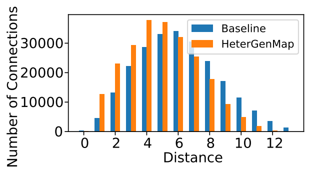
(d) Distribution of connection length of the final solution for **S#1** in 3D Mesh



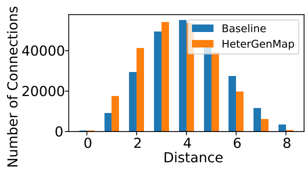
(e) Evolution of **S#2** in 2D Mesh



(f) Evolution of **S#2** in 3D Mesh



(g) Distribution of connection length of the final solution for **S#2** in 2D Mesh



(h) Distribution of connection length of the final solution for **S#2** in 3D Mesh

FIGURE 10: Evolution process of the synthetic SNN applications

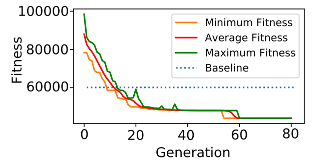
2) Using an existing solution as member of initialized population

Another approach to have better results is to use an existing solution as a member of the initialized solution. Here, we add the baseline as one of the initial solutions ($E = 1$). Table 3 shows the results of *HeterGenMap* with and without an initial solution. In summary, with the initial solution, *HeterGenMap* obtained better results under the same number of generations/population members. This is because, by using the baseline as one of the initial solutions, *HeterGenMap* can

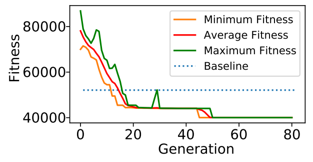
TABLE 2: Configuration of 2D and 3D Mesh for SNN mapping

Configuration	S#1[1]	S#2[2]	MLP-MNIST[3]	CNN-CIFAR10[4]
# cores	16	64	16	104
# neurons per core	256	256	256	256
# neurons	4096	16,384	4010	25,098
# synapses	8,192,000	76,624,800	5,588,000	138,357,544
2D Mesh config. (Y×X) [5,6]	4×4	8×8	4×4	10× 10
3D Mesh config. (Z×Y×X) [5,6]	2×2×4	4×4×4	2×2×4	2×6×10
Link defect rates	5%,10%, 15%, 20%	5%,10%, 15%, 20%	5%,10%, 15%, 20%	5%,10%, 15%, 20%
Neuron's spares	0	0	86	5622 (3D) / 502 (2D)
Neuron's defects	0	0	8, 16, 32, 64, 86	16, 32, 64, 128, 256
Multi-chip 2D-NoC	2 chips of 2×4	4 chips of 4×4	2 chips of 2×4	4 chips of 5× 5
Multi-chip 3D-NoC	2 chips of 2×2×2	4 chips of 4×2×2	2 chips of 2×2×2	2 chips of 2×6×5
Multi-chip link delay[8]	10× on-chip link	10× on-chip link	10× on-chip link	10× on-chip link

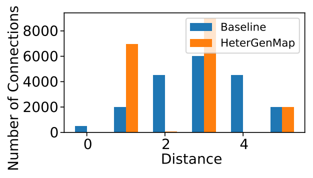
[1] **S#1**: Input(2000)-FC(2000)-FC(2000)-FC(96)
[2] **S#2**: Input(2000)-FC(10000)-FC(5000)-FC(1300)-FC(84)
[3] **MLP-MNIST** [33]: Input(28×28)-FC(2000)-FC(2000)-FC(10)
[4] **CNN-CIFAR10**: Input(32 × 32 × 3) - [Conv, Pool] × 16 - [Conv, Pool] × 32 - Conv × 8 - FC(10);
[5] Input spikes are sent from the interface node: node(0,0,0) for 3D-NoC and node(0,0) for 2D-NoC.
[6] Final output spikes of the last layer are sent back to the interface node: node(0,0,0) for 3D-NoC and node(0,0) for 2D-NoC.
[8] The ratio of off-chip/on-chip links are hypothesis. Depending on the system, these numbers can be varied.



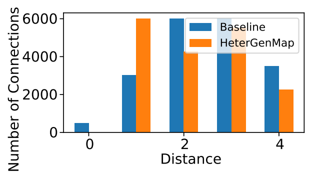
(a) Evolution of **MLP-MNIST** in 2D Mesh



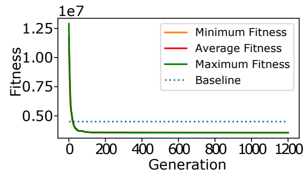
(b) Evolution of **MLP-MNIST** in 3D Mesh



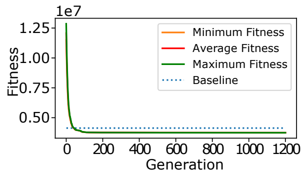
(c) Distribution of connection length of the final solution for **MLP-MNIST** in 2D Mesh



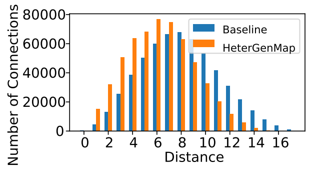
(d) Distribution of connection length of the final solution for **MLP-MNIST** in 3D Mesh



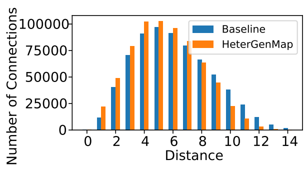
(e) Evolution of **CNN-CIFAR10** in 2D Mesh



(f) Evolution of **CNN-CIFAR10** in 3D Mesh



(g) Distribution of connection length of the final solution for **CNN-CIFAR10** in 2D Mesh



(h) Distribution of connection length of the final solution for **CNN-CIFAR10** in 3D Mesh

FIGURE 11: Evolution process of the realistic SNN applications

rapidly improve which makes the final solution much better than using a randomly generated initial population.

C. FAULT-TOLERANT LINKS MAPPING

Another feature of *HeterGenMap* is allowing heterogeneous communication links. Indeed, it supports faulty links/routing units. In this section, we insert defective links randomly with a probability of 5%,10%, 15%, and 20% to compute the efficiency of the method. All of the GA simulations are performed with the baseline being part of the initial solutions as explained in Section IV-B2.

Figures 12 and 13 illustrate the results of fault-tolerant mapping in *HeterGenMap* in comparison to non-fault tolerant methods. In summary, the fault-tolerant flavor of *HeterGenMap* offers lower finesses (less communication) in comparison to the baseline or the genetic algorithm without fault awareness. Compared to the baseline, the fault-tolerant *HeterGenMap* reduces the communication cost by 3.41-31.34%. Meanwhile, without fault-tolerant support, the original GA suffers a 1.41-14.78% increment in communication cost in comparison to its fault-tolerant version.

Moreover, the connection distribution also illustrates the domination of our approach in comparison to the baseline or our own method without fault awareness. By reducing the long-distance connections, the performance can be significantly better in the fault-tolerant version.

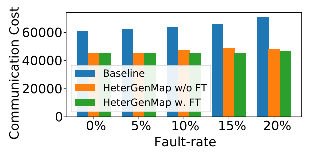
In terms of connection length distribution, we can easily observe that with the fault-tolerance, *HeterGenMap* has less connection with long distances in comparison to other approaches.

D. FAULT-TOLERANT NEURONS MAPPING

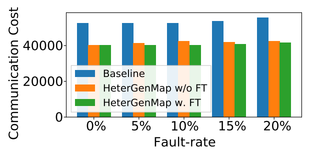
As mentioned earlier, *HeterGenMap* also supports mapping under defective neuron situations. Here, we allow the system to have spare neurons and use them as replacements for the faulty ones. As S#1 and S#2 have no spare neurons, we only simulate MLP-MNIST and CNN-CIFAR10 in this evaluation.

TABLE 3: Fitness value of *HeterGenMap* platform with and without initial members.

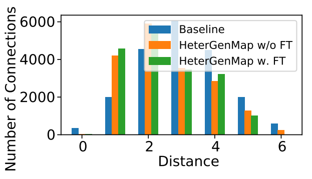
Benchmark		S#1		S#2		MLP-MNIST		CNN-CIFAR10	
NoC Configuration		2D	3D	2D	3D	2D	3D	2D	3D
Baseline		60,976	52,640	1,399,044	940,028	60,140	52,090	3,881,988	3,658,078
HeterGenMap	w/o. Initial Member	45,057	40,218	1,161,659	836,295	44,039	40,037	3,742,972	3,551,830
	w. Initial Member	**44,459**	**40,168**	**1,136,264**	**829,975**	**44,032**	**40,018**	**3,135,305**	**3,134,472**



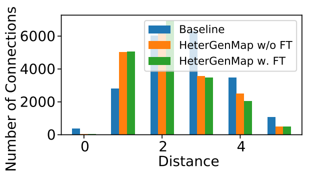
(a) Final communication cost for **S#1** in 2D Mesh.



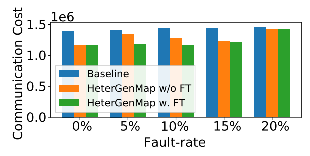
(b) Final communication cost for **S#1** in 3D Mesh.



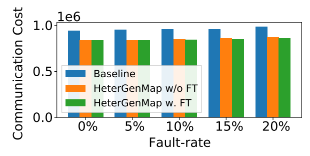
(c) Distribution of connection length of the final solution for **S#1** in 2D Mesh under 5% defect rate.



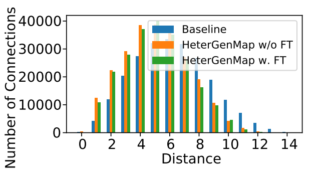
(d) Distribution of connection length of the final solution for **S#1** in 3D Mesh under 5% defect rate.



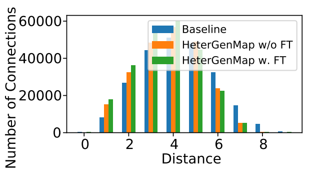
(e) Final communication cost for **S#2** in 2D Mesh.



(f) Final communication cost for **S#2** in 3D Mesh.

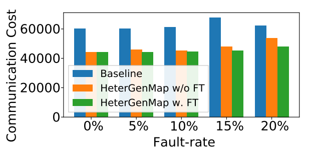


(g) Distribution of connection length of the final solution for **S#2** in 2D Mesh under 5% defect rate.

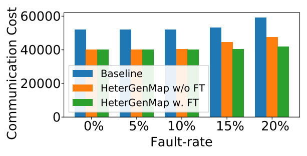


(h) Distribution of connection length of the final solution for **S#2** in 3D Mesh under 5% defect rate.

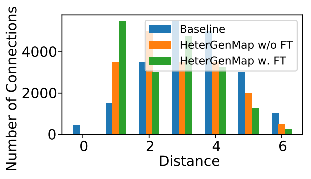
FIGURE 12: Final fitness results and distribution of connection lengths of the fault-tolerant mappings for synthetic applications under faulty links.



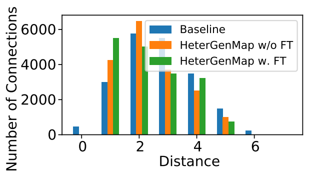
(a) Final communication cost for **MLP-MNIST** in 2D Mesh.



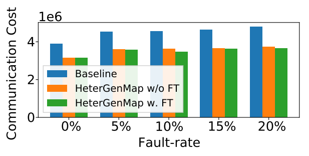
(b) Final communication cost for **MLP-MNIST** in 3D Mesh.



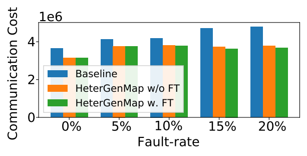
(c) Distribution of connection length of the final solution for **MLP-MNIST** in 2D Mesh under 5% defect rate.



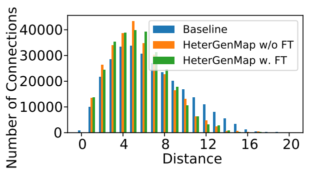
(d) Distribution of connection length of the final solution for **MLP-MNIST** in 3D Mesh under 5% defect rate.



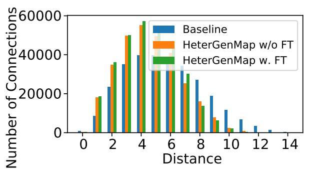
(e) Final communication cost for **CNN-CIFAR10** in 2D Mesh.



(f) Final communication cost for **CNN-CIFAR10** in 3D Mesh.



(g) Distribution of connection length of the final solution for **CNN-CIFAR10** in 2D Mesh under 5% defect rate.



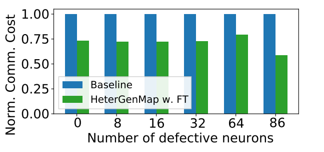
(h) Distribution of connection length of the final solution for **CNN-CIFAR10** in 3D Mesh under 5% defect rate.

FIGURE 13: Final fitness results and distribution of connection lengths of the fault-tolerant mappings for realistic applications under faulty links.

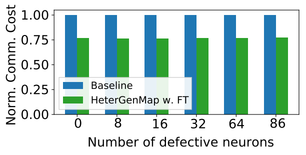
Here, for MLP-MNIST, the system has 86 spare neurons and we insert faults randomly in 8, 16, 32, 64, or 86 neurons to understand the impact of faults on the communication cost. For CNN-CIFAR10, the numbers of spare neurons are 622 and 502, for for 3D-NoC and 2D-NoC, respectively. We here insert faults randomly in 16, 32, 64, 128, and 256 neurons for this evaluation.

Figure 14 illustrates the communication cost under defective neuron situations. In summary, both *HeterGenMap* and the baseline can easily deal with defective neurons. The overall communication cost is similar between different numbers of defects because this type of defect only leads to some cores (clusters) with a lesser number of neurons. We also observe a drop in the communication cost of MLP-MNIST under 86 defective neurons while the communication costs are similar between other cases. This can be explained by the 86 defective neurons case is the number of spares which leads to the case where it is difficult to find a good solution that offers low communication cost. In other cases, the number of defective neurons is smaller than the number of spares, therefore, it allows more flexibility in the mapping process. Please also note that since the defects are randomly inserted;

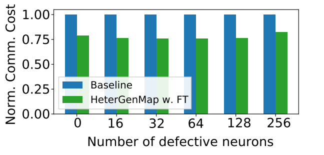
therefore, the result may be slightly different if we have different positions of defects.



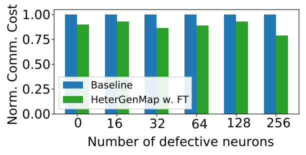
(a) Final communication cost for **MLP-MNIST** in 2D Mesh.



(b) Final communication cost for **MLP-MNIST** in 3D Mesh.



(c) Final communication cost for **CNN-CIFAR10** in 2D Mesh.

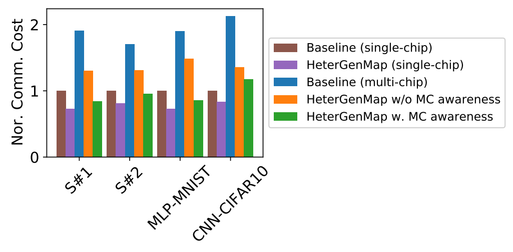


(d) Final communication cost for **CNN-CIFAR10** in 3D Mesh.

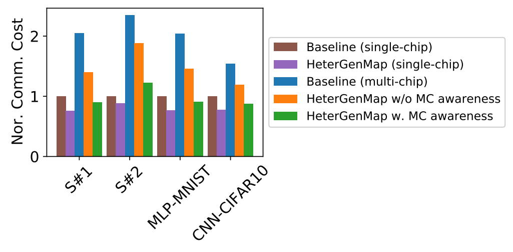
FIGURE 14: Normalized communication cost in the fault-tolerant mappings for realistic applications under defective neurons.

E. MULTI-CHIP MAPPING

Another feature of *HeterGenMap*'s heterogeneous mapping is allowing multi-chip systems because each communicating link can have different costs. In this section, we split the system into several chips as shown in Table 2 to compute the efficiency of the method.



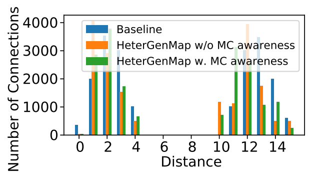
(a) 2D Mesh



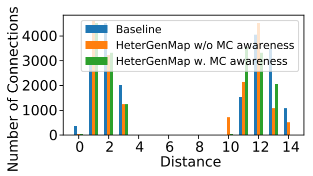
(b) 3D Mesh

FIGURE 15: Final fitness results of the multi-chip mappings.

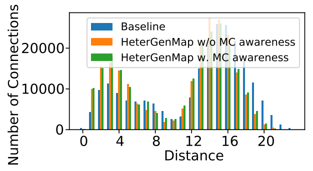
In terms of summarized communication cost, Figure 15 shows that with multi-chip awareness, *HeterGenMap* reduces it by 34.2-45.56% when compared to the baseline. Even when compared to the baseline in a single-chip setting, our method



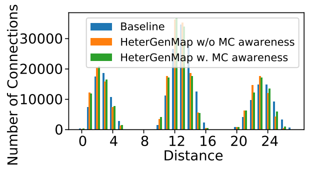
(a) Distribution of connection length of the final solution for **S#1** in 2D Mesh.



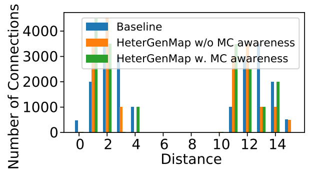
(b) Distribution of connection length of the final solutions for **S#1** in 3D Mesh.



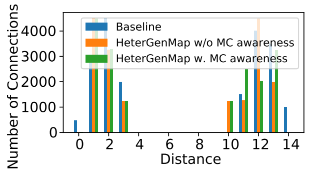
(c) Distribution of connection length of the final solution for **S#2** in 2D Mesh.



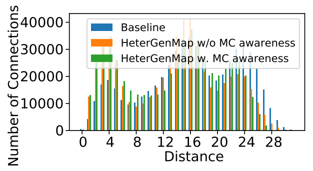
(d) Distribution of connection length of the final solutions for **S#2** in 3D Mesh.



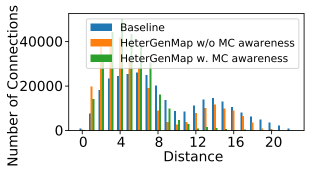
(e) Distribution of connection length of the final solutions for **MLP-MNIST** in 2D Mesh.



(f) Distribution of connection length of the final solutions for **MLP-MNIST** in 3D Mesh.



(g) Distribution of connection length of the final solution for **CNN-CIFAR10** in 2D Mesh.



(h) Distribution of connection length of the final solutions for **CNN-CIFAR10** in 3D Mesh.

FIGURE 16: Distribution of connection length of the final solutions for multi-chip mapping.

still provides a much better communication cost. The distribution of connection length is shown in Figure 16 where we can observe that with multi-chip awareness, *HeterGenMap* reduces the maximum length significantly. For instance, for MLP-MNIST, *HeterGenMap* with MC awareness has a maximum connection length of 12 and 11 for 2D and 3D Mesh, respectively. Meanwhile, the maximum connection lengths for baseline are 15 and 14. We also can observe that in terms of connection length distribution, thanks to the awareness of the long latency inter-chip wires, *HeterGenMap* rarely ends up with a considerable amount of long latency routing paths.

F. AVERAGE NUMBER OF HOPE COMPARISON

In this section, we compare the average number of hops as the main result of the mapping approach. Here, we extract the result from NeuMap [28] where it is compared to SNEAP [10] and SpiNeMap [18]. Here, we evaluate the average number of hops in connection with the mapping results for six different networks (CNN-CIFAR10, CNN-Fashion-MNIST, LetNet5-CIFAR10, LetNet-5-MNIST, MLP-Fashion-MNIST, and MLP-MNIST) as can be seen in the Table 4. The results for NeuMap [28], SNEAP [10] , and SpiNeMap [18] are for 8×8 2D-NoC and 256 neurons/core. In our *HeterGenMap* evaluations, we use 8×8 and $4 \times 4 \times 4$ NoC and 256 neurons/core. We can observe that, although NeuMap is better than *HeterGenMap* for CNN models, *HeterGenMap* is better for MLP. In comparison with SpiNeMap or SNEAP, we can see we can reduce 27.89% to 73.21% of the average hops.

On the other hand, with our main focus being on heterogeneous mapping, none of the existing works can adapt to fault-tolerant or multi-chip mapping. As we demonstrated in the previous section, without consideration of faults or multi-chip links, the average hopes or communication cost can be increased significantly.

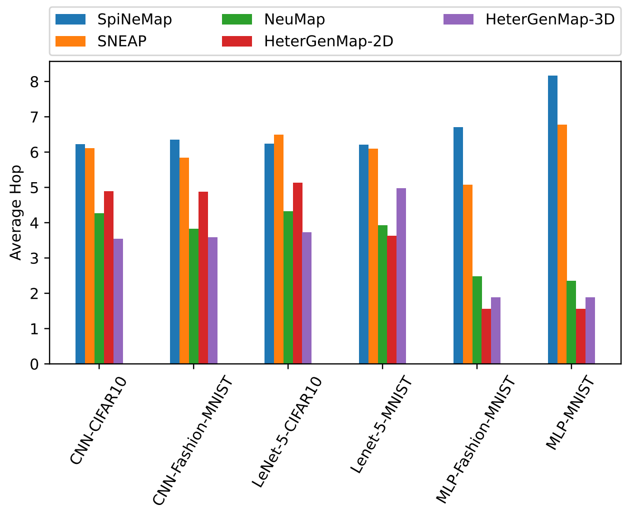


FIGURE 17: Comparison of the average of hops between *HeterGenMap* and the state-of-the-art works.

G. VALIDATING MAPPING WITH RTL SIMULATION

In the previous sections, we evaluated the communication cost with *HeterGenMap* and compared it to linear mapping. However, since it is a mathematical model, it could have a gap between the cost function and reality.

To validate the efficiency of the mapping, we apply the results of the mapping to our neuromorphic system from [22]. Since our hardware only supports fully connected models, we only evaluate the MLP-MInST benchmark. Table 5 illustrates the execution time of the system under two mapping. Here, we observe that with MLP-MNIST, *HeterGenMap* reduces the execution time by 77.87% and 63.10% for 2D- and 3D-Mesh NoC, respectively. These significant drops are due to two main reasons. First, the number of communications in *HeterGenMap* is significantly lower than linear mapping. Therefore, there is less traffic in the NoC which makes the NoC less congested. Since there is less congestion, the packets are delivered must faster which the *HeterGenMap* runs faster. We also observed that the system has communication bottlenecks since the LIF (Leaky-Integrate-and-Fire) neuron will need to wait for the spike to deliver to be able to compute which makes computation have no impact on the overall latency. Second, the number of long traversal paths is smaller which makes the long latency paths less frequent. Because the system will wait for all spikes to be delivered within the time-step to forward to the next time-step, this also has a significant impact on the overall execution times. Because of these two reasons, the *HeterGenMap* system yields much lower execution time in comparison to linear mapping.

V. DISCUSSION AND OUTLOOK

In this work, a genetic algorithm framework for mapping has been proposed with promising results. However, some limitations still need to be addressed:

- Although GA offers optimized solutions, one of its major drawbacks is time and space complexity. Because of this, our GA could not properly work with large-scale models such as ResNet. Therefore, we could observe a certain limitation of the scalability of the approach. However, we would like to emphasize that with an extremely large-scale network, linear mapping will fit large-scale models much better due to its low complexity. Here, we can certainly improve the GA by limiting the crossover or mutation which allows it to run faster.
- Compared to other approaches, our approach has similar complexity as the meta-heuristic algorithms [13], [21]. However, it has higher computation complexity (in both time and space) than linear mapping or non-meta-heuristic approaches [3], [4]. This can be understandable as Genetic Algorithms require a certain number of members in a population and a certain number of generations to converge. The trade-off of this work is the ability to tune the solution to your desired objective. Since *HeterGenMap* can re-use previously generated solutions as initial ones as in Algorithm 1, we can cooperate with this platform with pre-existing ones to further optimize them.
- The major constraint of this work is the neural network must be grouped efficiently. In other words, we must classify the neurons into a group that can be exchangeable between neurons. This constraint can be easily satisfied by layer-based SNNs such as feed-forwarded or convolution ones; however, for liquid state machines or reservoir computing, the grouping approach can be inefficient.
- Other outlooks of our work lie in its flexibility in allowing potential extensions. Unlike other approaches where the design of the mapping is fixed, our genetic algorithm framework offers several extensions such as:
    - *Different cost functions*: Although we have proposed several cost functions, we can observe that the designers can change the cost function to fit into their constraints. Communication may not be the critical issue and they may consider different

TABLE 4: Configuration of different benchmarks. FC: Fully Connected; Conv: Convolution; Pool: Pooling; In: Input layer.

Benchmark	Configuration
CNN-CIFAR10	In(32 × 32 × 3) - Conv × 16 - [Conv, Pool] × 32 - Conv × 8 - FC(10)
CNN-Fashion-MNIST	In(28 × 28) - [Conv, Pool] × 8 - [Conv, Pool] × 16 - FC(128) - FC(10)
LetNet-5-CIFAR10	In(32 × 32 × 3) - [Conv, Pool] × 6 - [Conv, Pool] × 16 - FC(120) - FC(84) - FC(10)
LetNet-5-MNIST	In(28 × 28) - [Conv, Pool] × 6 - [Conv, Pool] × 16 - FC(120) - FC(84) - FC(10)
MLP-Fashion-MNIST	In(28 × 28) - FC(256) - FC(128) - FC(10)
MLP-MNIST	In(28 × 28) - FC(400) - FC(10)

TABLE 5: Average execution time in clock cycles of the RTL simulations

Configuration	NoC Topo	Baseline	HeterGenMap	Reduction
MLP-MNIST	2D-Mesh	207,857	45,995	77.87%
	3D-Mesh	375,458	138,536	63.10%

targets such as temperature, reliability, area cost, and so on.

-- *Different crossover methods*: As we mentioned earlier, the crossover can be done in the core dimension and different ways: multi-points, randomly, etc.
-- *Using the output of NoC simulation as a cost value*: Instead of modeling mathematical equations, we can call a NoC simulator to emulate the operation of the system. This gives us the most accurate results. However, since it will take a huge amount to execute, it is not ideal.
-- *Multi-objective optimization*: Besides a single objective as we presented above, having multiple objectives (for example, communication & temperature) optimized simultaneously can also be another extension. Approaches such as Non-Dominant Sorting GA (NSGA-II) [34] can be applied to extend the GA framework.
-- *Adopting this work in Neuromorphic Systems:* In this work, we already provided the simulation for the RTL model to validate the efficiency of the mapping. The next step of this is to bring the solution into the neuromorphic system such as Loihi [2] or our neuromorphic design [9].

Because of these potential extensions, several unanswered questions could be addressed in future research.

- While dividing the neurons into groups for computing the GA can reduce the complexity, there are some models such as liquid state machine where the group is unclear. Therefore, the proposed GA framework may not be suitable for those applications or we need a clustering algorithm that provides us with the grouping.

Besides the compared work, there are existing work on multi-chip mapping and fault-tolerant mapping:

- Multi-chip mapping for non-SNN applications is also considered in several works [17], [35]. Here, the major concern is due to long-latency inter-chip communication, it may fail to speed up if the mapping is not efficient. In [17], [35], the authors empirically study the multi-chip implementation for ResNet-50 and found out it stops improving from 8-chip upward because inter-chip communication becomes the bottleneck. Here, the neural network is split up based on the layers. Compared to our work, we here model the inter-chip communication with a certain penalty which can help introduce the concept of inter-chip communication bottle-neck. We also would like to note that it is sometimes not feasible to control the number of chips inside the system because manufactured chips have their computing limitation. In [36], the authors use multi-chips to extend the size of the model. Here, each chip is considered to operate a fixed size of image, and to deal with larger sizes, a 2D mesh of the chips is considered. The authors introduce the concept of pixel redundancies to avoid inter-chip communications. The inter-chip communications are via a serial connection which can be a huge bottleneck for communication-intensive applications. By naturally exploiting the natural structure of the 2D image, there is no mapping algorithm needed; however, we can certainly observe the bottleneck issue of multi-chip mapping which is modeled and optimized in this work. In [37], the authors from Intel Lab introduce the mapping algorithm for multiple Loihi-chips systems. The main target of this algorithm is to reduce the inter-chip communication. Here, the cut penalty matrix is introduced to model the extra communication latency due to spreading the SNN into multiple chips. However, in this work, a greed-algorithm approach is adopted which makes our method more optimal due to the nature of the genetic algorithm. However, compared to this work, our method will be more computationally intensive. Another common approach is to unitize reinforcement learning in the mapping process [38], [39]. While the work in [38] shows some domination in comparison to GA; [39] still shows more room for improvement. In summary, this type of optimization approach can be interesting; however, one major problem is it requires actual implementations or full-system simulations to perform. Meanwhile, our method and most SNN mappings [3], [19], [21], [37] are based on modeling of the communication and computation. The major trade-off will be quick adaptation to different networks and systems. Furthermore, in terms of fault-tolerant mapping, reinforcement learning-based approaches must deal with case-by-case; however, our work can be easily adapted to different scenarios.
- On the other hand, fault-tolerant mapping for a NoC-based system is a well-known issue and has been addressed in several existing works. In [18], the authors

proposed adding spare cores and swapping the core functionalities from faulty core to healthy core as the method. In our previous work [13] also proposed adding spare neurons and migrating fault neurons to the spare one as a fault-tolerant method. However, in this type of approach, they did not consider the impact of defective routing modules on communication. This type of problem is addressed in this work as we model the communication cost using the shortest path between a source and a sink. In [40], the author proposed a mapping approach with faulty neuron consideration. However, unlike conventional computations, spiking neural network models consist of a large number of neurons and connections which will be addressed in our work. Furthermore, as we stated in Algorithm 1, our *HeterGenMap* platform can take a mapping solution as an initial member of the initial population. Therefore, our approach can reuse the other approaches and could further improve the efficiency of them.

VI. CONCLUSION

In this paper, we presented a Genetic Algorithm-based methodology to map multiple-layer SNN models into a NoC-based neuromorphic system. To reduce the complexity, we introduced an approach that uses layers as groups where the neurons in the group are exchangeable. Furthermore, the GA is performed with customized crossover and mutation approaches. To model heterogeneous designs, the GA platform allows the modeling of multi-chip systems, fault awareness, and different cluster sizes within the mapping process. Future works of the framework could be extending the selection of NoC topology or modeling the thermal/power properties of the NoC. With the change of NoC topology, the impact of faults and the communication costs can be varied. On the other hand, the temperature issue could be very important for large-scale SNNs, especially 3D-IC-based systems. Moreover, as we discussed, multi-objective optimization could also be another direction of this research.

ACKNOWLEDGMENT

This work is supported in part by the University of Aizu Competitive Research funding, Ref. 2023-P26.

REFERENCES

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[3] F. Akopyan *et al.*, "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.

[4] S. B. Furber *et al.*, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[5] B. V. Benjamin *et al.*, "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.

[6] J. Schemmel *et al.*, "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] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, p. e47314, Aug. 2019.

[8] H. Hazan *et al.*, "BindsNET: A machine learning-oriented spiking neural networks library in Python," *Frontiers in Neuroinformatics*, vol. 12, p. 89, 2018.

[9] M. Ogbodo, T. Vu, K. Dang, and A. Ben Abdallah, "Light-weight spiking neuron processing core for large-scale 3D-NoC based spiking neural network processing systems," in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2020, pp. 133–139.

[10] S. Li *et al.*, "SNEAP: A fast and efficient toolchain for mapping large-scale spiking neural network onto NoC-based neuromorphic platform," *arXiv preprint arXiv:2004.01639*, 2020.

[11] E. L. Lawler, "The quadratic assignment problem," *Management science*, vol. 9, no. 4, pp. 586–599, 1963.

[12] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 370–375.

[13] K. N. Dang, N. A. V. Doan, and A. B. Abdallah, "Migspike: A migration based algorithms and architecture for scalable robust neuromorphic systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 602–617, 2022.

[14] R. V. W. Putra, M. A. Hanif, and M. Shafique, "Respawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[15] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Neuron fault tolerance in spiking neural networks," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 743–748.

[16] S. Furber, S. Temple, and A. Brown, "On-chip and inter-chip networks for modeling large-scale neural systems," in *2006 IEEE International Symposium on Circuits and Systems*, 2006, pp. 4 pp.–.

[17] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[18] A. Balaji *et al.*, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2019.

[19] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "Neutrams: Neural network transformation and co-design under neuromorphic hardware constraints," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–13.

[20] A. Das, Y. Wu, K. Huynh, F. DellAnna, F. Catthoor, and S. Schaafsma, "Mapping of local and global synapses on spiking neuromorphic hardware," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 1217–1222.

[21] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. DellAnna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2020.

[22] A. Ben Abdallah and K. N. Dang, "Toward robust cognitive 3d brain-inspired cross-paradigm system," *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.690208

[23] C. Frenkel *et al.*, "A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 1, pp. 145–158, 2018.

[24] A. Balaji, P. K. Huynh, F. Catthoor, N. D. Dutt, J. L. Krichmar, and A. Das, "Neusb: A scalable interconnect architecture for spiking neuromorphic hardware," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–15, 2023.

[25] 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.

[26] X. Jin, "Parallel simulation of neural networks on spinnaker universal neuromorphic hardware," Ph.D. dissertation, The University of Manchester (United Kingdom), 2010.

[27] G. Kim *et al.*, "Optimal distribution of spiking neurons over multicore neuromorphic processors," *IEEE Access*, vol. 8, pp. 69 426–69 437, 2020.

[28] C. Xiao, J. Chen, and L. Wang, "Optimal mapping of spiking neural network to neuromorphic hardware for edge-ai," *Sensors*, vol. 22, no. 19, p. 7248, 2022.
[29] O. Jin, Q. Xing, Y. Li, S. Deng, S. He, and G. Pan, "Mapping very large scale spiking neuron network to neuromorphic hardware," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 419–432.
[30] P. V. Bhanu *et al.*, "Fault-tolerant network-on-chip design with flexible spare core placement," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 1, Jan. 2019.
[31] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.
[32] S. Carrillo, J. Harkin, L. J. McDaid, F. Morgan, S. Pande, S. Cawley, and B. McGinley, "Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2451–2461, 2013.
[33] P. U. Diehl *et al.*, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Oct 2016, pp. 1–8.
[34] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
[35] Y. S. Shao, J. Cemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: scaling deep-learning inference with chiplet-based architecture," *Communications of the ACM*, vol. 64, no. 6, pp. 107–116, 2021.
[36] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Hyperdrive: A multi-chip systolically scalable binary-weight cnn inference engine," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 309–322, 2019.
[37] C.-K. Lin, A. Wild, G. N. Chinya, T.-H. Lin, M. Davies, and H. Wang, "Mapping spiking neural networks onto a manycore neuromorphic architecture," *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 78–89, 2018.
[38] N. Wu, L. Deng, G. Li, and Y. Xie, "Core placement optimization for multi-chip many-core neural network systems with reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 2, pp. 1–27, 2020.
[39] J. Choudhary, J. Soumya, and L. R. Cenkeramaddi, "Raman: reinforcement learning inspired algorithm for mapping applications onto mesh network-on-chip," in *2021 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*. IEEE, 2021, pp. 52–58.
[40] L. Liu, C. Wu, C. Deng, S. Yin, Q. Wu, J. Han, and S. Wei, "A flexible energy-and reliability-aware application mapping for noc-based reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2566–2580, 2014.

KHANH N. DANG is currently an Associate Professor in the Department of Computer Science and Engineering at the University of Aizu. He received his Ph.D. from the University of Aizu and his M.Sc. from the University of Paris XI. His research interests include Network-on-Chips, 3D-ICs, neuromorphic computing, low-power, and fault-tolerant systems.

NGUYEN ANH VU DOAN is currently with Infineon Technologies AG, Germany. He was previously with Fraunhofer IKS as a Senior Scientist, after working as a Postdoctoral Researcher, first with the Amano lab, Keio University (Japan), then with the Chair of Integrated Systems, Technical University of Munich (Germany). He received a Ph.D. degree in electrical engineering from the Université libre de Bruxelles (Belgium) in 2015. His research interests include design space exploration, design automation, multiobjective optimization, and multicriteria decision aiding.

NGO-DOANH NGUYEN is currently a Master student in the Graduate School of Computer Science and Engineering at the University of Aizu, Aizu-Wakamatsu, Japan. He previously worked at Vietnam National University, Hanoi (2018 – 2022) as a research engineer on System Integration and VLSI design for Artificial Intelligence. His research interests include hardware/software co-design and verification, and low-power solutions for artificial intelligence.

ABDERAZEK BEN ABDALLAH (Senior Member, IEEE) received the Ph.D. degree in computer engineering from The University of Electro-Communications, Tokyo, in 2002. From April 2014 to March 2022, he was the Head of the Computer Engineering Division, The University of Aizu, Japan. Since April 2022, he has been the Dean of the School of Computer Science and Engineering, The University of Aizu. He is currently a Full Professor with The University of Aizu. He is the author of four books, four registered and eight provisional Japanese patents, and more than 150 publications in peer-reviewed journal articles and conference papers. His research interests include adaptive/self-organizing systems, brain-inspired computing, interconnection networks, and AI-powered cyber-physical systems. He is a Senior Member of ACM.