Mapping Large-Scale Spiking Neural Network on Arbitrary Meshed Neuromorphic Hardware

Ouwen Jin, Qinghui Xing, Zhuo Chen, Ming Zhang, De Ma, Ying Li, Xin Du, Shuibing He, Shuiguang Deng, *Senior Member, IEEE*, and Gang Pan, *Senior Member, IEEE*

***Abstract*—Neuromorphic hardware systems—designed as 2D-mesh structures with parallel neurosynaptic cores—have proven highly efficient at executing large-scale spiking neural networks (SNNs). A critical challenge, however, lies in mapping neurons efficiently to these cores. While existing approaches work well with regular, fully functional mesh structures, they falter in real-world scenarios where hardware has irregular shapes or non-functional cores caused by defects or resource fragmentation. To address these limitations, we propose a novel mapping method based on an innovative space-filling curve: the Adaptive Locality-Preserving (ALP) curve. Using a unique divide-and-conquer construction algorithm, the ALP curve ensures adaptability to meshes of any shape while maintaining crucial locality properties—essential for efficient mapping. Our method demonstrates exceptional computational efficiency, making it ideal for large-scale deployments. These distinctive characteristics enable our approach to handle complex scenarios that challenge conventional methods. Experimental results show that our method matches state-of-the-art solutions in regular-shape mapping while achieving significant improvements in irregular scenarios, reducing communication overhead by up to 57.1%.**

***Index Terms*—Neuromorphic computing, spiking neural networks (SNN), network on chip (NOC), mapping.**

I. Introduction

NEUROMORPHIC computing is an emerging research field focused on developing artificial intelligence that consumes less energy. Central to this field are Spiking Neural Networks (SNNs), regarded as the next generation of neural networks. These networks are designed to replicate the functions of biological brains, employing neuron and synapse models for processing information in both spatial and temporal dimensions. A critical aspect of this research is the development of neuromorphic hardware specifically designed to run SNNs. This includes various platforms like DYNAP-SE [1], TrueNorth [2], Neurogrid [3], SpiNNaker [4], Loihi [5], Tianji [6], and Darwin [7], each uniquely engineered to leverage the energy-efficient qualities of spike-based computing.

Received 16 March 2025; revised 11 July 2025; accepted 13 August 2025. Date of publication 25 August 2025; date of current version 26 September 2025. This work was supported in part by the National Science and Technology Program under Grant 2024YDLN0005, in part by the Natural Science Foundation of China under Grant 61925603 and Grant 62172361, and in part by the Major Projects of Zhejiang Province under Grant LD24F02001. Recommended for acceptance by A. Li. *(Corresponding author: Gang Pan.)*

The authors are with the Zhejiang University, Hangzhou 310058, China (e-mail: gpan@zju.edu.cn).

Digital Object Identifier 10.1109/TPDS.2025.3601993

A 2D-mesh structure Network-on-Chip (NOC) [8] is a widely adopted design in many neuromorphic hardware systems. This architecture features an array of neurosynaptic cores, such as the crossbars in Loihi [5] and ARM cores in SpiNNaker [4], which are dedicated to storing synaptic weights and simulating neural activities in parallel. The 2D-mesh layout not only facilitates easy expansion to accommodate larger networks but also proves highly effective for the free flow of spiking messages.

The deployment of SNN applications on such hardware requires careful mapping as a crucial initial step. This process entails two main phases: first, partitioning the network's neurons into hardware-compatible clusters, and then strategically positioning these clusters on computing cores. By placing strongly connected neurons on physically adjacent cores based on neural connectivity patterns, this approach minimizes inter-core communication overhead and optimizes overall system efficiency.

Researchers have developed various mapping approaches to address these challenges, including PACMAN [9], PyCARL [10], SpiNeMap [11], DFSynthesizer [12], eSpine [13] and our previous works [14], [15]. These studies have conclusively shown that mapping configurations significantly influence multiple aspects of SNN application performance, from power consumption and processing latency to system throughput.

The development of large-scale neuromorphic hardware brings a new challenge: mapping SNNs to meshes with irregular shapes. This requirement emerges from two real-world scenarios: (i) Neuromorphic hardware systems serve as platforms handling multiple concurrent SNN workloads. These tasks have varying resource needs, arrival times, and durations, leading to fragmentation over extended service periods. (ii) Large-scale systems commonly experience defective cores.

Current mapping approaches, which typically assume ideal and regular mesh structures, falter when faced with the realities of practical hardware. In large-scale, many-core systems, challenges such as manufacturing flaws, operational wear, and resource fragmentation make irregular hardware topologies an inevitable reality. A primary source of this irregularity is the presence of non-functional, or 'Not Available' (N/A), cores. These are often the macroscopic outcome of low-level physical defects, with stuck-at faults being a prominent example—a challenge especially acute in emerging memristive systems [16]. The inability of existing methods to adapt to these conditions leads to suboptimal mapping solutions, causing increased communication overhead, reduced system efficiency, and performance bottlenecks. Developing a robust mapping strategy that

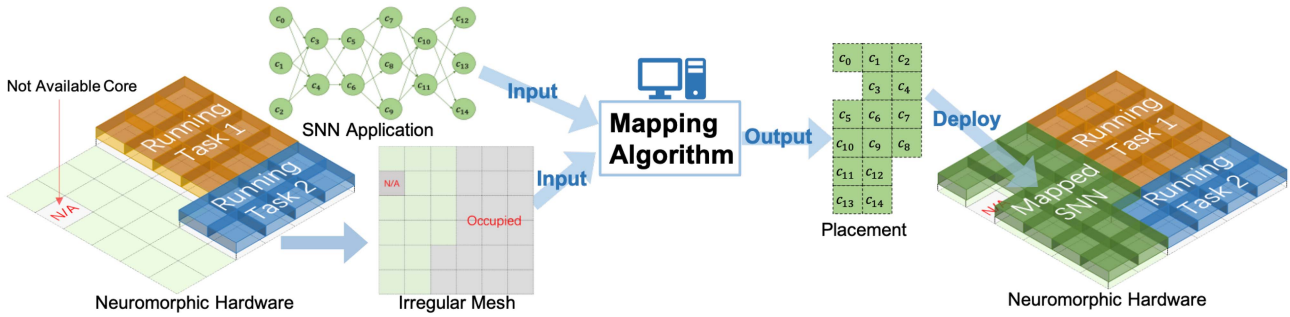


Fig. 1. Mapping SNN onto neuromorphic hardware with irregular meshes.

maintains high performance across these diverse and imperfect hardware configurations is therefore crucial, and it is precisely the problem our work is designed to address.

Fig. 1 illustrates the fundamental challenge this paper addresses: mapping SNNs onto neuromorphic hardware with irregular mesh shapes. The figure depicts an SNN application awaiting deployment on a neuromorphic hardware platform where two tasks are already running, occupying multiple computing cores. Additionally, one unoccupied core is unavailable due to a hardware fault. In this scenario, the mapping algorithm must strategically place neurons from the SNN onto the available computing cores within this irregular mesh configuration to achieve an optimal placement for deployment and execution.

To address this challenge, we propose an innovative two-stage mapping approach that combines the Adaptive Locality-Preserving (ALP) curve with the Force Directed (FD) algorithm. Our method first employs the ALP curve to establish a high-quality initial mapping, followed by the FD algorithm's local optimization to refine and finalize the mapping solution.

We propose the Adaptive Locality-Preserving (ALP) curve, a novel space-filling curve designed to adapt to meshes of any irregular shape, including those with internal gaps or non-functional cores. Crucially, the ALP curve not only offers this flexibility, but also retains and, in some cases, enhances the locality-preserving properties of the Hilbert curve. The construction algorithm is highly efficient, with a time complexity of $O(n \log n)$ for mapping n cores. This allows for mapping millions of cores in under one second, demonstrating the scalability required for mapping very large-scale SNNs to neuromorphic hardware.

The Force Directed (FD) algorithm [15] is an iterative optimization technique designed for large-scale SNN mapping challenges. In our novel approach, we integrate the ALP curve with the FD algorithm to complete the mapping process. The ALP curve initially provides a high-quality mapping placement, which is subsequently fine-tuned through the FD algorithm for local improvements. Our experimental results indicate that this method not only matches the performance of existing state-of-the-art approaches but also significantly expands mapping flexibility. This enhancement in adaptability holds profound practical significance, demonstrating the potential of our approach in diverse mapping tasks.

The main *contribution* of this paper are as follows:

- We propose the Adaptive Locality-Preserving (ALP) curve, a novel space-filling curve that uniquely adapts to *arbitrary irregular* shapes while preserving locality properties.
- We propose an SNN mapping approach that combines the ALP curve with the Force Directed (FD) algorithm, delivering exceptional adaptability across diverse mapping scenarios while maintaining high efficiency for large-scale implementations.
- We conduct thorough experiments to compare our method with current state-of-the-art techniques. This includes a comparison of our mapping algorithm with other leading algorithms and a detailed evaluation of the ALP curve's performance against various space-filling curves. The results show that our approach matches the best existing methods in regular mapping tasks and significantly reduces communication overhead in irregular mapping tasks.

The remainder of this paper is organized as follows. Section II provides the necessary background on neuromorphic hardware and space-filling curves. Section III reviews related works in SNN mapping and space-filling curve applications. Section IV introduces our novel Adaptive Locality-Preserving curve, followed by Section V which details our proposed mapping approach. Section VI presents comprehensive experiments and evaluations. Section VII discusses broader applications and implications of our work. Finally, Section VIII concludes the paper.

II. Background

Neuromorphic computing is receiving increasing attention for its brain-mimicking and energy-efficiency characteristics. The core technique of neuromorphic computing is Spiking Neural Network (SNN) [17], which has shown great potential and provides competitive results comparing to traditional Artificial Neural Network in many machine learning tasks, including vision classification, reinforcement learning, and robotic autonomous control [18].

Several emerging pieces of dedicated hardware have been designed for SNN implementation, such as TrueNorth [2], SpiNNaker [4], Loihi [5], Tianji [6], and Darwin [7]. These hardware systems are typically composed of a large number of neurosynaptic computing cores responsible for simulating neuronal and

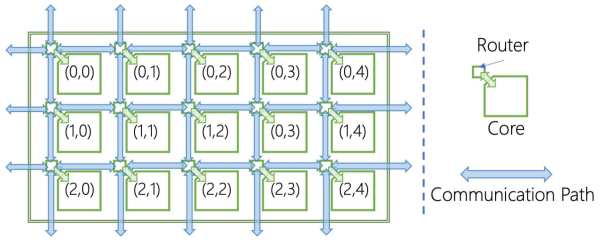


Fig. 2. 2D-Mesh NOC hardware model.

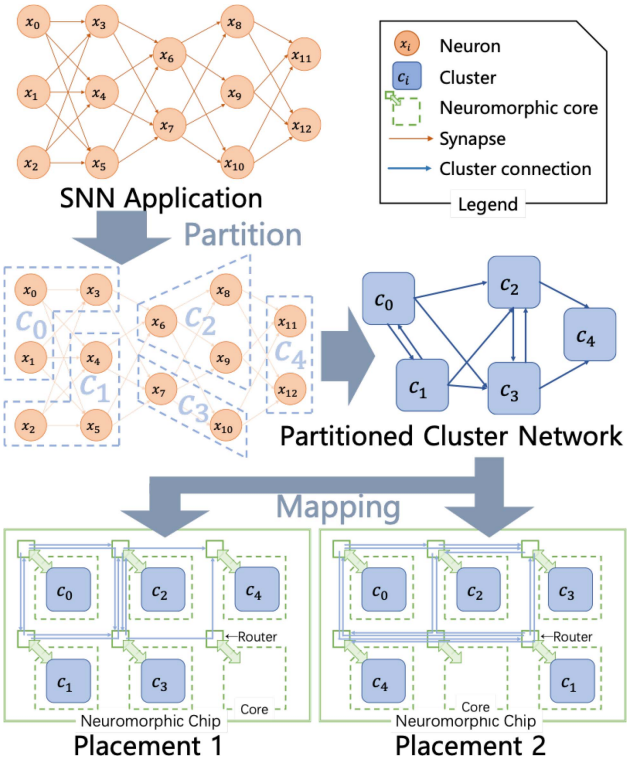


Fig. 3. Comprehensive overview of the SNN mapping process.

synaptic dynamics in parallel. The neurosynaptic computing cores are connected by a 2D-mesh structured Network-on-Chip (NOC) [8] for easy scalability to achieve massively parallel computing systems (as shown in Fig. 2).

To operationalize SNN applications on neuromorphic hardware, the SNN model must be mapped onto the hardware. Initially, the model is conceptualized as a graph: neurons are nodes, and synapses are edges. The SNN is then partitioned into clusters to match neurosynaptic core capabilities, forming a Partitioned Cluster Network (PCN). In this PCN, nodes represent neuron clusters, and edges indicate data transmission links between clusters. These clusters are then strategically placed on the computing cores. The mapping of the PCN is critical as it determines the pathways for spike information transfer on the NoC, directly influencing key performance metrics such as communication overhead, latency, and throughput.

Fig. 3 illustrates an example of mapping SNN applications onto neuromorphic hardware. First, the SNN is partitioned into PCN. In our case, the core is limited to accommodate a maximum of 3 neurons, and an SNN comprising 13 neurons x_{0-12} is partitioned into a PCN consisting of 5 clusters c_{0-4}. Subsequently, the nodes within the PCN are mapped onto the neurosynaptic cores. Fig. 3 shows two potential mapping results of the PCN, referred to as Placement 1 and Placement 2, onto a neuromorphic hardware configuration comprising 6 cores structured as a mesh with 2 rows and 3 columns. Placement 1 represents a superior solution compared to placement 2, as it exhibits shorter or equal path lengths for all connections in comparison to placement 2.

This paper addresses the problem described by the following equation:

$$\arg\min_F \mathcal{M}(F(M, G_{PCN})). \tag{1}$$

It seeks a mapping method $F()$ that inputs a mesh shape M and a graphically represented network G_{PCN}, outputting a mapping placement. The goal of this placement is to optimize performance metrics defined by $\mathcal{M}()$ such as total spiking distance or system power consumption.

The task of mapping Spiking Neural Networks (SNNs) onto hardware is a combinatorial optimization problem of NP-hard complexity [19]. This is because the end-to-end process encapsulates two well-known NP-hard challenges. First, partitioning the neuron graph into clusters is an instance of the Graph Partitioning problem. Subsequently, the placement of these clusters onto physical cores to minimize communication cost is a classic formulation of the Quadratic Assignment Problem (QAP). The computational intractability of finding optimal solutions for such problems necessitates the use of effective heuristics, which is the primary motivation for the approach presented in this paper.

III. Related Works

A. Related Works on SNN Mapping Problem

Prior researches have introduced diverse methodologies for SNN mapping. These methods, distinguished by their underlying algorithms, fall into three primary categories: heuristic-based, optimization-based, and SFC-based approaches.

Mapping SNNs to neuromorphic hardware, an NP-hard problem [19], commonly employs heuristic algorithms such as Particle Swarm Optimization (PSO). In PSO implementations, the search space has N dimensions, where $N = C \times V$ (C: number of neurons/clusters, V: number of cores), with each point representing a specific placement. Various approaches—PSOPART [20], Song et al. [21], SpiNeMap [11], and PyCARL [10], eSpine [13]—each offer distinct problem formulations and objective functions. PSOPART, for instance, applies PSO without partitioning, which increases computational complexity. SpiNeMap takes a different approach by binarizing the search space, while Song et al. leverage SDF^3 for throughput optimization. Similarly, eSpine employs PSO to optimize the placement of synapses on memristors based on their activation patterns, aiming to maximize the lifetime of memristive crossbar arrays by considering endurance variations. However, these methods share a critical limitation: their high algorithmic complexity results in poor efficiency. Designed primarily for small-scale scenarios (typically fewer than 100 cores), they

prove impractical for modern large-scale hardware with thousands of parallel cores, where generating a single mapping solution could require over a hundred hours.

Methods based on mathematical optimization, such as Mixed-Integer Linear Programming (MILP), can find provably optimal solutions but are computationally intractable for large-scale SNN mapping due to exponential time complexity. This scalability limitation is severe: finding an optimal mapping for even a mere 16-core system can exceed an hour of computation [22], [23]. This prohibitive cost is a widely recognized challenge, confirmed by comprehensive surveys of the field [24].

Many toolchains instead leverage faster, though sub-optimal, greedy algorithms. For example, PACMAN [9] uses a first-come-first-serve approach for core assignments, while the Corelet toolchain [25] adopts a layer-by-layer mapping strategy to reduce communication overhead. LCompiler [26], built specifically for Loihi [5], focuses on partitioning to manage Loihi's high computing energy costs. However, these greedy methods still have key limitations. Their iterative optimization processes can pose scalability challenges for large-scale applications, and they often cannot handle irregular resource availability, resulting in significant performance losses on irregular meshes or defective hardware.

In our previous work [15], we proposed an approach for large-scale neuromorphic hardware mapping that combined the Hilbert Space-Filling curve (HSFC) for initial element placement with a Force-Directed (FD) algorithm for refinement. This strategy demonstrated computational efficiency for mapping large-scale SNNs onto neuromorphic hardware, primarily due to the Hilbert curve's locality-preserving property. By placing connected neurons close together, the curve effectively reduced inter-core communication overhead. However, despite these achievements, we identified a critical limitation in the method's reliance on HSFC: its inability to construct curves on irregular shapes. This inflexibility presents challenges in handling complex real-world scenarios, which constrains its practical applications.

The challenge of mapping to non-ideal hardware is not unique to SNNs and has also been addressed in the context of non-spiking neural networks, with a primary focus on fault tolerance to maintain computational accuracy. For example, some studies propose redundancy-based mapping schemes, such as the greedy search approach presented by Yousuf et al. [16] that utilizes layer ensemble averaging to mitigate faults. Other works focus on encoding-based approaches, where workloads are allocated and tuned to counteract hardware defects, as discussed by Xia et al. [27]. While these methods effectively address device-level computational errors, our work targets a different challenge. We focus on the system-level problem of minimizing inter-core communication overhead for SNNs on geometrically irregular hardware, which is a distinct objective from preserving computational accuracy within individual cores.

B. Related Works on Space-Filling Curves

The Hilbert space-filling curve, due to its unique properties, has found widespread applications in many fields. However, its limitation of being constructible only on squares with side lengths that are powers of two has always been a challenge.

TABLE I
PROPERTIES OF SPACE-FILLING CURVES

	Non-Square Rectangle	Regions with Holes	Arbitrary Meshes	Specify Start/Endpoint	Locality Property
Hilbert [28]	×	×	×	×	Good
Rong's [29]	✓	×	×	×	Good
Evasion Curve [30]	×	✓	×	×	Good
Context-based [31]	✓	✓	✓	×	Bad
General SFC [32]	✓	×	×	✓	Fair
Dense Curve [33]	✓	✓	✓	✓	Bad
ALP[ours]	✓	✓	✓	✓	Good

Consequently, many studies have sought to enhance its flexibility by extending the Hilbert curve or designing its variants.

Table I summarizes several attempts to extend the Hilbert Curve. Rong et al. [29] introduced a Modified Hilbert Curve that expands the curve's application from squares to rectangles. Nair et al. [30] designed a strategy based on "avoidance", enabling the Hilbert curve to circumvent holes within shapes. Dafner and others proposed a Context-based space-filling curve [31], designed to construct filling curves within arbitrary contour shapes. However, this curve, based on a Hamiltonian cycle approach, completely abandons the important characteristic of locality inherent in the Hilbert curve.

Sasidharan et al. [32] introduce a method of partitioning the plane using a KD-TREE and then determining the order of leaf visitation to create a space-filling curve. However, we found that it cannot genuinely construct in any shaped region, as there are scenarios that a KD-TREE cannot divide, e.g., a complex shape with an internal unavailable area will lead to no legal partitioning. Furthermore, our experimental findings, reported in Section VI-C, show that its locality features are weaker than those of the Hilbert curve and ALP.

Ban et al. [33] present a method for generating a curve that densely covers any geometric domain. However, this curve does not prioritize locality, making it less effective for mapping problems.

In summary, there is still a need for a space-filling curve that can be flexibly constructed on various shapes while preserving the crucial characteristic of locality.

IV. The Proposed ALP Curve

A. Overview

Some space-filling curves, owing to their inherent locality properties, are effective in mapping SNNs onto 2D mesh architectures. However, existing space-filling curves lack the necessary flexibility for practical applications.

We introduce the Adaptive Locality-Preserving(ALP) curve, a novel space-filling curve designed to address this challenge. Specifically, the ALP curve is a mapping relationship generated by the ALP construction algorithm, transforming a 1D sequence into 2D mesh coordinates. This curve exhibits two critical features:

- *Flexibility:* The ALP curve can be constructed on two-dimensional meshes of any shape, even those with internal gaps or irregular boundaries.

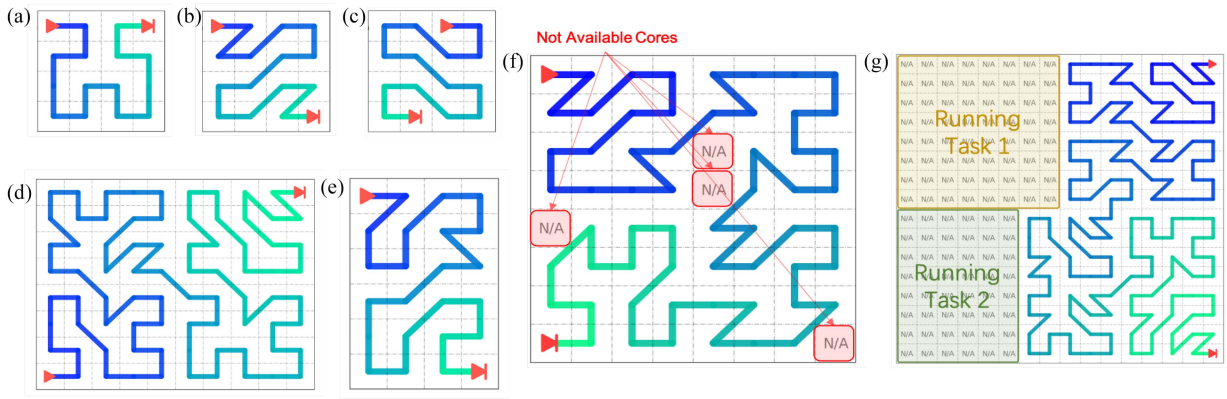


Fig. 4. Examples of the ALP curve. **(a-c)** The ALP curve on a standard 4x4 mesh with different start and end points. **(d-e)** The curve's adaptability to non-power-of-two rectangular meshes (8x10 and 6x4). **(f)** The curve navigating a mesh with internal non-functional cores, essential for fault tolerance. **(g)** The curve mapped onto a fragmented, irregularly shaped mesh, typical in multi-tasking environments.

- *Locality Preservation:* It maintains spatial closeness in the 2D space for points that are close in the 1D sequence.

Fig. 4 displays seven instances of the ALP curve applied to meshes of various configurations, each demonstrating the curve's adaptability. Fig. 4(a), (b) and (c), and **(c)**) depict the ALP curve on a standard 4×4 mesh, highlighting the curve's flexibility through different shapes achieved by varying start and end points. This ability to specify these points is particularly beneficial for neuromorphic hardware with specific input/output (I/O) requirements. By aligning the SNN's input and output layers with designated I/O ports, the ALP curve effectively minimizes communication overhead with external systems.

In Fig. 4(d) and (e), the ALP curve is applied to 8×10 and 6×4 meshes, respectively, demonstrating its adaptability to non-power-of-two rectangular meshes. This showcases the curve's versatility in accommodating various hardware layouts.

In Fig. 4(f) illustrates the curve's capability to navigate around internal gaps, a crucial feature for practical applications in large-scale parallel computing systems like neuromorphic hardware. Here, the ALP curve's flexibility ensures effective utilization of computational resources, even when some cores are non-functional due to malfunctions or other issues.

Fig. 4(g) shows the ALP curve mapped onto an irregularly shaped mesh, emphasizing its efficiency in multi-tasking environments where fragmented mesh regions are common. This example underscores the ALP curve's proficiency in managing and utilizing available computing resources in complex scenarios.

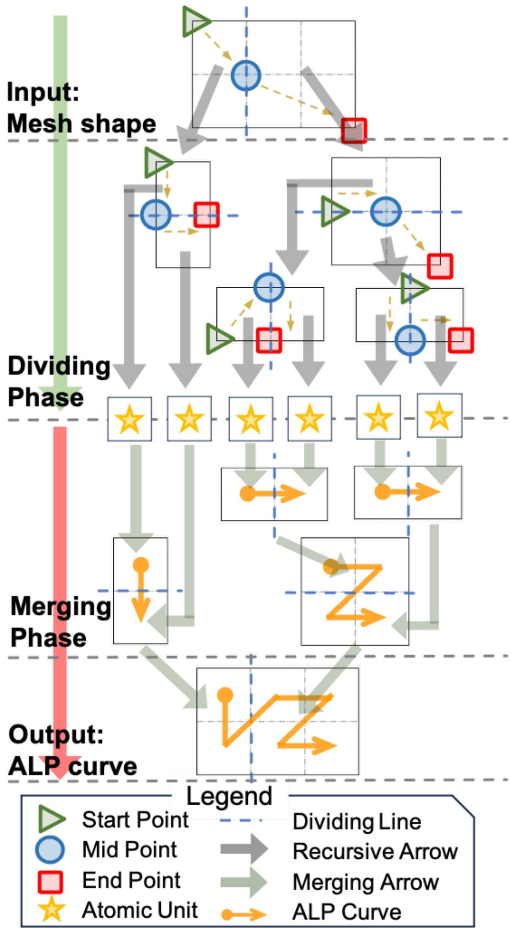


Fig. 5. Detailed construction of the ALP curve on a 2×3 mesh.

B. Main Idea

The ALP curve construction algorithm uses a divide-and-conquer recursive strategy. At each recursion level, the algorithm begins by identifying the central point of the mesh, which serves as the midpoint of the curve. The mesh is then split into two equal sub-shapes. In the sub-shape containing the start point, the midpoint is set as the new end point for recursion. Conversely, in the sub-shape with the end point, the midpoint becomes the new start point. This recursive division continues until the mesh is reduced to a 1×1 unit. Finally, the algorithm merges the curve segments from both sub-shapes.

Fig. 5 illustrates the ALP curve's construction process on a 2×3 mesh. In this figure, each recursion level's start, midpoint, and end points are marked with green triangles, blue circles, and red squares, respectively. The dividing lines for each mesh level

are shown as blue dashed lines. During the recursion, each level is segmented into two sub-problems (indicated by grey arrows), continuing until the mesh is reduced to its smallest unit. In the backtracking phase, these sub-problems are recombined at their original problem level, as illustrated by green arrows.

The remainder of this section will formalize the complete details of the ALP construction algorithm, analyze its complexity, and discuss the underlying logic of some of the special design decisions of the algorithm.

C. Formulation

We first define the smallest unit of a mesh, referred to as a cell, denoted by a lowercase letter c and represented by a pair of integer coordinates (x, y). Without loss of generality, we define the coordinates of the top-left cell as (0,0), with x increasing downwards and y increasing to the right.

$$c_i = (x_i, y_i), \text{ where } x_i, y_i \in \mathbb{N}. \tag{2}$$

A mesh is a shape composed of a series of available cells, meaning a mesh M is a set of distinct cells. A mesh composed of n cells can be expressed as:

$$M = \{c_0, c_1, \ldots, c_{n-1}\}. \tag{3}$$

Furthermore, we define the four vertices of each cell and their coordinates in the mesh as v, as well as the set of all vertices in the mesh M as V_M. Similarly, the coordinates of the top-left vertex of the top-left cell are defined as (0,0), with x increasing downwards and y increasing to the right.

$$v_i = (x_i, y_i), \text{ where } x_i, y_i \in \mathbb{N}. \tag{4}$$

$$V_M = \{v | v \text{ is one of the vertices of } c \in M\}. \tag{5}$$

A space-filling curve is a mapping from a 1D sequence to 2D coordinates, i.e., a one-to-one correspondence between a continuous sequence of natural numbers and the coordinates of cells in a mesh. This mapping is represented by a *bijective* function:

$$F_{SFC} : \{x | x \in \mathbb{N}, x < n\} \rightarrow M, where n = |M|. \tag{6}$$

D. The ALP Curve Construction Algorithm

Algorithm 1 outlines the construction of the ALP curve with the pseudocode for `constructALP()`, a recursive procedure. This function takes four inputs: the target mesh shape M, the starting vertex v_{start}, the ending vertex v_{end}, and the initial curve label base $base \in \mathbb{N}$, set to 0 initially. Additionally, the global variable F_{ALP}, representing the ALP curve's mapping function, is defined externally. The `constructALP()` method constructs F_{ALP} throughout the recursion rather than returning it upon completion.

The pseudocode in lines 2 to 4 illustrates the recursion's base case when the mesh M reduces to a single 1×1 cell. At this point, the single viable cell c corresponds to the label $base$. The mapping is established by setting $F_{ALP}(base)$ to c, aligning the $base$-th node in the 1D sequence with the 2D mesh position c in the ALP curve.

Algorithm 1: $constructALP(M, v_{start}, v_{end}, base)$.

```
1  Global: F_ALP;
   Input: M, v_start, v_end, base
   Output:
   /* Recursive Base Case                          */
2  if size_of(M) == 1 then
3      F_ALP(base) ← the unique cell in M;
4      Return;
   /* Find the v_mid                               */
5  v_center ← The vertex at the center of the mesh;
6  V_M ← Set of all vertices in the mesh M;
7  if v_center ∈ V_M then
8      v_mid ← v_center;
9  else
10     v_mid ← v_i in V_M that is equidistant from v_start
         and v_end and nearest to v_center;
   /* Divide the mesh                              */
11 if its possible to cut M by a dividing line passing
     through v_mid then
12     Cut M by the dividing line;
13     M_start ← {c|c ∈ M, c in the same part as v_start};
14     M_end ← M − M_start;
15 else
16     M_start, M_end ← {};
17     dis_start ← getBfsDis(v_start, V_M);
18     dis_end ← getBfsDis(v_end, V_M);
19     foreach c_i in M do
20         if cellDis(dis_start, c_i) < cellDis(dis_end, c_i)
             then
21             M_start ← M_start + c_i;
22         else
23             M_end ← M_end + c_i;
   /* Recursive processing                         */
24 base_end ← base + size_of(M_start);
25 constructALP(M_start, v_start, v_mid, base);
26 constructALP(M_end, v_mid, v_end, base_end);
```

The pseudocode in lines 5 to 10 focuses on identifying the midpoint v_{mid} of the curve. This step involves finding the geometric center v_{center} of the target mesh shape M. Due to M's irregularity, v_{center} is not simply the average of the mesh dimensions. The algorithm employs specific formulas to calculate $v_{center} = (x_{center}, y_{center})$:

$$x_{center} = \left\lfloor \frac{\sum_{c_i=(x_i,y_i)\in M} x_i}{|M|} \right\rfloor. \tag{7}$$

A similar formula is used to calculate y_{center}, substituting y_i for x_i. This represents the calculation of the "centroid" of all cells in the mesh, rounded to the nearest vertex. Once v_{center}, the center point coordinates, are obtained, it's crucial to verify if it lies within the mesh's shape, as shown in Fig. 6(a), where v_{center} may be outside the mesh. If v_{center} is inside the mesh, it directly becomes v_{mid}. Otherwise, the distances from the start and end

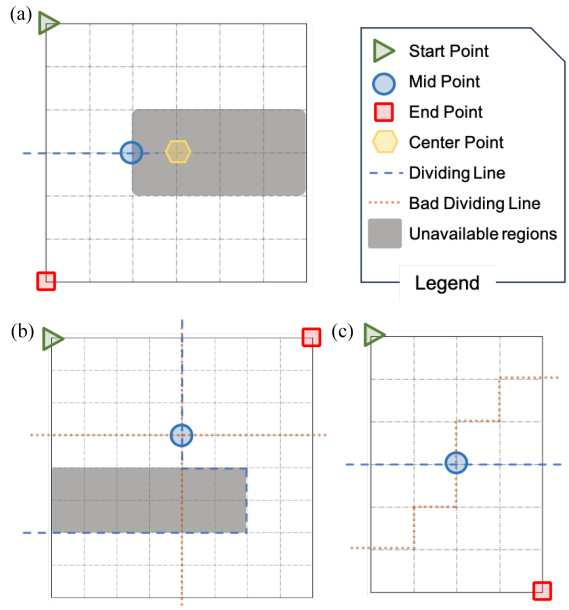


Fig. 6. Key design considerations for the ALP algorithm. **(a)** Handling an off-mesh geometric centroid. **(b)** BFS partitioning for meshes unsplittable by straight-line cuts. **(c)** Prioritizing straight-line division over BFS to maintain regular shapes and better locality.

vertices to all other vertices are computed. This is achieved using a standard Breadth-First Search (BFS), a well-known graph traversal algorithm that systematically finds the shortest path from a source on an unweighted grid. The BFS calculates the minimum number of four-directional movements required to reach every vertex from v_{start} and v_{end}, populating the distance arrays $dis_{start}[\ldots]$ and $dis_{end}[\ldots]$, respectively. Finally, the algorithm identifies and selects the vertex equidistant from the start and end points and closest to v_{center} as the new midpoint.

Lines 11 to 23 of the pseudocode describe the algorithm's strategy to divide the mesh into two subregions using the midpoint v_{mid}. The division is attempted through either horizontal or vertical lines, but it must satisfy two criteria: 1) the line must split the mesh into two distinct parts, and 2) the line must place the start and end points of the curve in different subregions. The preferred division direction is the one with fewer vertices directly on the dividing line, optimizing cross-region point pair distance, thereby improving the curve's locality performance. In regular meshes, the algorithm favors more equal subregion dimensions. However, as seen in Fig. 6(b), when neither horizontal nor vertical cuts are feasible, the algorithm resorts to a BFS-based division. This method, using the distances dis_{start} and dis_{end} generated via the aforementioned BFS process, divides cells based on their proximity to the start or end point. The BFS distance for a cell to a point is the average of the BFS distances of its four vertices to that point, as given by the formula:

$$cellDis(dis_{v_0}, c) = \frac{\sum_{v_i \text{ is a vertex of cell } c} dis_{v_0}[v_i]}{4}. \quad (8)$$

where v_0 is either v_{start} or v_{end}.

Lines 24 to 26 in the pseudocode address the recursive aspect of completing the ALP curve construction. After splitting the current mesh into two sub-meshes, the algorithm assigns the curve labels from $base$ to $base + |M_{start}| - 1$ to the first sub-mesh and the remaining labels to the second. The label base $base\_end$ for the latter is set to $base + |M_{start}|$. `constructALP()` is then recursively applied to both sub-meshes. Once this recursive mapping is complete, further merging is unnecessary as all assignments to the mapping function F_{ALP} are finalized during the recursion's base cases.

E. Complexity Analysis

As modern neuromorphic hardware systems may dynamically face the demands of mapping SNNs comprising billions of neurons, the efficiency of the mapping algorithm is of paramount importance. This section focuses on a detailed analysis of the time and space complexity of the Adaptive Locality-Preserving (ALP) curve construction algorithm, particularly in the context of large-scale mapping problem.

1) Time Complexity Analysis: The time complexity of the ALP curve construction algorithm is determined by the recursive process and the complexity of operations within each recursion:

- *Recursive depth analysis:* The depth of the recursive process is a crucial factor in determining the algorithm's time complexity. Given that the mesh is divided approximately evenly at each step, the depth of recursion would be $O(\log n)$, where n is the number of cells in the mesh.
- *Complexity within each recursive layer:* At each recursive level, the complexity of operations varies. For simple and regular meshes, calculating the midpoint and dividing the mesh is a constant-time operation, $O(1)$. However, in more complex cases that necessitate the BFS mechanism for midpoint calculation, the complexity at that level increases to $O(n)$, where n is the number of cells in the mesh at that recursive level. Despite the potential increase in complexity due to the BFS mechanism, the total complexity for all recursive calls at the same depth collectively remains $O(n)$.
- *Overall time complexity:* Combining the depth of recursion and the per-layer operations, the overall time complexity of the ALP curve construction algorithm is bounded by $O(n \log n)$. This upper bound holds even for the most complex, irregular shapes that necessitate the frequent use of the BFS-based partitioning mechanism. In ideal scenarios where the mesh can be consistently divided using simple geometric splits, the complexity can be as efficient as $O(n)$.

2) Space Complexity Analysis:

- *Recursive calls:* The depth of the recursive call stack contributes logarithmically to the space complexity, $O(\log n)$.
- *BFS process:* The BFS process for more complex midpoint calculations and mesh divisions adds to the space requirement. However, since different recursive levels are independent, this space can be efficiently reused across the recursion, maintaining $O(n)$ space.
- *Overall space complexity:* Therefore, the space complexity, including the BFS process, remains $O(n + \log n)$, which simplifies to $O(n)$.

F. Special Design Considerations in the ALP Curve Construction Algorithm

In this section, we analyze and highlight several key design aspects of the ALP curve construction algorithm, which significantly contribute to its performance and the locality properties of the resulting curve.

- *Utilizing the geometric centroid as the midpoint:* The primary purpose of using the geometric center of the shape as the midpoint is to ensure an even division of the mesh, which is crucial for the efficiency of the algorithm. Second, the midpoint acts as a junction for the curve segments before and after it, meaning that in the mapped SNN network, all cross-domain spike communications pass through this area, forming a communication hotspot. The center of the mesh is often an open and accessible area, making it an ideal location for such a hotspot. In fact, we believe this is the most important reason for the ALP curve's inherent locality properties.
- *Introduction of BFS:* The BFS mechanism addresses the challenges of finding midpoints and dividing irregular shapes. It ensures the uniformity of the division, thereby maintaining the algorithm's efficiency. Additionally, the BFS algorithm has excellent time and space complexity, adding minimal extra overhead to the overall algorithm.
- *Why BFS is not always used:* Despite the advantages of BFS, it is employed only as an alternative when normal midpoint finding and mesh division are not feasible, rather than as the default approach. The reason for this is that simple horizontal or vertical divisions contribute to regularizing the mesh shape. Utilizing BFS divisions might lead to irregular, jagged edges on the mesh, which can be detrimental to the curve's locality properties. Complex and irregular shapes increase the surface area of the division line, enlarging the average distance between points in the subdivided subgraphs and raising the likelihood of the curve navigating into dead ends. Fig. 6(c) illustrates the difference between divisions using BFS and those not using it. Thus, the algorithm prefers straight-line divisions wherever possible, gradually regularizing even highly irregular initial mesh shapes into more uniform rectangles.
- *Guaranteed Convergence for Arbitrary Shapes:* A key strength of our algorithm is its guaranteed convergence for any arbitrarily shaped mesh, including extreme cases with disconnected "islands" of cores. This robustness is achieved through a comprehensive, two-tiered distance-based partitioning strategy. For any given core, the algorithm first attempts to partition it based on its BFS graph distance to the v_{start} and v_{end} points. However, in the case of a core being on a completely isolated island and thus unreachable via BFS from either endpoint, we employ a fallback mechanism: the core is assigned to a sub-problem based on whichever (start or end) point is closer in terms of Manhattan distance. This two-tiered approach ensures that every core, regardless of its location or connectivity, can be decisively assigned during each recursive step, guaranteeing that the algorithm will always converge and successfully construct a valid mapping for any hardware topology.

In summary, the construction method of the ALP curve is specifically designed to ensure the algorithm's adaptability to arbitrary mesh shapes while preserving essential locality properties. These strategic designs significantly enhance the algorithm's efficiency, particularly in addressing the complexities arising from various hardware configurations in mapping very-large-scale SNNs.

V. The Proposed Mapping Approach

The ALP curve, leveraging its inherent locality properties, efficiently maps the data flow of an SNN from the input to the output layers within neuromorphic hardware. This mapping is achieved without requiring the structural details of the network. Consequently, after employing the ALP curve for initial mapping, significant opportunities arise for local adjustments and optimizations, particularly focusing on the communication intensity relationships between cores. Building upon our previous work [15], which introduced the Force Directed (FD) algorithm for iterative refinement in SNN mapping, we incorporate this proven optimization technique into our current approach. The distinctive aspect of our present work lies in replacing the conventional Hilbert curve with our newly proposed ALP curve. This advancement aims to offer enhanced adaptability and efficiency in accommodating the diverse configurations of neuromorphic hardware.

Fig. 7 presents a diagram of our complete mapping approach. The full mapping process primarily consists of three steps: 1) Using topological sorting to arrange the original PCN into a 1D sequence; 2) Constructing the ALP curve on the target mesh, thereby mapping the 1D sequence onto the 2D cores to achieve an initial mapping placement; 3) Iteratively refining this initial mapping using the FD algorithm, continuing until the algorithm converges.

A. Topological Sorting

First, we employ a classic topological sorting algorithm to obtain a 1D topological sequence for the input Partitioned Cluster Network (PCN). The primary goal of this step is to prepare the input required for the ALP space-filling curve: a 1D sequence that preserves the logical order from input to output of the network as much as possible. Formally, for a graph structure $G_{PCN} = \{V_{PCN}, E_{PCN}\}$ represented by the vertex set V_{PCN} and edge set E_{PCN}, its topological sequence is a mapping $F_{Topo} : V_{PCN} \to \mathbb{N}$

$$F_{Topo}(C_i) = j. \tag{9}$$

indicates that the neuron cluster C_i in the vertex set is positioned at the j-th place in the topological order.

B. Initial Mapping Placement Using ALP Curve

Upon obtaining the PCN's topological sequence from the first step, and after constructing the ALP curve F_{ALP} on the target mesh M, we establish a mapping relationship from the

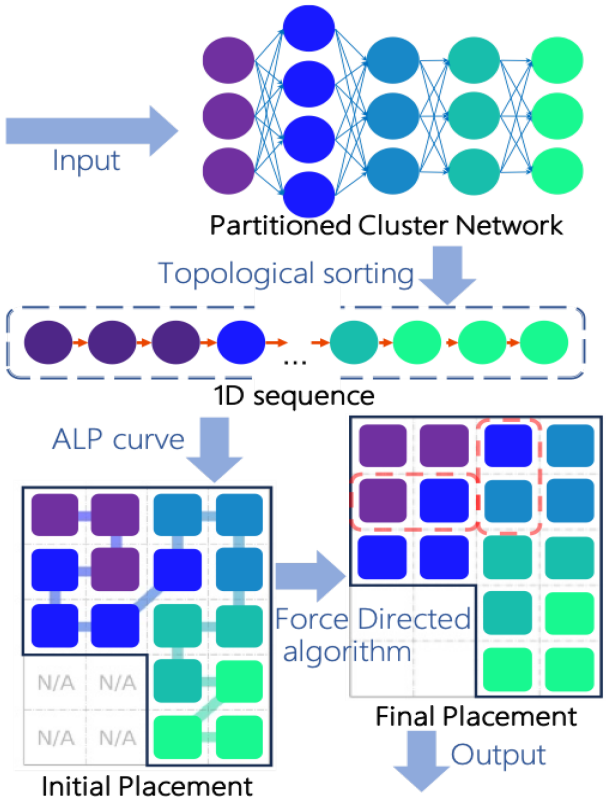


Fig. 7. Diagram of the proposed approach.

network's clusters C_i to the cells c_j (i.e., available neuromorphic computing cores) on the mesh. This forms an initial Placement $P_{init} : V_{PCN} \to M$

$$P_{init}(C_i) = F_{ALP}(F_{Topo}(C_i)) = c_j. \tag{10}$$

C. Force Directed Algorithm

The FD algorithm is an iterative optimization algorithm, where the core idea is to view communication intensities between computing cores as tension relationships. Cores with more intensive communications exert greater tension on each other. Based on this physical model, clusters in the Placement adjust their positions relative to neighboring cores to release tension, thereby minimizing the system's total energy (the optimization cost) and optimizing the Placement.

We chose the FD algorithm for further optimization due to its status as the best existing algorithm for large-scale SNN mapping in terms of solution quality and solving speed. Additionally, its basic iterative operations, based on swapping positions between adjacent cores, are inherently suitable for complex mesh shapes, aligning with our objective of mapping SNNs to meshes of arbitrary shapes. Overall, the FD algorithm can be seen as an optimization process. Given an initial Placement P_{init}, the PCN structure G_{PCN}, and the target mesh shape M, the algorithm iteratively refines the existing Placement until convergence, yielding the final Placement:

$$P_{final} = FD\_algorithm(P_{init}, G_{PCN}, M). \tag{11}$$

Algorithm 2: Proposed Approach.

Input: $M, G_{PCN} = (V_P, E_P, w_P)$
Output: P_{final}
/* Get initial placement */
1 $Topo \leftarrow topological\_sorting(G_{PCN})$;
2 $F_{ALP} \leftarrow constructALP(M, ...)$;
3 $P_{Init} \leftarrow$ apply $Topo$ sequence to F_{ALP} ;
/* Optimize using the FD algorithm */
4 $L \leftarrow$ empty $List$;
5 **foreach** $pair = (p_u, p_v)$ *in 2D mesh that* p_u *and* p_v *is adjacent* **do**
6 $Tension[pair] \leftarrow$ tension between clusters at p_u and p_v ;
7 **if** $Tension[pair] > 0$ **then**
8 $L \leftarrow L + pair$
9 **while** L *is not empty* **do**
10 **foreach** $pair_i = (p_u, p_v)$ *in* L *and* $i < \lambda|L|$ **do**
11 **if** $Tension[pair_i] > 0$ **then**
12 $P_{Init} \leftarrow$ update P_{Init} by swapping $pair_i$;
13 $Tension \leftarrow$ maintain $Tension$;
14 $L_{next} \leftarrow$ empty $List$;
15 **foreach** $pair_i$ **do**
16 **if** $Tension[pair_i] > 0$ **then**
17 $L_{next} \leftarrow L + pair$;
18 $L \leftarrow L_{next}$;
19 $P_{final} \leftarrow P_{Init}$;

Algorithm 2 presents the complete workflow of our mapping approach. The algorithm takes two inputs: the shape of the mesh M and a PCN G_{PCN}. Lines 1 to 3 describe how to obtain an initial placement by performing topological sorting on the PCN and constructing the ALP curve using Algorithm 1 based on the mesh shape. Lines 4 to 19 outline the general process of the FD algorithm's iterative optimization of the mapping scheme.

Lines 4 to 8 construct queue L, which contains all adjacent pairs of points with tension greater than 0. Swapping the clusters mapped to these pairs can reduce the system's total energy, thereby optimizing the current solution. Lines 9-18 describe the main iterative process of the FD algorithm: examining selected pairs from queue L, performing swap operations on clusters of qualifying pairs, and simultaneously updating and maintaining the force conditions of other clusters.

During lines 9 to 18, the FD algorithm iteratively processes the queue L: examining pairs from L, swapping clusters if their tension is greater than 0, and updating tension values for affected pairs. After each iteration, pairs whose tension remains above 0 are collected into a new queue L_{next} for the next round. This cycle continues until L becomes empty, indicating no more high-tension pairs exist for swapping, thus confirming the system has reached its minimum energy state and the algorithm has converged.

It should be noted that the above process is a general outline of the FD algorithm. A strict time complexity analysis for this

iterative refinement stage is impractical, as the runtime depends heavily on the number of iterations required for convergence. However, a key advantage of our approach is that the high-quality initial placement provided by the ALP curve significantly accelerates this convergence process. Due to space limitations and because it is not the main contribution of this paper, many other details and optimization techniques are not reflected in this pseudocode. For a detailed implementation of the FD algorithm, refer to [15].

VI. Experiments and Evaluations

A. Overview

Our experimental setup simulates a neuromorphic computing hardware model using software tools. These experiments aim to evaluate our proposed SNN mapping approach against existing methods and compare various space-filling curves in terms of performance indicators and algorithmic efficiency.

We structure our experiments into two parts. First, we conduct a comprehensive evaluation of our complete SNN mapping approach by benchmarking it against existing methods. This involves mapping multiple real-world machine learning SNN applications, providing a thorough assessment of our method's performance in practical scenarios. Second, to further validate our approach, we carry out a comparative analysis of the ALP curve against several classic space-filling curves, offering additional insights into the ALP curve's effectiveness.

The experiments are conducted on an Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-59-generic x86_64) workstation with 40 CPU cores (Intel(R) Xeon(R) Silver 4210R CPU @ 2.40 GHz), 256 GB of memory, and 4 GPU cards (GeForce RTX 3080). We utilize GPUs for training and transforming SNN applications. All mapping algorithms are implemented in C++ without the use of GPU computing power or multicore parallel computing features.

B. Comparative Analysis of SNN Mapping Approaches

1) Experimental Setup: In the first part of our experiments, we compare our complete SNN mapping approach with other existing methods. We evaluate the following four approaches:

1) *The Baseline:* Random mapping, where clusters are randomly mapped into the neuromorphic hardware.
2) *DFSynthesizer:* A greedy mapping algorithm proposed in [12].
3) *PSO:* Particle Swarm Optimization, a classic optimization algorithm used in various mapping approaches [11], [12], [21], with configurations taken from [21].
4) *Hilbert_FD:* The key reference SFC-based method [15], which employs a mapping strategy based on the Hilbert curve and the FD algorithm. Due to the Hilbert curve's inherent limitation in handling irregular shapes, we were only able to include Hilbert_FD algorithm in the comparisons for regular mesh scenarios. The algorithm's inability to generate valid traversal sequences for irregular shapes made it unsuitable for irregular mesh evaluations.

TABLE II
Benchmarks

Applications	G_{SNN} Neurons	G_{SNN} Synapses	G_{PCN} Clusters	G_{PCN} Connections
LeNet	1.0M	188M	251	2151
AlexNet	0.9M	1.0B	229	4289
MobileNet	6.9M	0.5B	1764	37418
InceptionV3	14.6M	5.4B	3570	117597
ResNet-50	28.5M	11.6B	6956	478602
SpikingBERT	24.6M	50.0B	6005	2.50M
STBP-tdBN	28.0M	25.56M	6930	109471
STDP	10000	2.0M	16	256

5) *Proposed Approach:* Our approach, using the ALP curve for initial solution generation followed by FD algorithm optimization.

Our benchmark comprises a diverse suite of neural networks to ensure a comprehensive evaluation. This suite includes models created via ANN-to-SNN conversion using the SNNToolBox [34], such as *LeNet* [35], *AlexNet* [36], *MobileNet* [37], *InceptionV3* [38] and *ResNet-50* [39], originally trained on the ImageNet [40] dataset with TensorFlow [41]. To address greater model diversity, we also incorporate the *SpikingBERT* language model [42], which is trained through knowledge distillation from a pre-trained BERT. Furthermore, the benchmark contains a deep vision SNN directly trained on the ImageNet [40] dataset with the *STBP-tdBN* method [43]. Finally, to evaluate biological plausibility, we include a neuroscience model from Vogels et al. [44] based on the *STDP* (Spike-Timing-Dependent Plasticity) mechanism. Table II details the parameters of the application suite.

We refer to Reference [15] and use two estimation metrics to quantitatively assess placement quality:

- *Energy Consumption:* The total energy consumed by all spikes on interconnect given $G_{PCN} = (V_P, E_P, w_P)$ and Placement P is computed as follow

$$\mathcal{M}_{ec} = \sum_{e_{i,j} \in E_p} \left(w_P(e_{i,j}) \left(\|P(c_i) - P(c_j)\| + 1\right) E_r + w_P(e_{i,j}) \|P(c_i) - P(c_j)\| E_w\right), \quad (12)$$

  where $\|\cdot\|$ is the $L1$ norm, which gives the Manhattan distance between two cores, E_r is the energy consumption for a router route one spike message, and E_w is the energy consumption for one spike message transmitted through a wire between routers. w_P is the weight function, presents the communication traffic volume between clusters, and it is proportional to the total number of spikes pass through this connection.
- *Latency:* The maximum time spike messages spent on transmission in interconnect network among all connection routes, given $G_{PCN} = (V_P, E_P, w_P)$ and Placement P, is computed as follow

$$\mathcal{M}_l = \max_{e_{i,j} \in E_p} \left(\left(\|P(c_i) - P(c_j)\| + 1\right) L_r + \|P(c_i) - P(c_j)\| L_w\right). \quad (13)$$

TABLE III
CONSTANTS CONFIGURATION

E_r	E_w	L_r	L_w
1	0.1	1	0.01

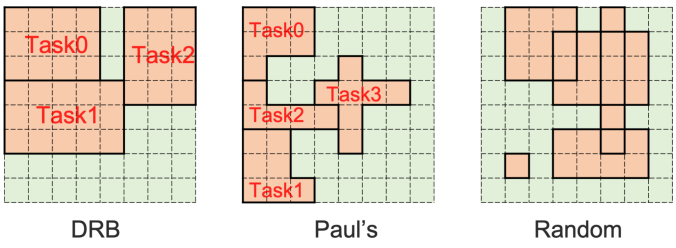


Fig. 8. Examples of generated irregular meshes.

where L_r is the delay for a router route one spike message, and L_w is the delay for one spike message transmitted through a wire between routers.

Consistent with Reference [15], we configure the constants in (12) and (13) as specified in Table III.

To evaluate the performance of the algorithm in irregular mesh shapes, we focus on the most common and crucial practical scenario: mesh configurations that result from parallel task execution (as shown in sub-fig g) of Fig. 5). To simulate this scenario, we generate dynamic task arrivals and use state-of-the-art task allocation algorithms to assign computing resources on the mesh. As tasks arrive and depart randomly, they create irregular patterns of available resources, which we capture as benchmarks.

We employed three task allocation algorithms:

1) *DRB* (Dynamic Resource Balance) [45]: An algorithm that maximizes system throughput by balancing on-chip computing and communication resources, primarily through a multi-rectangle selection (MRS) algorithm for application area allocation.
2) *Paul's algorithm* [46]: A hybrid allocation and scheduling strategy that uses design-time results during runtime to reduce communication overhead and optimize deadline satisfaction.
3) *Random*: A naive method that creates irregular shapes by randomly marking regions as occupied.

Fig. 8 illustrates examples of irregular mesh shapes generated by these three different algorithms. Since our focus is on SNN mapping, we use these algorithms solely to simulate irregular mesh shapes produced during run-time task transitions, excluding temporal information such as task scheduling details.

We conducted multiple simulation rounds for each target SNN using different algorithms to generate various samples of irregular shape. We then filtered these samples to ensure that each shape's largest connected available region could accommodate the target SNN. Table IV shows the number of irregular shape benchmarks produced by each algorithm.

2) Execution Time Comparison: Fig. 9 illustrates a comparison of the normalized average execution time required by each method, presented on a logarithmic y-axis. All execution

TABLE IV
IRREGULAR SHAPE BENCHMARK COUNTS

Applications	Allocation Algorithms DRB [45]	PAUL's [46]	Random	Total
STDP	99	97	100	296
LeNet	93	97	100	290
AlexNet	90	85	100	285
MobileNet	80	92	100	272
InceptionV3	96	97	100	293
ResNet	99	94	100	293
SpikingBERT	90	89	100	279
STBP-tdBN	85	91	100	276

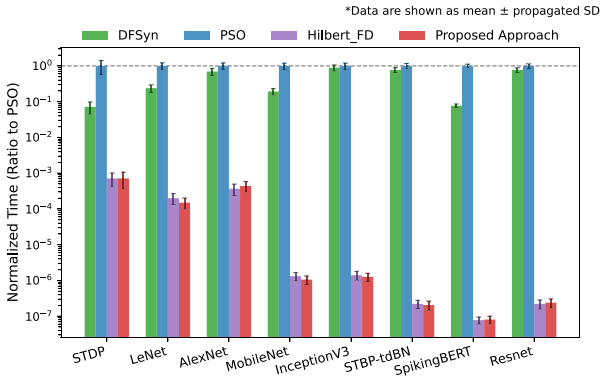


Fig. 9. Results on execution time.

times are normalized to that of the PSO method. The data points represent the mean values from multiple experiments, with the error bars indicating the propagated standard deviation. The results clearly demonstrate that SFC-based approaches hold a significant advantage in computational efficiency, particularly in large-scale mapping tasks. In contrast to the compared methods, whose time requirements increase geometrically with their algorithmic complexity, our proposed method maintains an extremely low normalized time, even for complex tasks like mapping the ResNet network. Its computational cost is several orders of magnitude lower than the baseline PSO method and the DFSynthesizer method.

3) Energy Consumption Evaluation: Fig. 10 illustrates the Energy Consumption results, measured using the metric defined in (12). The bar chart presents the energy consumption of different methods relative to the baseline across various benchmarks generated by DRB, Paul's, and Random algorithms. The results clearly demonstrate that our Proposed Approach consistently outperforms all other methods across all benchmark scenarios. Moreover, the performance advantage of our approach becomes increasingly pronounced as the SNN scale increases, highlighting its superior scalability for large-scale applications.

Quantitatively, when averaging across all experimental samples, our algorithm achieves remarkable energy efficiency, requiring only 24.1%, 42.9%, and 33.5% of the energy consumed by Baseline, DFSynthesizer, and PSO methods, respectively. This represents a 57.1% reduction in energy

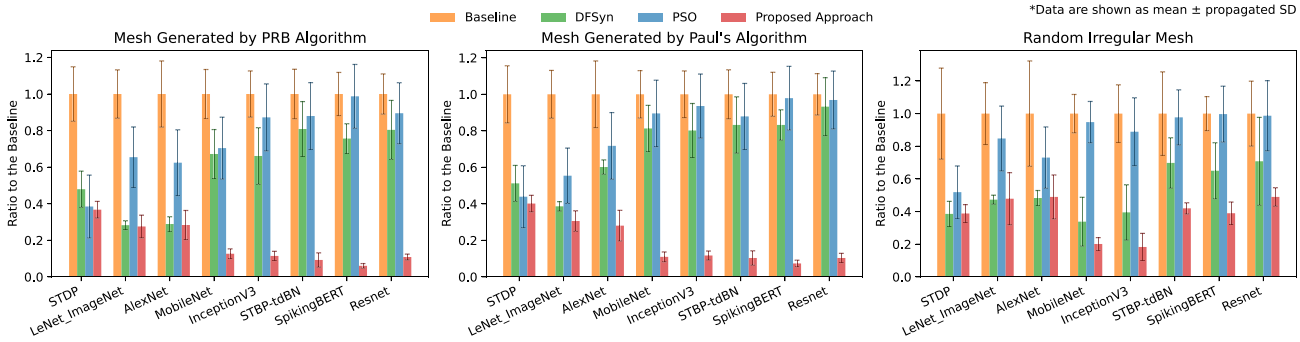


Fig. 10. Results on energy consumption.

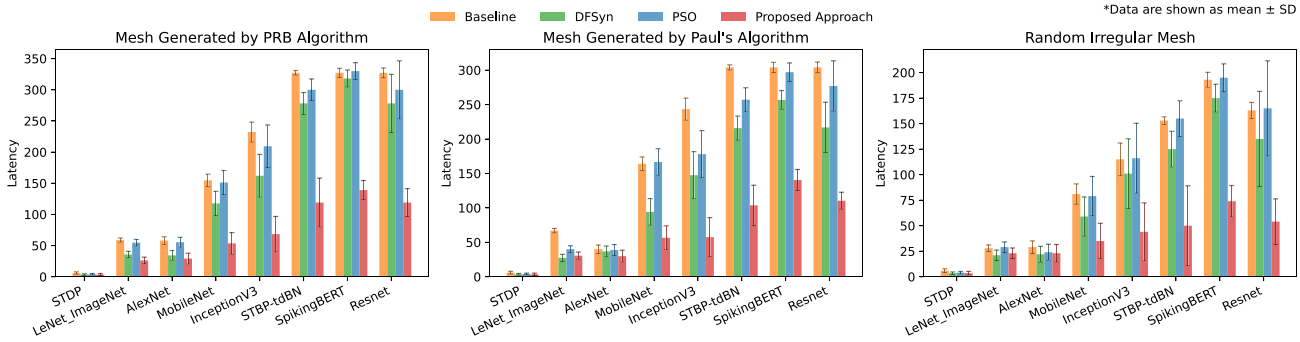


Fig. 11. Results on latency.

consumption compared to the best competing method (DFSynthesizer). The advantage is even more substantial in large-scale scenarios, particularly with ResNet, where our algorithm reduces energy consumption by an average of 76.9%, 66.4%, and 75.4% compared to Baseline, DFSynthesizer, and PSO methods, respectively.

4) Latency Evaluation: Fig. 11 presents the Latency comparison results, a metric defined by (13), across different benchmark scenarios generated by DRB, Paul's, and Random algorithms. The results demonstrate that our method significantly outperforms all competing approaches across all scenarios, with the performance advantage widening as the SNN scale increases.

Quantitatively, our algorithm achieves remarkably lower maximum Latency, averaging only 45.5%, 62.9%, and 50.4% of that observed in Baseline, DFSynthesizer, and PSO methods, respectively. This considerable improvement is especially prominent in large-scale models. For instance, in the ResNet benchmark, our approach demonstrates a pronounced reduction in latency, requiring only 38.4%, 51.8%, and 43.7% of the latency of the Baseline, DFSynthesizer, and PSO methods, respectively, underscoring its effectiveness in complex, large-scale SNN applications.

5) Impact of Mesh Irregularity: We quantify the irregularity of benchmark mesh shapes using an $Irregularity$ parameter, as defined in (14). For meshes generated by DRB or Paul's algorithm, $Irregularity$ represents the number of tasks occupying regions in the shape, while for meshes generated by the Random algorithm, it represents the number of randomly removed rectangles.

$$Irregularity = \begin{cases} N_{existing}, & \text{for DRB/Paul's alg.} \\ N_{removed}, & \text{for Random alg.} \end{cases} \quad (14)$$

where $N_{existing}$ is the number of existing tasks and $N_{removed}$ is the number of removed rectangles.

Fig. 12 presents the average performance of different algorithms in varying levels of irregularity when mapping the ResNet SNN. The results demonstrate that compared to other methods, our approach experiences significantly slower performance degradation as the irregularity of the mesh increases, both in terms of energy consumption and latency metrics. This highlights the robustness and adaptability of our algorithm in handling irregular topologies. While competing methods show steep performance declines with increasing irregularity, our approach maintains relatively consistent performance, demonstrating its effectiveness in real-world scenarios where perfect regularity is rarely achievable.

C. Performance and Efficiency Comparison of Space-Filling Curves

To complement our main experiments, we conducted an additional set of tests to evaluate the performance and efficiency of the ALP curve against other established space-filling curves (SFCs). The compared SFCs included:

- *Random mapping:* as a baseline.
- *Circle:* sequential outward-in mapping.
- *Zigzag:* serpentine row-by-row mapping.

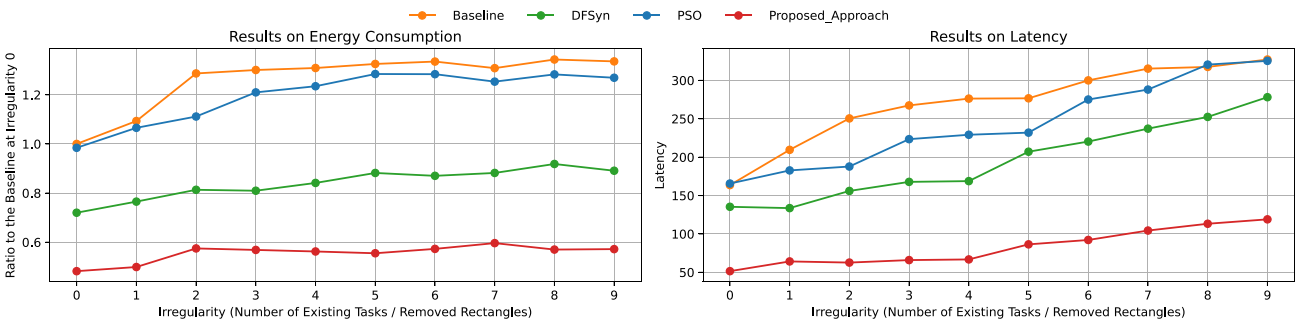


Fig. 12. Results on different mesh irregularity for ResNet.

- *Z-order:* a classic fractal space-filling curve.
- *Hilbert:* a classic fractal space-filling curve known for its good locality properties.
- *Context-based:* an adaptive space-filling curve introduced in [31], designed for non-regular regions.
- *General SFC:* a method to create a space-filling curve using a KD-TREE [32].
- *Dense Curve:* a curve that densely covers any geometric domain [33].

Since most space-filling curves, except for our proposed ALP curve, cannot be constructed on irregular shapes, this part of the experiment compares the performance of different curves on regular meshes of varying sizes.

To quantify the locality property of a space-filling curve represented by the mapping function F_{SFC} (refer to (6)) for a target mesh M, we define the locality score $\mathcal{L}$ with the following formula:

$$\mathcal{L}(F_{SFC}, M) = \sum_{i,j\in\mathbb{N}, i<j<|M|} \frac{||F_{SFC}(j) - F_{SFC}(i)||}{(j-i)} |M|^{1.5} \quad (15)$$

where $||\cdot||$ denotes the Manhattan distance. The formula calculates the sum of the distances between all pairs of points on a 2D plane, weighted by the reciprocal of their separation in 1D sequence. Essentially, pairs of points close in the 1D sequence contribute more significantly to the metric after mapping. Consequently, a smaller sum of these weighted distances suggests that the curve more effectively maps points that are close in 1D sequence to nearby locations in 2D space, indicating superior locality. The division of this weighted sum by the 1.5th power of the mesh size mitigates scaling effects, yielding a quantified locality metric for comparative analysis.

Fig. 13 presents the metrics as defined by (15). The results show that the ALP and Hilbert curves display nearly identical quantified locality metrics, outperforming other curves significantly. This advantage is more pronounced at larger scales. At the largest scale, the metrics for the Z-order, ZigZag, and Circle curves are 1.13, 1.81, and 2.69 times those of the ALP curve, respectively. Context-based curve, whose generation algorithm does not focus on locality properties, matches only the metric of the ZigZag curve.

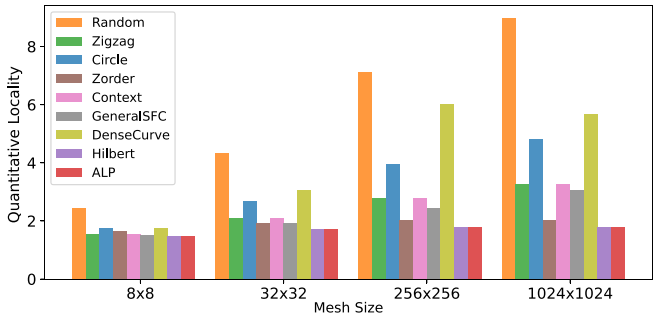


Fig. 13. Quantitative locality for various SFCs across different mesh sizes.

TABLE V
TOTAL SPIKE TRAVEL DISTANCES FOR VARIOUS SFCS ACROSS DIFFERENT MESH SIZES

Mesh_Size	8×8	32×32	256×256	1024×1024
Random	$2,297.2$	$3.43e5$	$1.79e8$	$1.15e10$
Circle	$2,699.3$	$2.42e5$	$1.68e7$	$2.68e8$
Zigzag	$1,589.6$	$1.82e5$	$1.62e7$	$2.66e8$
Zorder	$1,985.7$	$1.25e5$	$8.92e6$	$1.44e8$
Context	$1,592.1$	$1.83e5$	$1.62e7$	$2.66e8$
GeneralSFC	$1,579.9$	$1.80e5$	$1.61e7$	$2.65e8$
DenseCurve	$2,134.5$	$2.59e5$	$1.70e7$	$2.77e8$
Hilbert	$1,563.4$	$84,905.6$	$5.51e6$	$8.81e7$
ALP	$1,552.4$	$84,925.6$	$5.51e6$	$8.82e7$

Table V presents the quality of SNN mapping placements generated by various SFCs, quantified using the Total Spike Travel Distance (TSTD) metric. The $\mathcal{M}_{TSTD}$ for a mapping placement P and network $G_{PCN} = \{V_{PCN}, E_{PCN}\}$ is calculated as follows:

$$\mathcal{M}_{TSTD}(P) = \sum_{e_{i,j}\in E_{PCN}} ||P(c_i) - P(c_j)|| \quad (16)$$

where $e_{i,j}$ represents spike communication between neuron clusters i and j, and $||\cdot||$ is the Manhattan distance. The results show that the ALP curve achieves performance equal to the Hilbert curve across different mesh sizes, highlighting its locality properties.

Table VI presents the construction times for each curve on a large-scale mesh of size 1024×1024. The ALP curve's construction time, while higher than other directly defined curves

TABLE VI
CONSTRUCTION TIMES (MS) FOR SPACE-FILLING CURVES ON A 1024×1024 MESH

Random	Zigzag	Circle	Zorder	Hilbert
42	11	13	13	49
Context	General SFC	Dense Curve	ALP	
132	471	759	233	

due to additional recursive computations, is sufficiently fast at only 233 milliseconds for a million-scale curve, demonstrating its efficiency.

VII. DISCUSSION

While the initial motivation for this paper was to address the challenge of mapping large-scale SNNs onto complex neuromorphic hardware environments, our method can be generalized to a broader range of communication-intensive multicore computational task mapping problems, particularly on target hardware platforms organized in a 2D mesh structure.

Space-filling curves have been recognized for their versatile applications in various research fields, extending well beyond network mapping problems. Works [47] and [48] apply SFC for robotic exploration tasks. Works [49][50] and [51] use SFC for image compression. Work [52] employs SFC for image smoothing. Work [53] apply SFC in the field of computer graphics to assist in rendering. Works [54][55] utilize SFC for 2D and 3D data visualization. Work [56] uses SFC to solve FPGA placement problems.

Our proposed ALP curve, being one of the most flexible and locality-preserving space-filling curves to date, has the potential to facilitate breakthroughs in some of these diverse application areas. Expanding on these applications, however, falls beyond the scope of this paper and will be a subject for our future work.

VIII. CONCLUSION

This paper introduces the Adaptive Locality-Preserving (ALP) curve, a novel space-filling curve designed to map Spiking Neural Networks efficiently onto neuromorphic hardware. The ALP curve overcomes limitations of traditional curves by balancing flexibility, locality preservation, and manageable complexity—making it ideal for large-scale applications with complex mesh configurations. Our experimental results demonstrate that the ALP curve matches the Hilbert curve's locality performance in regular scenarios while significantly outperforming existing algorithms in irregular scenarios. Notably, our method reduces communication overhead by 57.1% compared to current best algorithms when handling irregular mesh shapes. This substantial improvement highlights the ALP curve's exceptional adaptability and efficiency, particularly in complex, real-world neuromorphic hardware environments.

Importantly, our method efficiently handles irregular mesh shapes and defective cores. This is essential for running multiple applications and enhancing resource utilization in ultra-large-scale neuromorphic systems, which often serve as server-like infrastructures with potentially millions of computing cores. These capabilities highlight the critical role of our method in delivering dynamic, high-performance mapping solutions that meet the evolving demands of neuromorphic computing.

REFERENCES

[1] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[2] M. V. DeBole et al., "Truenorth: Accelerating from zero to 64 million neurons in 10 years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019.

[3] B. V. Benjamin et al., "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[4] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

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

[6] L. Shi et al., "Development of a neuromorphic computing system," in *Proc. IEEE Int. Electron Devices Meeting*, 2015, pp. 4–3.

[7] D. Ma et al., "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, 2017.

[8] X. Liu, W. Wen, X. Qian, H. Li, and Y. Chen, "Neu-NoC: A high-efficient interconnection network for accelerated neuromorphic systems," in *Proc. 23rd Asia South Pacific Des. Automat. Conf.*, 2018, pp. 141–146.

[9] F. Galluppi, S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber, "A hierachical configuration system for a massively parallel neural hardware platform," in *Proc. 9th Conf. Comput. Front.*, 2012, pp. 183–192.

[10] A. Balaji et al., "PyCARL: A PyNN interface for hardware-software co-simulation of spiking neural network," 2020, *arXiv:2003.09696*.

[11] A. Balaji et al., "Mapping spiking neural networks to neuromorphic hardware," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 28, no. 1, pp. 76–86, Jan. 2020.

[12] S. Song, H. Chong, A. Balaji, A. Das, J. Shackleford, and N. Kandasamy, "DFSynthesizer: Dataflow-based synthesis of spiking neural networks to neuromorphic hardware," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 3, pp. 1–35, 2022.

[13] T. Titirsha et al., "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 288–301, Feb. 2022.

[14] S. Deng et al., "Darwin-s: A reference software architecture for brain-inspired computers," *Computer*, vol. 55, no. 5, pp. 51–63, 2022.

[15] O. Jin, Q. Xing, Y. Li, S. Deng, S. He, and G. Pan, "Mapping very large scale spiking neuron network to neuromorphic hardware," in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2023, pp. 419–432.

[16] O. Yousuf et al., "Layer ensemble averaging for fault tolerance in memristive neural networks," *Nature Commun.*, vol. 16, no. 1, 2025, Art. no. 1250.

[17] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[18] Y. Hu, Q. Zheng, X. Jiang, and G. Pan, "Fast-SNN: Fast spiking neural network by converting quantized ANN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 12, pp. 14546–14562, Dec. 2023.

[19] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

[20] A. Das, Y. Wu, K. Huynh, F. Dell'Anna, F. Catthoor, and S. Schaafsma, "Mapping of local and global synapses on spiking neuromorphic hardware," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, 2018, pp. 1217–1222.

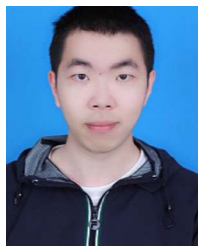
[21] S. Song, M. L. Varshika, A. Das, and N. Kandasamy, "A design flow for mapping spiking neural networks to many-core neuromorphic hardware," in *Proc. IEEE/ACM Int. Conf. On Comput. Aided Des.*, 2021, pp. 1–9.

[22] S. Tosun, O. Ozturk, and M. Ozen, "An ILP formulation for application mapping onto network-on-chips," in *Proc. Int. Conf. Appl. Inf. Commun. Technol.*, 2009, pp. 1–5.

[23] C.-L. Chou and R. Marculescu, "Contention-aware application mapping for network-on-chip communication architectures," in *Proc. IEEE Int. Conf. Comput. Des.*, 2008, pp. 164–169.

[24] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for network-on-chip design," *J. Syst. Archit.*, vol. 59, no. 1, pp. 60–76, 2013.

[25] A. Amir et al., "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *Proc. Int. Joint Conf. Neural Netw.*, 2013, pp. 1–10.
[26] 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, doi: 10.1145/3296979.3192371.
[27] L. Xia et al., "Stuck-at fault tolerance in RRAM computing systems," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 102–115, Mar. 2018.
[28] D. Hilbert, *Dritter Band: Analysis Grundlagen der Mathematik Physik Verschiedenes, Nebst Einer Lebensgeschichte*. Berlin, Germany: Springer, 1935, pp. 1–2.
[29] Y. Rong, X. Zhang, and J. Lin, "Modified Hilbert curve for rectangles and cuboids and its application in entropy coding for image and video compression," *Entropy*, vol. 23, no. 7, 2021, Art. no. 836.
[30] S. H. Nair, A. Sinha, and L. Vachhani, "Hilbert's space-filling curve for regions with holes," in *Proc. IEEE 56th Annu. Conf. Decis. Control*, 2017, pp. 313–319.
[31] R. Dafner, D. Cohen-Or, and Y. Matias, "Context-based space filling curves," *Comput. Graph. Forum*, vol. 19, no. 3, pp. 209–218, 2000.
[32] A. Sasidharan, J. M. Dennis, and M. Snir, "A general space-filling curve algorithm for partitioning 2D meshes," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun., IEEE 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, 2015, pp. 875–879.
[33] X. Ban, M. Goswami, W. Zeng, X. Gu, and J. Gao, "Topology dependent space filling curves for sensor networks and applications," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 2166–2174.
[34] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Front. Neurosci.*, vol. 11, 2017, Art. no. 682.
[35] Y. LeCun et al., "LeNet-5, convolutional neural networks," 2015. [Online]. Available: http://yann.lecun.com/exdb/lenet
[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[37] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015, *arXiv:1512.00567*. [Online]. Available: http://arxiv.org/abs/1512.00567
[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[41] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.
[42] M. Bal and A. Sengupta, "SpikingBERT: Distilling BERT to train spiking language models using implicit differentiation," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 10998–1100 6.
[43] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going deeper with directly-trained larger spiking neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11062–110 70.
[44] T. P. Vogels, H. Sprekeler, F. Zenke, C. Clopath, and W. Gerstner, "Inhibitory plasticity balances excitation and inhibition in sensory pathways and memory networks," *Science*, vol. 334, no. 6062, pp. 1569–1573, 2011.
[45] C. Wang, Y. Zhu, J. Jiang, M. Qiu, and X. Wang, "Dynamic application allocation with resource balancing on noc based many-core embedded systems," *J. Syst. Archit.*, vol. 79, pp. 59–72, 2017.
[46] S. Paul, N. Chatterjee, P. Ghosal, and J.-P. Diguet, "Adaptive task allocation and scheduling on NoC-based multicore platforms with multitasking processors," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 1, pp. 1–26, 2020.
[47] A. Tiwari, H. Chandra, J. Yadegar, and J. Wang, "Constructing optimal cyclic tours for planar exploration and obstacle avoidance: A graph theory approach," in *Proc. Adv. Cooperative Control Optim.: Proc. 7th Int. Conf. Cooperative Control Optim.*, Springer, 2007, pp. 145–165.
[48] A. Wakode and A. Sinha, "Online obstacle evasion with space-filling curves," 2023, *arXiv:2308.02200*.
[49] W. Chen, X. Yao, X. Zhang, and B. Yu, "Efficient deep space filling curve," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 17525–17534.
[50] H. Wang, K. Gupta, L. Davis, and A. Shrivastava, "Neural space-filling curves," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2022, pp. 418–434.
[51] T. Ouni, A. Lassoued, and M. Abid, "Gradient-based space filling curves: Application to lossless image compression," in *Proc. IEEE Int. Conf. Comput. Appl. Ind. Electron.*, 2011, pp. 437–442.
[52] Y. Zang, H. Huang, and L. Zhang, "Efficient structure-aware image smoothingby local extrema on space-filling curve," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 9, pp. 1253–1265, Sep. 2014.
[53] O. Mischenko, "Optimizing cache behavior of ray-driven volume rendering using space-filling curves," Ph.D. dissertation, Dept. Comput. Sci., Stony Brook Univ., Stony Brook, NY, USA, 2006.
[54] H. T. Vo, C. T. Silva, L. F. Scheidegger, and V. Pascucci, "Simple and efficient mesh layout with space-filling curves," *J. Graph. Tools*, vol. 16, no. 1, pp. 25–39, 2012.
[55] C. Muelder and K.-L. Ma, "Rapid graph layout using space filling curves," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1301–1308, Nov./Dec. 2008.
[56] P. Banerjee, S. Sur-Kolay, A. Bishnu, S. Das, S. C. Nandy, and S. Bhattacharjee, "FPGA placement using space-filling curves: Theory meets practice," *ACM Trans. Embedded Comput. Syst.*, vol. 9, no. 2, pp. 1–23, 2009.



Ouwen Jin is currently working toward the doctoral degree with the College of Computer Science and Technology, Zhejiang University. His research interests include brain-inspired computing and neuromorphic computing hardware architectures. His work focuses on optimizing the compilation, deployment, and execution efficiency of spiking neural networks on neuromorphic hardware. He has published as the first author at ASPLOS.

Zhuo Chen is currently working toward the PhD degree with the School of Computer Science, Zhejiang University. His research include neuromorphic computing systems, with particular emphasis on the design, implementation, and application of neuromorphic chips. His work aims to bridge the gap between biological neural processing and artificial intelligence hardware through brain-inspired computing architectures.

Qinghui Xing is currently working toward the PhD degree with the School of Computer Science and Technology, Zhejiang University. His research interests include neuromorphic computing and computer architecture.

Ming Zhang received the PhD degree in computer science from Zhejiang University, in 2019. He is a software engineer with College of Computer Science and Technology, Zhejiang University, China. His research interests include building basic software tools for neuromorphic computers.

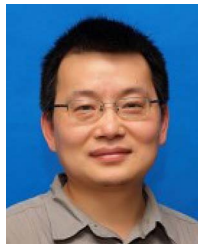
De Ma received the PhD degree in electronic science and technology from the Institute of VLSI, Zhejiang University, Hangzhou, China, in 2013. He is currently an associate professor with the College of Computer Science and Technology, Zhejiang University. In 2013, he joined the Information Engineering School, Hangzhou Dianzi University, Hangzhou, as a faculty member. In February 2018, he joined Zhejiang University. He was an intern with the VERIMAG Laboratory, Grenoble, France, in 2010, and a visiting scholar with IMEC, Leuven, Belgium, in 2013. His research interests include neuromorphic hardware, VLSI design, and SoC architecture.

Ying Li received the BS, MS, and PhD degrees in computer science from Zhejiang University, China, in 1994, 1997, and 2000, respectively. He is currently an associate professor with the College of Computer Science, Zhejiang University. He is leading several research projects supported by the National Natural Science Foundation of China. His research interests include service computing, process mining, and compiler technology.

Xin Du received the PhD degree in computer science from Fudan University, in 2024. He is an assistant professor with the School of Software Technology, Zhejiang University, China. His research interests include service computing, distributed system and brain-inspired computing. He has published several papers in flagship conferences and journals including IEEE ICWS, the *IEEE Transactions on Parallel and Distributed Systems*, etc. He has received the Best Student Paper Award of IEEE ICWS 2020 and IEEE ICWS 2023.

Shuibing He is a professor with the College of Computer Science and Technology, Zhejiang University (ZJU), China, where he leads the Intelligent Storage and Computing Systems (ISCS) Laboratory. He also serves as the vice president of Zhejiang Lab and the deputy director of the Zhejiang Key Laboratory of Big Data Intelligent Computing. His research interests include storage systems, intelligent computing, computer architecture, and high-performance computing. He serves as an associate editor for the *IEEE Transactions on Computers (TC)* and previously held the same role for the *IEEE Transactions on Parallel and Distributed Systems (TPDS)* from 2018 to 2022. Additionally, he has served as the program chair of NAS 2024, general chair of ChinaSys 2024, and a program committee member for conferences such as ICDCS, SRDS, ICPP, IPDPS, and CLUSTER.

Shuiguang Deng (Senior Member, IEEE) received the BS and PhD degrees in computer science from Zhejiang University, China, in 2002 and 2007, respectively. He is currently a full professor with the College of Computer Science and Technology, Zhejiang University. He previously worked as a visiting scholar with the Massachusetts Institute of Technology and Stanford University, in 2014 and 2015, respectively. His research interests include edge computing, service computing, cloud computing, and business process management. He serves as an associate editor for the journal *IEEE Transactions on Services Computing*, *Knowledge and Information Systems*, *Computing*, and *IET Cyber-Physical Systems: Theory & Applications*.



Gang Pan (Senior Member, IEEE) received the BSc and PhD degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a professor with the College of Computer Science and Technology, Zhejiang University. He has published more than 100 refereed papers and was a visiting scholar with the University of California, Los Angeles, from 2007 to 2008. His research interests include pervasive computing, computer vision, and pattern recognition.