Florets for Chiplets: Data Flow-aware High-Performance and Energy-efficient Network-on-Interposer for CNN Inference Tasks

HARSH SHARMA, Washington State University, Pullman, WA, USA
LUKAS PFROMM, University of Wisconsin Madison, Madison, WI, USA
RASIT ONUR TOPALOGLU, Topallabs, Poughkeepsie, NY, USA
JANARDHAN RAO DOPPA, Washington State University, Pullman, WA, USA
UMIT Y. OGRAS, University of Wisconsin Madison, Madison, WI, USA
ANANTH KALYANRAMAN and PARTHA PRATIM PANDE, Washington State University, Pullman, WA, USA

Recent advances in 2.5D chiplet platforms provide a new avenue for compact scale-out implementations of emerging compute- and data-intensive applications including machine learning. Network-on-Interposer (NoI) enables integration of multiple chiplets on a 2.5D system. While these manycore platforms can deliver high computational throughput and energy efficiency by running multiple specialized tasks concurrently, conventional NoI architectures have a limited computational throughput due to their inherent multi-hop topologies. In this paper, we propose Floret, a novel NoI architecture based on space-filling curves (SFCs). The Floret architecture leverages suitable task mapping, exploits the data flow pattern, and optimizes the inter-chiplet data exchange to extract high performance for multiple types of convolutional neural network (CNN) inference tasks running concurrently. We demonstrate that the Floret architecture reduces the latency and energy up to 58% and 64%, respectively, compared to state-of-the-art NoI architectures while executing datacenter-scale workloads involving multiple CNN tasks simultaneously. Floret achieves high performance and significant energy savings with much lower fabrication cost by exploiting the data-flow awareness of the CNN inference tasks.

CCS Concepts: • **Hardware → Analysis and Design of Emerging Devices and Systems; • On-chip Resource Management; • Emerging Architectures; • System on Chip; • Algorithms; • Interconnects**;

Additional Key Words and Phrases: Chiplet Architecture, In-Memory Compute, CNN Inferencing, Server-Scale Computing, High-Performance Computing, Space Filling Curve, Network-on-Package

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2023. This work was supported, in part by the US National Science Foundation (NSF) under grants CNS-1955353 and Semiconductor Research Corporation under task ID 3012.001 and task ID 3014.001.

Authors' addresses: H. Sharma, J. R. Doppa, A. Kalyanraman, and P. P. Pande, Washington State University, School of Electrical Engineering and Computer Science, Pullman, WA, 99163, USA; emails: {harsh.sharma, doppa, ananth, pande}@wsu.edu; L. Pfromm and U. Y. Ogras, University of Wisconsin-Madison, Department of Electrical and Computer Engineering, Madison, WI, 53706, USA; emails: lukaspfromm@gmail.com, uogras@wisc.edu; R. O. Topaloglu, Topallabs, Poughkeepsie, NY, USA; email: rasit@topallabs.com.

1539-9087/2023/09-ART132 $15.00
https://doi.org/10.1145/3608098

ACM Reference format:
Harsh Sharma, Lukas Pfromm, Rasit Onur Topaloglu, Janardhan Rao Doppa, Umit Y. Ogras, Ananth Kalyanraman, and Partha Pratim Pande. 2023. Florets for Chiplets: Data Flow-aware High-Performance and Energy-efficient Network-on-Interposer for CNN Inference Tasks. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 132 (September 2023), 21 pages.
https://doi.org/10.1145/3608098

1 INTRODUCTION

Chiplet-based architectures that integrate multiple small dies on an interposer are drawing the attention of leading silicon manufacturers due to their higher energy efficiency and lower fabrication cost [1]. Chiplet-based systems (also known as 2.5D systems) connect multiple small dies (chiplets) through a network-on-interposer (NoI). Designing chiplet-based systems targeted for machine learning (ML) workloads is a relatively unexplored and promising direction since ML is becoming ubiquitous in many real-world applications.

ITRS 2.0 and IRDS roadmaps highlight the unprecedented need for memory and processing over the next decade [2–4]. This need dictates the design of large-scale chips with high memory and compute capability, offering a high degree of parallelism. Such large-scale chips include multiple processing cores, scaling from a few tens to even hundreds. This large-scale integration significantly increases the area of monolithic chips [2]. One of the major challenges in the silicon industry is the exploding fabrication cost as the monolithic chips approach the reticle limit. The chiplet-based design concept offers a promising solution for reducing the manufacturing cost of large monolithic chips [1].

Recent works have proposed several NoI architectures for efficient communication between multiple chiplets on a 2.5D system [5–8]. Existing NoI architectures assume a single and typically fixed application workload executed one at a time, so that the NoI can be optimized for a specific application class mapped onto the chiplet-based system. Offline application-specific NoI optimization is challenging in some real-world settings for two main reasons. First, multiple application workloads with varying inputs may need to be executed simultaneously in a real-world scenario (e.g., inferencing for different images using the same deep model). Second, various types of workloads may appear at any given time (e.g., inferencing tasks with different deep models). Specifically, the mapping of the neural layers onto the chiplets needs special attention for multiple concurrent convolutional neural network (CNN) based inference tasks. Since each neural layer of a CNN typically sends data to the subsequent layer (i.e., the data flow graph is mostly linear), consecutive neural layers need to be mapped to neighboring chiplets to reduce latency. Existing NoI architectures are primarily based on standard multi-hop regular topologies such as mesh, torus, etc. These NoI architectures do not guarantee contiguously placed chiplets to map successive neural layers. Hence, we aim to design an NoI architecture where the chiplets are connected in a contiguous path (through NoI) so that the communicating neural layers are highly probable to run on neighboring chiplets without introducing a significant volume of long-range and multi-hop data exchange. Multiple CNN inference workloads (e.g., object detection, scene understanding in self-driving cars, augmented/virtual reality) frequently appear on the cloud infrastructure where multiple users schedule requests concurrently [9, 10]. Below, we describe occurrences of multiple CNNs in server-scale applications, encompassing various real-world scenarios:

- **Real-time video analytics:** Real-time video analytics is a challenging task that requires high performance and low latency. Multiple CNNs can be used to improve the performance and accuracy of real-time video analytics. For example, one CNN can be used to detect objects in a video stream, while another CNN can be used to classify those objects. This

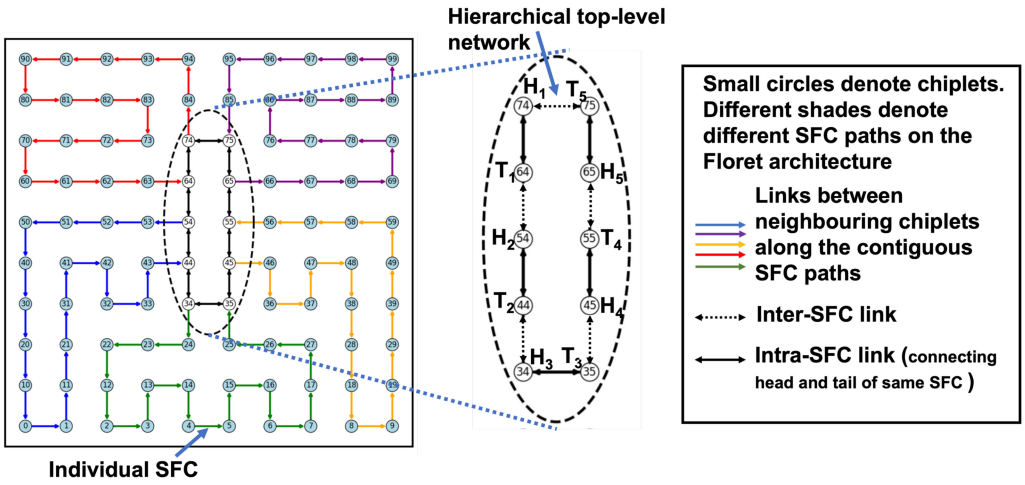

Fig. 1. Illustration of the SFC-based architecture called Floret for a 100-chiplet-based system with five SFCs on the interposer network. The top-level network allows continuity among the multiple SFCs on the NoI.

can be used for applications such as security surveillance, autonomous driving, and video content analysis [53].

- **Cloud computing:** Cloud computing is used to process large amount of data, which is generally expensive. Multiple CNNs can be used to improve the performance and cost-effectiveness of cloud computing. For example, multiple CNNs can be used to process different parts of a large dataset in parallel to create ensemble models. This can help to reduce the time to process the dataset, and it can also help to reduce the cost of cloud computing. Moreover, ensembles of multiple CNNs are effectively utilized in Facebook servers to provide image tagging, feed suggestions among other applications [55].
- **Edge computing:** Multiple CNNs can be used to process data locally at the edge. This can help to improve performance and reduce latency and can protect sensitive data. Specifically, this will improve performance and reduce latency for applications that require real-time processing of data as in the case of augmented/virtual reality (AR/VR) applications [54].

Prior studies sought to improve cloud capacity, application scheduling, and resource utilization while executing ML workloads concurrently on the cloud [11, 12]. In this work, our aim is to capture cloud-scale computing via chiplet-based systems. We propose a novel NoI topology inspired by space-filling curves (SFCs) referred to as *Floret*. An example is shown in Figure 1. The proposed solution enables incoming neural layers associated with CNN inference tasks to be mapped onto contiguous chiplets to avoid long-range communication. Specifically, we leverage the space-filling property to generate a path where a single curve, without any gaps, traverses the area of the interposer with no closed loops. We first divide the chiplet-based system into multiple SFCs. Each SFC stitches a set of chiplets along the 2D planar path, as illustrated in Figure 1. Each SFC consists of a head and a tail connecting a group of chiplets in a contiguous path. We also need to minimize the inter-SFC path length among the non-overlapping SFCs to reduce latency in long-range data exchanges.

The advantages of the proposed mapping along the space-filling path of the NoI are two-fold. First, neural layers of any CNN task get mapped to contiguous chiplets and executed in the order they appear until the system is fully utilized. Second, the space-filling NoI architecture, which minimizes the inter-SFC data exchange, reduces the latency when we need to find contiguous

chiplet resources belonging to different SFCs. Instead of one monolithic SFC, we use multiple SFCs to introduce inherent redundancy in the system, which is beneficial when executing multiple CNN inference tasks concurrently; hence the name "Floret" – to imply a cluster of multiple connected SFC "petals". Experimental evaluation with multiple CNN inference tasks running concurrently for various system sizes demonstrates that SFC-enabled NoI outperforms existing NoI architectures with significant energy savings.

Contributions: The key contribution of this paper is the algorithmic development to enable Floret NoI optimized for CNN inference tasks and its comprehensive experimental evaluation. Our major contributions include:

(1) We propose a novel NoI architecture called *Floret* with multiple non-overlapping SFCs specifically targeting running multiple concurrent CNN inference tasks.
(2) We propose a new type of SFC called the Floret curve that is targeted for chiplet-based systems, and using this Floret curve we propose a novel NoI architecture along with a mapping algorithm to efficiently map successive neural layers to contiguous chiplets for achieving high performance and energy efficiency.
(3) Experimental results show that the Floret architecture can achieve up to 58% and 64% reduction in latency and energy respectively compared to state-of-the-art counterparts.

The rest of the paper is organized as follows. Section 2 describes the relevant prior work on 2.5D systems and NoI architectures. Section 3 presents the design and optimization principles for executing the CNN inference tasks on the *Floret* architecture. Section 4 presents the detailed experimental results and analysis. Finally, Section 5 concludes the paper by highlighting the salient contributions and pointing to the future directions.

2 RELATED WORK

The manufacturing cost of monolithic chips is increasing rapidly with the growing die area requirements of emerging applications. First, fewer large chips can be integrated for a given wafer size than many smaller ones, decreasing the area utilization [2]. Second, when defective, a larger die wastes more silicon area than its relatively smaller counterparts. Most chip vendors and foundries are moving towards non-monolithic alternatives such as 2.5D interposer-based systems to partition the on-chip resources into smaller discrete cores called chiplets [1, 13, 14]. The emergence of 2.5D chiplet platforms provides a new avenue for compact scale-out implementations of various deep learning (DL) applications. Integrating multiple small chiplets on a large interposer enables not only significant cost reductions and higher manufacturing yield compared to 2D ICs [1], but also better thermal efficiency than 3D ICs [13] and ease of heterogeneous integration [2]. Designing both general-purpose and application-specific 2.5D-based systems have been explored so far. The design and fabrication of interposers also add significant non-recurring engineering costs and development cycles which might be prohibitive for application-specific designs having low volume. To address this challenge, a General Interposer Architecture (GIA) is proposed, to amortize costs and accelerate integration flows of interposers across different chiplet-based systems effectively [15].

The recently proposed SIAM framework enables fast design space exploration of 2.5D-based systems [6]. SIAM employs ReRAM-based chiplets that can be used both as memory and to perform in-situ multiply-and-accumulate (MAC) operations [6, 16]. Since DL workloads rely heavily on such MAC operations, ReRAM-based architectures are excellent candidates for DL training and inferencing [17–19]. ReRAM-based heterogeneous architectures were proposed to improve the accuracy of trained models while also addressing communication bottlenecks [20, 21]. Thus, ReRAM-based 2.5D architecture can outperform CPUs/GPUs for almost all types of DL workloads

as they support near-data computation [22]. Recent prior work has devised ReRAM-based DL accelerators that overcome the limited write endurance and high write energy costs of ReRAMs [23, 24]. Yet, the evaluation framework proposed in SIAM assumes a mesh-based NoI, which is not scalable for multiple concurrent CNN tasks and large system sizes. SIMBA introduces tiling optimizations on fixed NoI topologies for executing DL model such as ResNet50 [7]. NN-Baton focuses on choosing a specific design allocation across several benchmarks on a fixed topology [8]. However, NN-Baton does not consider the scale of the data centers where the number of DL parameters reach order of billions. To this end, silicon-photonic interposers have been proposed to improve the latency and bandwidth [25]. A reconfigurable Silicon-Photonic 2.5D NoI architecture is proposed to dynamically deploy inter-chiplet photonic gateways to improve the overall network congestion. An application specific architecture using photonics called BiGNoC is proposed, which highlights how network-on-chip can be designed for manycore chiplet-based system to meet the unique communication requirements of big data analytics applications but at the intra-chiplet level [26]. Moreover, the NoI paradigm becomes crucial due to the high communication demand arising from integrating an increased number of chiplets on the same substrate [1, 6].

Space-filling curves (SFCs) represent a specialized class of algorithmic mapping techniques that are widely used to generate locality-preserving data structures in numerous scientific applications that do spatial and range queries [27–29]. More specifically, an SFC maps a multi-dimensional point cloud onto a single dimension; therefore, each SFC represents a linear ordering of the input set of points. Numerous types of SFCs have been defined over the decades, including simple schemes such as row/column major curves to more sophisticated curves such as the Hilbert curve [28], Morton or Z-curve [30], or onion curve [31]. For a review of classical SFCs, please refer to [32, 33]. SFCs come with various provable properties. One such property concerning locality is called clustering [34, 35], which is a measure of the number of hops taken along the linear ordering of an SFC, to access neighboring data in the multi-dimensional point cloud. Some curves, such as the Hilbert and Z-curves in particular, have demonstrated a better clustering property over others both in theory and practice [32, 34, 36, 37]. SFCs have been predominantly used in databases and in parallel scientific computing [37]; for exploring data layouts in memory for multi-core platforms [38]; and in bioinformatics for creating locality-preserving layouts for DNA nanostructures [39], sequence alignment [40] and phylogenetic inference [41].

Despite their popularity in various engineering domains, SFCs have not yet been explored for designing NoI-based manycore chiplet architectures or for accelerating machine learning workloads. Most previously proposed NoI architectures are based on conventional multi-hop networks, like mesh and torus. Recently, the Kite family of NoI topologies has been proposed for a 2.5D-based system considering synthetic traffic/workloads [5]. However, Kite is also primarily based on a Torus architecture, and all such regular NoI architectures are not workload-aware. Emerging DL applications use more than a billion parameters [6, 17]. We increasingly rely on large-scale manycore computing platforms to execute these massive workloads. It has been shown that a significant portion (about 30–75%) of the overall execution time of DL workloads arises from the communication among the processing elements, which is hidden by overlapped computation [42]. This characteristic necessitate communication aware paradigms for designing such NoI architectures for DL workloads. Recently, application-specific NoI design for 2.5D-based systems has been explored using ML-based techniques [17]. However, this work is oblivious to the occurrence of real-world data-center scale ML application workloads for executing concurrent CNN inference tasks with unseen neural networks. The goal of this paper is to precisely fill this important gap in the existing state-of-the-art NoI architectures by proposing novel design principles for chiplet-based systems, which are well-suited for executing multiple CNN inference tasks concurrently.

3 DESIGN AND OPTIMIZATION OF THE SFC-ENABLED NETWORK-ON-INTERPOSER

This section presents the overview and design methodology of the *Floret* architecture. We start by presenting the salient features of the chiplet configuration considered here. We then describe the key principle to design the overall *Floret* architecture using multiple space-filling curves. It should be noted that the proposed methodology is generic, and it can be used to design other large-scale 2.5D chiplet systems. This work focuses on the NoI level optimization aspects without modifying the design of individual chiplets.

3.1 ReRAM-based 2.5D Chiplet Architecture

Processing-in-memory (PIM) is a promising technique to accelerate deep learning (DL) workloads [19]. PIM-enabled architectures improve energy efficiency by reducing communication between computing cores and the main memory [43]. Crossbar arrays (CBAs) are the most popular representation for PIM. They are highly efficient for matrix-vector multiplication (MVM), which forms the core of many DL and scientific computing algorithms. Prior work has investigated binary CBAs based on various memory technologies, including phase change memory (PCM), Resistive Random Access Memory (ReRAM), Spin-Transfer Torque Magnetic RAM (STT-MRAM), and Ferroelectric Field-Effect Transistor Memory (FeFETs), and has experimentally demonstrated their functionality at various scales [44–46]. In this work, we employ ReRAM-based chiplets as the enabling technology to accelerate CNN inference tasks, noting that the proposed architecture and associated design optimization methodologies are also applicable to other CBA-based PIM chiplets. The chiplets are connected through NoI routers and links, which enable high-bandwidth communication. Each chiplet is composed of 16 tiles and peripheral circuits such as accumulator, buffer, activation units (ReLU in our work), and pooling unit. Within each chiplet, a mesh-based network-on-chip (NoC) connects the tiles, where each tile comprises multiple processing elements (PEs) that consists of 128×128 ReRAM crossbar arrays. It should be noted that within chiplets the number of tiles is limited (e.g., 16 tiles in the Floret architecture). Hence, a simple mesh-based NoC is sufficient as there is no scope for any significant multi-hop or long-range data exchange. In other words, the intra-chiplet latency and energy costs are negligible compared to inter-chiplet data exchange costs. Therefore, we focus on optimizing the NoC/NoI interconnectivity at the entire system level. Note that the Floret architecture is independent of the NoC architecture used within a chiplet, and so our proposed design methodology is generic enough to work with any interconnect used within chiplets.

The target chiplet architecture has 40 PEs inside each tile, connected through an H-Tree-based point-to-point network. In our approach, we assume that all CNN weights are transferred to the ReRAM chiplets from the DRAM before performing CNN inference, which is consistent with previous investigations [18, 23, 47]. Following prior work, we also assume that the global buffer is available for processing weights due to storing activations from the previous layer for a residual addition operation that is prevalent in dense (DenseNet) and residual (ResNet) class of neural networks [6]. The number of PEs necessary to map a neural layer is dependent on several factors, including kernel size, number of input and output features, and bit precision. These factors determine the number of tiles required for each neural layer, as well as the total number of chiplets needed to map the whole neural network. It is possible to fit multiple layers on a single chiplet or a single layer to spread across multiple chiplets. In a server-scale scenario, the number of CNN parameters can reach billions, leading to heavily utilized chiplets.

3.2 Space-filling Curve Enabled NoI Architecture

The problem: Given the need to execute various deep learning tasks simultaneously [14, 42], modern-day servers and high-end processors need to be designed to target a workload consisting

of a mixture of tasks. We consider CNNs with different neural layer architectures – including linear (e.g., VGG), residual (e.g., ResNet), and dense (e.g., DenseNet) connections – for performing inference tasks while designing a chiplet-based system. However, mapping different CNNs dynamically to a chiplet-based system is challenging. The common property of CNN inference tasks is that activations flow from the i^{th} layer to the $(i+1)^{th}$ layer. Hence, there is a need to maintain contiguity on the physical NoI layer, to the extent possible, between any two consecutive neural layers to reduce communication overhead. Since existing NoI architectures are primarily based on standard multi-hop regular topologies such as a mesh or a torus, it may not always be possible to find contiguously placed chiplets available to map successive neural layers. If two consecutive layers of a CNN are mapped far apart, it will lead to long-range multi-hop communication through the NoI. This, in turn, will degrade the performance and energy efficiency of the NoI. Hence, our objective is to design an efficient NoI architecture which is capable of co-locating adjacent neural layers.

In theory, this design problem can be viewed as one of embedding a linear ordering (i.e., an SFC) of chiplets over the given topology. However, there may be multiple CNN tasks that need to be dynamically mapped to the system, and each such task may consist of different numbers of neural layers. Furthermore, the number of chiplets needed to execute each layer may also vary. Therefore, the problem becomes one of generating *multiple SFCs*, each with its own sequence of chiplets to map to the neural layers of any of the tasks. Moreover, as the different CNN tasks complete, the chiplets used for that task need to be *reassigned* to newer tasks. If a consecutive sequence of chiplets is not sufficient to accommodate all the layers of a CNN task, the spill over layers will need to utilize chiplets in *other parts* of the NoI (i.e., from other SFCs) so as to ensure successful completion. Therefore, the placement of the SFCs and the resulting hop separation between them become important measures to reducing CNN task execution times. Taken together, these factors – i.e., the need to accommodate multiple SFCs, the dynamic nature of mapping those SFCs to multiple CNN tasks, and the need to potentially hop from one SFC to another (for the same task) – all make this a challenging problem, one where classical SFC designs may not apply.

Approach: In this work, we present a custom-designed SFC called the *Floret* curve that is equipped to address all the aforementioned challenges. In particular, our approach connects the chiplets (in the order the neural layers are mapped) along the contiguous path formed by the Floret architecture in a two-dimensional (2D) space, as illustrated in Figure 1. The intuition behind the Floret architecture is to subdivide a multi-dimensional space into smaller contiguous segments (or individual SFCs), and then to stitch those pieces together; hence the term "Floret" as the resulting topology can be viewed as a cluster of individual SFCs (or petals). The resulting curve is a continuous, non-intersecting (planar) path that covers all the chiplets in the system – hence the term "space-filling".

Definition of a Floret curve: More formally, let C denote the set of n chiplets distributed across a given 2D grid coordinate system. The chiplets are numbered arbitrarily from $[0, n-1]$. For example, the chiplets in Figure 1 are numbered in row major fashion along the grid. Given n and a constant λ, a *Floret curve* (denoted by Π) is a collection of λ individual SFCs $\{\Pi_0, \Pi_1, \ldots \Pi_{\lambda-1}\}$. Let $\psi = \lceil \frac{n}{\lambda} \rceil$. Then, each of the λ SFCs represents a sequence of ψ chiplets that are contiguously placed along the grid. In other words, each SFC covers a distinct subset of size ψ chiplets such that no two SFCs intersect. Each SFC (Π_i) has a dedicated *head* (h_i) and a corresponding *tail* (t_i) on the other end, connecting $\psi - 2$ chiplets in between. As an example, Figure 1 shows a Floret curve with five SFCs. One can view this Floret curve also as a hierarchical design with two levels, where the top level corresponds to the λ head-tail pairs and the next level consists of all the individual SFCs.

3.2.1 Algorithm for Designing Floret Curves. Next, we describe our algorithm to design a Floret curve, given C, the set of n chiplets on a 2D grid,[1] and λ, the number of different SFCs. At a high level, the algorithm has two major steps. First, a subset of λ chiplet pairs of the form $\langle$head h_i, tail $t_i\rangle$ are selected, one pair for each SFC Π_i. Next, using the head and the tail chiplet pairs as end points of a Π_i, we fill the remaining $\lambda - 2$ chiplet locations for Π_i. Algorithm 1 shows the pseudocode for our design approach. In what follows, we provide details for each step.

For the first step of choosing λ head-tail chiplet pairs, note that the search space is $\binom{n}{2\lambda}$ in theory. However, during mapping phase, since the same CNN task may possibly use chiplets from two or more SFCs, it is important to reduce the average number of hops separating the tail of an SFC to a head of another SFC. Therefore our search objective becomes one of minimizing this average path length d between the tail of one SFC to the heads of the other non-overlapping SFCs:

$$Minimize: \quad d = \frac{1}{p} \sum_{i,j \in [0,\lambda-1]} |t_i - h_j|_{where\ i \neq j, \quad p = 2\binom{\lambda}{2}} \tag{1}$$

Here the distance between any tail-to-head pair is calculated as the Manhattan distance over the 2D grid. Minimizing this average distance measure d is imperative as communication delays between tail of one SFC and the head of the next SFC can have a significant impact on the overall system performance. We follow an iterative approach to identify λ head-tail pairs. Intuitively, concentrating all the λ head-tail pairs at the center of the NoI architecture is expected to reduce the number of hop counts between an arbitrary tail and an arbitrary head. Alternatively, if one were to spread out the head-tail pairs across the NoI, inter-SFC hop count can only increase. Using this simple yet key insight, our algorithm selects head-tail pairs from the center of the NoI. In particular, we identify a subset of 2λ chiplets along a pair of central columns (as shown in Figure 1. If the length of a column is not adequate to accommodate all the λ chiplet pairs, then we iteratively identify further evenly spaced pairs of columns from either side of the center until all pairs are identified. This algorithm effectively performs a block decomposition of the columns starting from the center and radiating outwards.

ALGORITHM 1: Algorithm for designing floret architecture

Input: $\mathcal{C}$: a 2D grid of n chiplets represented as a graph $G(V, E)$; λ: the desired number of SFCs
Output: A list of λ SFCs: $\Pi = \{\Pi_0, \Pi_1, \ldots \Pi_{\lambda-1}\}$, where each $\Pi_i : C_\psi \rightarrow [0, \psi - 1]$, $\psi = \lceil \frac{n}{\lambda} \rceil$, and $C_\psi \subseteq \mathcal{C}$ of size ψ

1: $[\langle H, T\rangle] \leftarrow$ Assign a list of λ $\langle$head, tail$\rangle$ chiplet position pairs in $\mathcal{C}$
2: **for all** $\langle h_i, t_i\rangle \in [\langle H, T\rangle]$ **do**
3: Initialize $\psi \leftarrow \lceil \frac{n}{\lambda} \rceil$ `/* i.e., TSP tour length for each SFC */`
4: Initialize Π_i to an empty array of (TSP tour) size ψ
5: $\Pi_i \leftarrow ComputeTSP(G(V, E), \langle h_i, t_i\rangle, \psi)$
6: Update graph G by removing all edges incident on Π_i
7: **end for**
8: $\Pi \leftarrow \bigcup_i \Pi_i$
9: **return** Π

Once the head-tail pairs are selected, the next step is to fill (or complete) each of the λ SFCs from their respective heads to their tails (as shown in Algorithm 1: lines 2 through 7). The goal is to create each of the λ SFCs, Π_i with head h_i and tail t_i, of length ψ. The important design consideration is to maintain contiguity for the chiplets assigned to the same SFC. This problem can be effectively solved as an instance of the Euclidean traveling salesman problem (TSP) problem

[1]Even though the algorithm presented is for a 2D grid system of chiplets, we argue later on how the algorithmic methodology is generic enough to be extended to other symmetric topologies [5].

[48]. More specifically, let $G(V, E)$ denote the initial (planar) graph corresponding to the 2D grid system – i.e., V corresponds to the set of all n chiplets, and E consists all the 1-hop neighboring chiplet pairs on the grid. Our algorithm iteratively enumerates one SFC at a time (for loop in line 2 of Algorithm 1), such that during the i^{th} iteration we enumerate SFC Π_i. Since an SFC is a linear ordering of ψ chiplets contiguously located along the grid, the problem of finding an SFC can be reduced to one of finding the Hamiltonian subpath of length ψ on the planar G. Furthermore, to facilitate tail to head inter-SFC transfers during mapping, we treat it as a planar Hamiltonian cycle problem. Since the cost is dictated by the number of hops (along the grid), the goal becomes one of computing a minimum cost planar Hamiltonian cycle, which is an instance of the Euclidean TSP problem [49]. Therefore, as shown in lines 3–5 of Algorithm 1, we call a TSP solver on G to obtain each SFC. It should be noted that the graph G needs to be updated after the enumeration of each SFC. Specifically, at the end of every step i, after we generate Π_i, we remove all edges in E that are incident on the vertices selected as part of Π_i. This step ensures none of the chiplets from previous SFCs are eligible for inclusion in any of the subsequent SFCs – thereby ensuring that all SFCs are mutually disjoint in their chiplet space.

For the TSP computation step in line 5 of Algorithm 1, we implemented a recursive backtracking-based TSP solver that works on the tour length ψ. This implementation explores all possible tours through a recursive search process. Backtracking is a powerful technique for solving the Euclidean TSP (over planar graph G), which can be computationally expensive for large problem instances [49]. However, this is a preprocessing step (and is hence a one-time cost) and the sizes of $G(V,E)$ in practice is expected to be small for the target platforms. For instance, computing all the SFCs for a system with $n = 36$ and $\lambda = 6$ SFCs, took only 10 milliseconds.

Additional remarks:

(a) The TSP formulation makes our algorithmic approach more generic to be extended to design Floret curves for additional topologies and not just for the 2D grid (which we selected for ease of exposition). In particular, any NoI topology can be represented in the form of a graph, and our TSP solver implementation does not make any assumptions on planarity of the graph. However, as the planarity assumption is removed, then the degree distribution of the vertices in the graph can no longer be bounded to a constant. This could lead to increased execution times for the TSP solver.

(b) Even though the proposed algorithm for Floret curve design was presented for a 2D grid system of chiplets, the design methodology is generic enough to be extended in principle to other *symmetric* topologies – e.g., Kite, Butter Donut, Double Butterfly [5]. This is because our algorithm to assign the head-tail pairs simply relies on starting at the center of the NoI and radiating outwards iteratively. However, given that CNNs primarily rely on communicating between neighboring layers, a simple 2D grid topology is sufficient to serve as the breadboard for generating our Floret curve architecture.

(c) A key parameter to the Floret architecture design is the number of SFCs (λ). Intuitively, having too many SFCs unnecessarily increases the top-level network size. On the other hand, too few SFCs will reduce the number of router ports, which could degrade redundancy across SFCs and could hamper the overall achievable performance. Minimizing the average hop count between tails and heads of non-overlapping SFCs provides us with the optimum number of SFCs and the router port configurations for each system size. Section 4.2 evaluates this tradeoff in selecting an optimum number of SFCs.

3.2.2 Algorithm for Mapping CNN Workloads to the Floret Architecture. We describe the algorithm to dynamically map a workload of CNN tasks to the Floret architecture (as designed in Section 3.2.1). The input is a *workload* consisting of a set of CNN *tasks* ($W = \{w_i\}$), each consisting

of multiple neural layers. The output is a mapping $\Phi : W \rightarrow 2^C$, which maps each w_i to a subset of c_i chiplets along the Floret curve; here, c_i denotes the number of chiplets required to execute all the neural layers of w_i. The value of c_i can be precomputed by adding the number of chiplets required for computing each layer of a CNN tasks. Note that multiple layers of an individual CNN can fit within a single chiplet (i.e., $c_i \leq 1$), or alternatively, a single layer could require multiple chiplets (i.e., $c_i > 1$). However, with CNN inference tasks, communication typically occurs between two consecutive layers. For this reason, the Floret architecture is well positioned to keep the communicating pairs of chiplets near to one another.

Algorithm 2 details the major steps of the mapping procedure to map W to the Floret architecture. We start by considering the workload W as a queue of multiple CNN tasks. For each $w \in W$, we first compute the number of chiplets (c) required. Initially, all chipets across all λ SFCs of Π are considered available. We track a *next* pointer to point to the next chiplet along Π that is due for assignment. Initially, *next* is initialized as the head chiplet of the first SFC (Π_0).

The major function that computes $\Phi(w)$ for any given task w is $BlockAssign(\mathrm{w}, \Pi,\ next,\ c, n')$, shown in line 5 of Algorithm 2. This function maps the task w to a sequence of c chiplets, starting from the *next* position along Π. Note that the actual chiplet coordinates for this *next* position is given by $\Pi^{-1}(next)$. The *BlockAssign* function returns when all the c chiplets were successfully assigned in the mapping process. During the course of mapping, there are two subcases to consider. (a) When all the chiplets along the current SFC have been assigned, we move on to another SFC. This SFC is chosen based on the proximity of its head to the tail of the current SFC. Subsequently, the assignment of the remaining layers resumes on the next SFC. This process is iterated until all layers are successfully assigned. (b) Note that it is possible that along the assignment process, the next chiplet to be assigned is occupied with another task. In this case, the procedure waits until it becomes available. Once all the chiplets in the system are utilized, then we will have to wait till a set of contiguous chiplets required for the incoming neural layer becomes free. This would happen when a prior loaded CNN finishes execution on the Floret, which would in turn release a contiguous region for the new CNN. Once contiguous chiplets become available, then the inter-chiplet data flow still follows the one-hop path.

ALGORITHM 2: Mapping algorithm for floret architecture

Input: Workload with multiple CNNs ($\mathcal{W} = \{w_i\}$) each with multiple layers
$\mathcal{C}$: the set of n chiplets ordered by the Floret SFC as $\Pi : C \rightarrow [0, n-1]$
Output: Mapping of each workload $w_i \in \mathcal{W}$ to a distinct subset of chiplets
(i.e., $\Phi : \mathcal{W} \rightarrow 2^{\mathcal{C}}$ such that $\Phi(w_i) \cap \Phi(w_j) = \emptyset$ for any $w_i, w_j \in \mathcal{W}$ where $w_i \neq w_j$)
1: Initialize $next = 0$ `/* allocation to start at the first chiplet Π(1) */`
2: Initialize $n' = n$ `/* the running count of the number of available chiplets */`
3: **for all** $w \in \mathcal{W}$ **do**
4: $c \leftarrow$ number of chiplets required by w (rounded to the next integer)
5: $\langle \Phi(w), n' \rangle = BlockAssign(w, \Pi, next, c, n')$ `/* Map` w `to a sequence of` c `chiplets starting at` *next* `position along the SFC, and returns also the updated` n' `*/`
6: Update $next \leftarrow (\Phi(w).lastindex + 1) \mod n$
7: **end for**
8: **return** Φ

The above mapping approach has multiple advantages:

- First, chiplet resources become available for new layer allocation in the order they were mapped. The activations would be transferred sequentially among contiguously placed chiplets as the computation moves from the first layer to the output layer of the CNN.

- Second, we utilize all the available chiplets as per the computational requirements of the neural layers.
- Third, the mapping algorithm is deadlock-free, because the mapping process treats the list of tasks (W) as a queue, assigning one CNN task at a time. Deadlocks could happen only if either there is a cyclic dependency between two tasks (which is not possible here as CNN tasks are mutually independent), or if there are two *concurrent* mapping threads that are stuck and waiting for one another to release their resources (also not possible here due to the sequential queue-based mapping of the workloads).
- Finally, our mapping approach exploits the inherent redundancy built in the NoI architecture via multiple available SFCs. In particular, if during the course of assignment, we reach the tail of one SFC, we have more than one option for selecting the next SFC. For instance, in the Floret architecture shown in Figure 1, tail T_1 is connected to two heads (H_1, H_2) within just 1-hop distance. In fact, this connectivity can further be increased to include H_5 as well if we decide to retain the original 2D grid level links in the top-level network. This implies that if an assignment reaches T_1 and if there are more chiplets needed to complete that inference task, then there are between 2 to 3 options for switching to another SFC, all at a 1-hop distance. Our mapping algorithm can select the next SFC in a reconfigurable manner. This property is also vital to extend our architecture in the future toward providing fault-tolerant executions. A formal analysis of these properties of the Floret architecture could provide further insights; however, it is out of scope for this paper. Instead, we focus on the key ideas, concepts, and a thorough experimental evaluation.

4 EXPERIMENTAL RESULTS

In this section, we present a detailed performance analysis and experimental evaluation of the proposed NoI architecture for various CNN inferencing tasks. We also present a detailed comparative performance evaluation with respect to existing state-of-the-art NoI designs for chiplet-based platforms.

4.1 Experimental Setup

4.1.1 System Specification and Evaluation Setup. To demonstrate the scalability of the Floret architecture, we consider four different system sizes (n) with 36, 64, 81, and 100 chiplets. We use a modified NeuroSim to partition and map CNN tasks onto a 2.5D-based system [50]. The inter-chiplet traffic is generated by the activations between the neural layers. Each chiplet in our design has 64KB of buffer space to compute the activations associated with the skip connections, which flow through the same NoI links. This buffer size was sufficient for computing residual activations, [7, 14]. When there are non-contiguous neural layers, the inter-chiplet data exchange involves multi-hop paths. Each chiplet covers about $2.64mm^2$ area, including the peripherals. All the NoI topologies are simulated using the BookSim simulator [51]. The inputs to the BookSim simulator are the connectivity between NoI routers and the inter-chiplet traffic for the concurrent CNN inference tasks. It outputs the area, latency, and energy consu mption of the NoI. We use the Nvidia ground-referenced signaling (GRS) parameters for chiplets on a 32nm technology to evaluate the NoI area and power consumption [7]. Table 1 shows the other system-level parameters considered in the performance evaluation [16, 52]. We note that the experimental analysis and performance evaluation considered in this paper is valid for other technology parameters.

4.1.2 Datasets and DL Workloads. We evaluate the Floret architecture on multiple CNN inferencing tasks running concurrently. Table 2 shows different neural networks executed on the corresponding datasets, and their number of parameters. As the system size increases, we use

Table 1. NoI Hardware Parameters Considered for Evaluation

NoI Hardware Parameters	Value
NoI frequency	1.15 GHz
NoI bus width	32
One-hop NoI link length	1.449 mm
Quantization bit	8
Technology	32nm
Link Frequency	0.6 ns/mm

Table 2. List of Neural Networks for Inferencing Along with Their Corresponding Number of CNN Parameters with (a) CIFAR-100, (b) ImageNet Dataset

(a)

Name	Neural Network	Number of Parameters (CIFAR100)
NN_1	ResNet18	1.8M
NN_2	ResNet34	2.79M
NN_3	ResNet50	4.15M
NN_4	ResNet110	9.42M
NN_5	ResNet152	12.96M
NN_6	VGG16	1.67M
NN_7	VGG19	1.91M
NN_8	DenseNet40	1.6M

(b)

Name	Neural Network	Number of Parameters (ImageNet)
NN_9	ResNet18	24.76M
NN_{10}	ResNet34	36.5M
NN_{11}	ResNet50	25.94M
NN_{12}	ResNet101	9.42M
NN_{13}	ResNet110	43.6M
NN_{14}	ResNet152	54.84M
NN_{15}	VGG19	93.4M
NN_{16}	DenseNet169	892.72M

Table 3. List of CNN Tasks in a Workload for Inferencing Along with Their Total Number of Parameters with (a) CIFAR-100, (b) ImageNet Based Dataset

(a) – CIFAR100

Name	List of CNNs in a workload	Total number of parameters
$WL1$	$16NN_2, NN_7, 5NN_3, 3NN_8, NN_5, NN_7, 4NN_4, NN_6, NN_1, NN_3, NN_6$	133M
$WL2$	$NN_7, NN_1, NN_6, NN_1, 11NN_3, 3NN_3, NN_7, 3NN_4$	88M
$WL3$	$NN_7, NN_6, NN_5, 3NN_4, 9NN_8, 4NN_2, 12NN_1, 5NN_3, NN_6$	114M
$WL4$	$NN_5, NN_7, NN_1, NN_3, NN_6, 5NN_3, 3NN_8, 16NN_2, 4NN_4, NN_6, NN_7$	133M
$WL5$	$NN_5, NN_8, NN_1, NN_3, NN_4, 6NN_2, 4NN_6, 11NN_4, 5NN_5, 2NN_6$	240M

(b) - ImageNet

Name	List of CNNs in a workload	Total number of parameters
$WL6$	$16NN_{10}, NN_{15}, 5NN_{11}, 3NN_{16}, NN_{13}, NN_{15}, 4NN_{12}, NN_{14}, NN_9, NN_{11}, NN_{14}$	1.1B
$WL7$	$2NN_{15}, NN_{14}, NN_{13}, NN_{12}, 7NN_{11}, 2NN_{12}, NN_{16}, NN_{12}$	1.4B
$WL8$	$NN_{15}, NN_{14}, NN_{13}, 3NN_{12}, 9NN_{16}, 4NN_{10}, 12NN_9, 5NN_{11}, NN_{14}$	8.8B
$WL9$	$NN_{13}, NN_{15}, NN_9, NN_{11}, NN_{14}, 5NN_{11}, 3NN_{16}, 16NN_{10}, 4NN_{12}, NN_{14}, NN_{15}$	3.8B
$WL10$	$NN_{13}, NN_{16}, NN_9, NN_{11}, NN_{12}, 6NN_{10}, 4NN_{14}, 11NN_{12}, 5NN_{13}, 2NN_{14}$	1.8B

ImageNet-based CNNs with more parameters to illustrate the merits of the proposed architecture. Table 3 shows the naming convention of the CNN tasks in each workload along with their total number of parameters with (a) CIFAR-100 and (b) ImageNet datasets. Tables 3(a) & (b) show the CNNs executed simultaneously on the 2.5D system. Various combinations of the neural networks in Table 2 are executed concurrently to capture the workloads (WL) considered in the experimental setup. We evaluate 36 chiplet system using workloads running for CIFAR-100 dataset. For scalability, we evaluate 64, 81 and 100 chiplet system on ImageNet based workloads as the number of parameters approach in the order of billions. As an example, WL1 consists of sixteen instances of NN_2 (ResNet34), along with one instance of NN_7 (VGG19), and so on. We cover the whole spectrum by randomly choosing each of the CNNs such that at least 90% of the 2.5D system is always

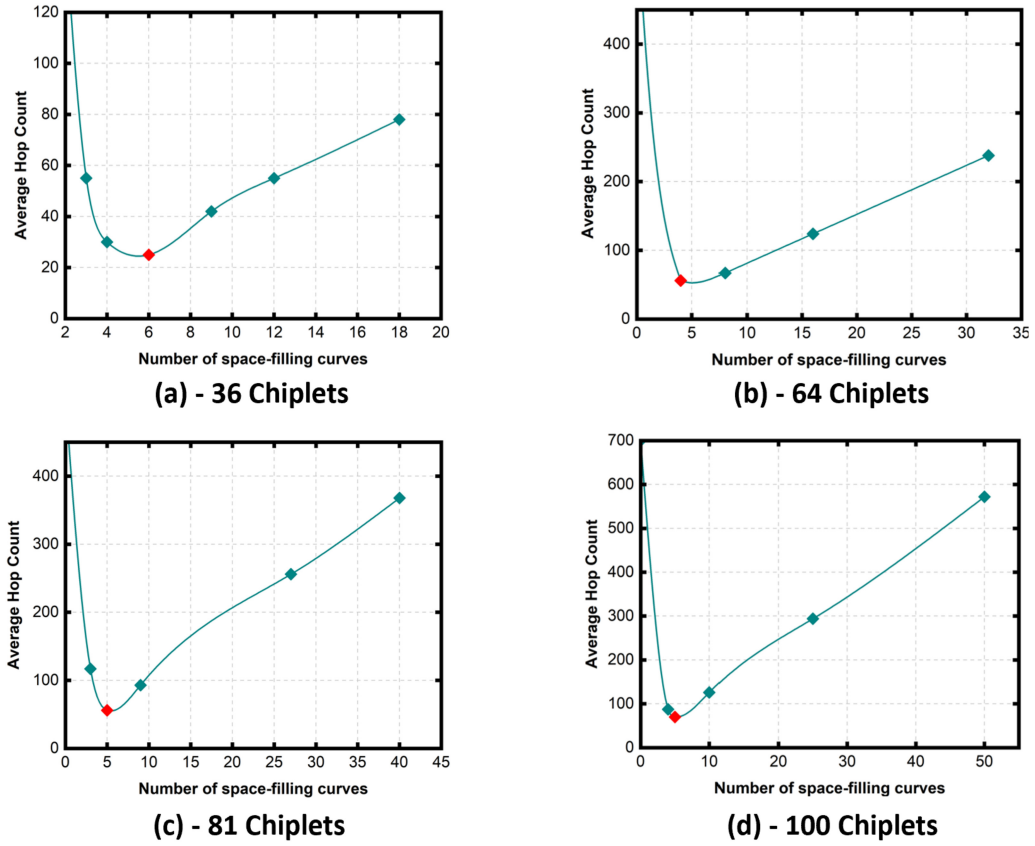

Fig. 2. Illustration of the optimal number of SFC for (a) 36 chiplets, (b)64 chiplets, (c)81 chiplets, and (d) 100 chiplet system.

utilized. Note that the general concept behind our NoI design is applicable to any type of CNN inference tasks.

4.1.3 Baseline NoI Design. We compare the performance of Floret against three baselines: Kite, SIAM, and a recently proposed application-specific NoI architecture SWAP [5, 6, 17]. Kite is primarily a Torus-based NoI, and SIAM is essentially a 2-D mesh NoI. The application-specific SWAP NoI is an irregular architecture where the chiplets and the associated links are placed as per specific design time considerations for a given set of CNN applications. We set the same system parameters and evaluate over the same CNN workloads for all four architectures (Kite, SIAM, SWAP, and Floret) for a fair comparison.

4.2 Optimum Number of SFCs

In this sub-section, we evaluate the optimum number of SFCs which would occur on the interposer network considering the average hop count (H_{avg}) bet ween an y two communicating pair of chiplets for a CNN task. Figure 2 shows the optimum number of SFCs with varying system size. Here, we consider iso-chiplet area configuration, i.e., each individual chiplet is of the same size irrespective of the system size. As the number of chiplets, n, increases from 36 to 64 to 100, the interposer area also increases while the size of each of the individual chiplet remains the same.

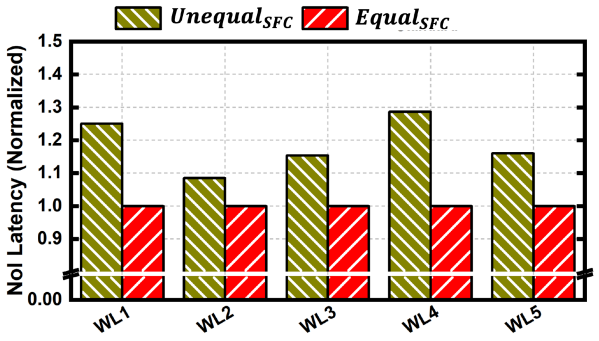

Fig. 3. Normalized NoI latency for the 36-chiplet Floret architecture with equal and unequal SFC lengths. This shows that having unequal SFC lengths is not advantageous compared to having equal length of SFCs.

We observe that the optimum number of SFCs lie between four to six as the number of chiplets vary. Due to the iso-chiplet but increasing interposer area assumption the number of SFCs remains within a limited range for varying system size. These SFC configurations minim ize the averag e hop count of the top level network (6, 4, 5, 5 SFCs in case of 36-,64-,81- and 100- chiplets respectively). Ultimately, the minimization of H_{avg} leads to higher performance benefits of Floret over its counterparts.

4.3 Effect of SFC Lengths

In this sub-section, we evaluate the effect of keeping SFCs of equal length (as is part of our default design) versus allowing them to vary in their lengths on the interposer network. SFCs with varying lengths could lead to traffic imbalance and thereby, latency degradation for the system; whereas an even length reduces such imbalances and could deliver better performance. To test this hypothesis, we experimented with different (unequal) lengths for the SFCs of the Floret architecture, and compared them with the performance derived from the equal length setting. We consider the Floret architecture with 36 chiplets as an example here. For the equal-length SFC configuration, each SFC consist of 6 chiplets. However, for the unequal-length configuration the SFCs contain 8, 7, 7, 5, 4, 5 chiplets respectively. Figure 3 shows the comparison between the latency obtained under these two settings, for a 36-chiplet system. It is clear that the Floret with unequal-length SFC degrades performance compared to the equal-length SFC configuration, corroborating our hypothesis. This happens since when SFCs are of different lengths then the distance between head-tail pairs in the top-level network increases. This results in latency degradation. It should be noted that there are other configurations possible for the unequal-length scenario. In each case, we expect to see similar trends. For brevity, we show the result for only one configuration.

4.4 Variation of Number of Router Ports

Each NoI architecture consists of inter-chiplet routers and links. Since each architecture has different connectivity, this section compares the distribution of the number of router ports in the Floret architecture against the other state-of-the-art counterparts. We also compare the number of links involved in each architecture. Figures 4(a)–(d) show the router-port configurations for all four s ystem sizes considered in this work. We observe that four-port routers are the most frequent ones with Kite. SIAM with mesh NoI mostly consists of routers with three and four ports. In contrast, SWAP primarily uses two- and three-port routers, where the links are on average longer due to the small-world network approach [17]. However, all the routers in Floret except the heads and tails have only two ports. The peak moves towards the left, demonstrating that the frequency of

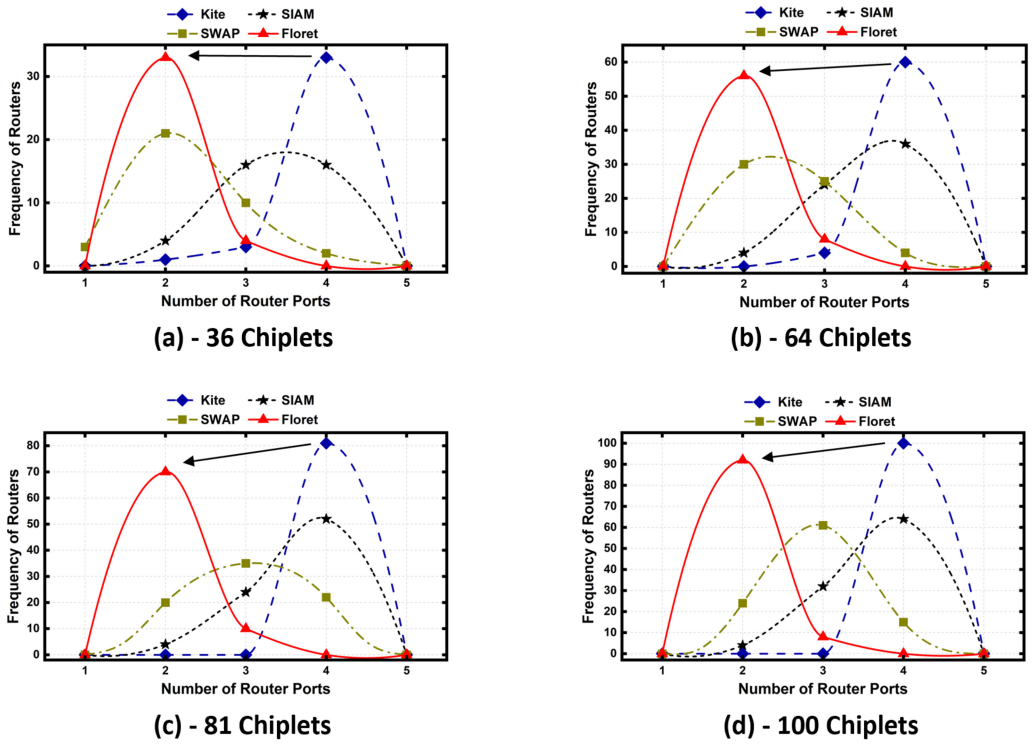

Fig. 4. Variation of router-port configuration for Kite, SIAM, SWAP and SFC for a 2.5D system with (a) 36 chiplets, (b) 64 chiplets, (c) 81 chiplets and (d) 100 chiplets. Peak of the plot is observed to be moving towards the case of Floret which is based on SFC.

routers with fewer ports is increasing in the case of Floret, with the mean router port frequency being between two and three. Similarly, as the system scales to higher number of chiplets, both Kite and SIAM have an average port count of around four, as shown in Figure 4(b), (c), & (d). In case of SWAP, the mean router port frequency lies between two and three with some four port router for larger-system size. Reducing the number of router ports also decreases the total number of links. Figure 5 compares the number of links in each of the considered architecture for all four system sizes. From Figure 4 and Figure 5, it is evident that Floret has smaller routers and fewer associated links compared to all the other architectures. As a result, the total NoI area of Floret is significantly smaller than the other architectures. It should be noted that only reducing the number of links and router port size on their own does not necessarily lead to performance and energy efficiency. To achieve these benefits, it is crucial to consider the length of the links between routers because the communication delay depends on the link lengths. Therefore, the communication delay should be considered while evaluating the NoI architecture. Kite, for example, has mostly two hop links and the routers are inherently bigger. SIAM, being principally a 2D Mesh, has single hop link connections to its neighboring chiplets. However, SIAM ha s bigger routers with higher number of router-ports. SWAP has reduced number of links and smaller router ports, but not all links are necessarily single hop. SWAP also has some longer links like four or five hops. Floret mainly consists of routers with fewer ports and most links being one-hop connections. In the top-level network, we allow the tail of one SFC to communicate with the heads of other SFCs separated by at most three hops. Within each SFC, all the intra-SFC connections are single hops with small router ports. All these factors together improve NoI performance and energy efficiency.

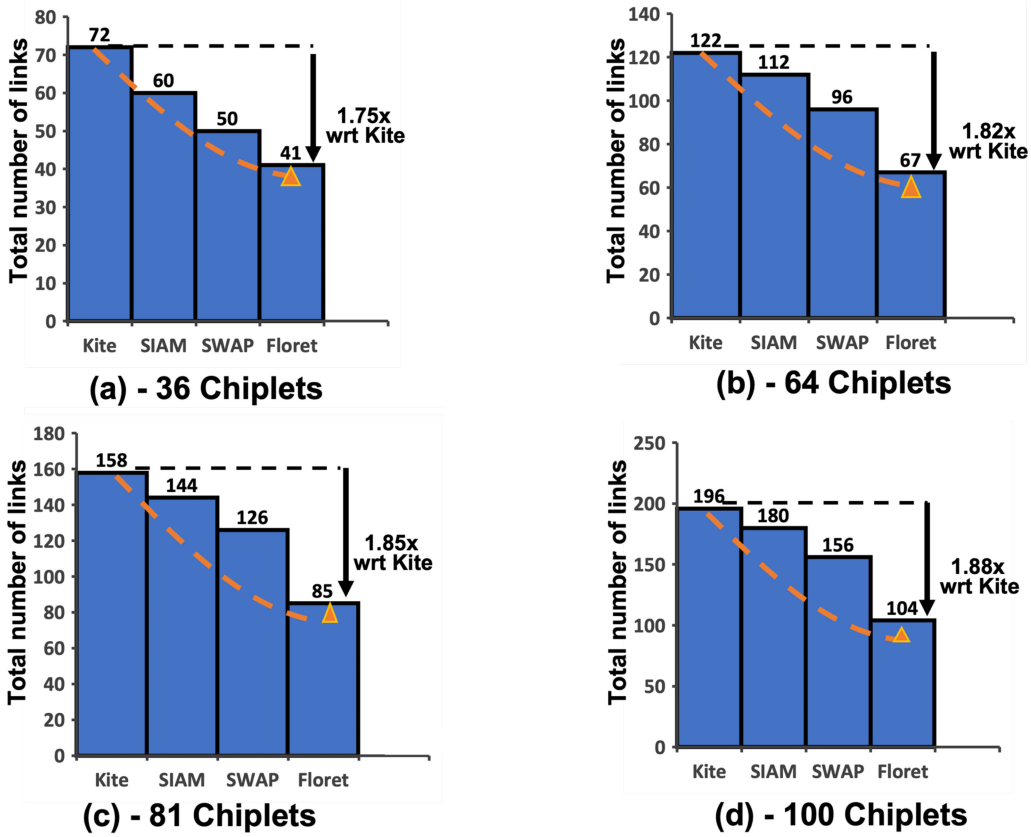

Fig. 5. Variation of number of links for Kite, SIAM, SWAP and Floret for a 2.5D system with (a) 36 chiplets, (b) 64 chiplets, (c) 81 chiplets and (d) 100 chiplets. As the system size increases, the number of links is consistently lower in case of SFC.

In the case of skip connections (such as those found in ResNet or DenseNet), we may have to communicate among non-contiguous chiplets. However, that will still be consecutive single hop paths. Moreover, smaller routers, fewer links, and smaller link lengths reduce the NoI area and hence the fabrication cost, as highlighted in the following subsections.

4.5 NoI Fabrication Cost

One of the main advantages of 2.5D systems over monolithic architectures for large-scale designs is the fabrication cost as the system requirement scales. Therefore, it is crucial to consider the fabrication cost of 2.5D systems along with performance and energy benefits in such a datacenter-scale application. The NoI is the biggest contributor to the overall 2.5-D system area [1]. Hence, reducing the NoI area is important as the computational requirements are expected to grow at scale [1, 2]. This section discusses the relative fabrication cost improvement by Floret with respect to previously proposed architectures. It has been already shown in existing literature that the total NoI area (A_{NoI}) is proportional to the sum of the area of the NoI routers and the links [6]:

$$A_{NoI} \propto \left(\sum_{i=1}^{n} A_{router_i} + \sum_{j=1}^{q} A_{links_j} \right) \tag{2}$$

where A_{router_i} is the area of the i^{th} router and A_{link_j} is the area of the j^{th} link, n and q are the number of NoI routers and links respectively. Each chiplet is connected to an associated NoI router. So, n denotes the total number of chiplets in the system, too. Therefore, increasing the number of router ports (both input and output) as well as NoI links increase the total NoI area. In case of the SFC-based architecture, the number of routers and the corresponding links vary based on the number of SFC λ. As the chiplets in the top-level network have higher connectivity, the router sizes are bigger and hence the NoI area A_{SFC} is defined as:

$$\mathrm{A_{SFC}} = \left(\sum_{i=1}^{2\lambda} \mathrm{A_{inter-SFC}} + \sum_{j=1}^{n-2\lambda} A_{intra-SFC} \right) \tag{3}$$

where $A_{inter-SFC}$ is the area of the top-level network and $A_{intra-SFC}$ is the area of the chiplets within each SFC. Considering total number of chiplets as n and λSFCs on the interposer, the total number of chiplets in top-level network is 2λ and the sum of all chiplets within SFCs is $n - 2\lambda$. The number of links and the router sizes will vary if a particular chiplet exists in the top-level network or not which was discussed in Section 4.3 above. Furthermore, the relative fabrication cost of two NoIs is expressed as [6, 17]:

$$\frac{C_{NoI_1}}{C_{NoI_2}} = e^{-D_0\left(A_{NoI_2} - A_{NoI_1}\right)} \tag{4}$$

where $\mathrm{A_{NoI_1}}$ and $\mathrm{A_{NoI_2}}$ are the NoI area under consideration. Equation (4) assumes that both the system have same number of chiplets, with parameter D_0 representing the wafer defect density. We consider a 2.5D system designed by AMD with 864 mm^2 interposer area and 64 chiplets as the reference in this work [1]. It is evident from that the relative fabrication cost of Floret with respect to any other architectures, like Kite, principally boils down to the difference between the two NoI areas. Since the NoI area increases with increasing number of router ports and NoI links, the corresponding fabrication cost also increases. Considering the router-port configuration and number of links as shown in Figure 4 and Figure 5, Floret reduces fabrication cost by about 80%, 61%, and 49% with respect to Kite, SIAM, and SWAP for a 36-chiplet system. The relative fabrication cost for bigger system sizes reduces more for Floret as the reduction in the number of links is more with the increase in system size (Figure 5). In contrast, the average number of router ports for Floret remains almost unchanged. Moreover, Floret always has more shorter link s than any other architectures considered here.

4.6 NoI Performance and Energy Analysis

This section presents the NoI performance and energy efficiency of Floret compared to the baseline designs (Kite, SIAM, and SWAP). We benchmark the latency and energy consumption of the Floret architecture compared to Kite, SIAM, and SWAP for five different CNN workloads (WL1-WL5 on CIFAR-100; WL6-WL10 on ImageNet) for each system sizes. Each workload has an equivalent probabilistic occurrance of residual(ResNets), dense(DenseNet), and sequential (VGG) CNNs occurring concurrently. This makes sure we cover the entire spectrum of the CNNs without inducing any inherent bias in the experimental evaluation.

Figure 6(a) shows the latency of each NoI for the 36-chiplet system considering CNN workloads WL1 to WL5. Both latency and energy are normalized with respect to the corresponding Floret configuration for all system sizes. We observe that Floret architecture outperforms all the baselines for all the system sizes. As an example, Floret improves the latency by ~27%, ~22%, and ~25% compared to Kite, SIAM, and SWAP architecture for WL1, respectively. On average, Floret performs 23%, 18%, and 19% better than Kite, SIAM and SWAP for 36-chiplet system, respectively. The highest latency improvement of 31% is achieved for WL4 in the 36-chiplet Floret with respect

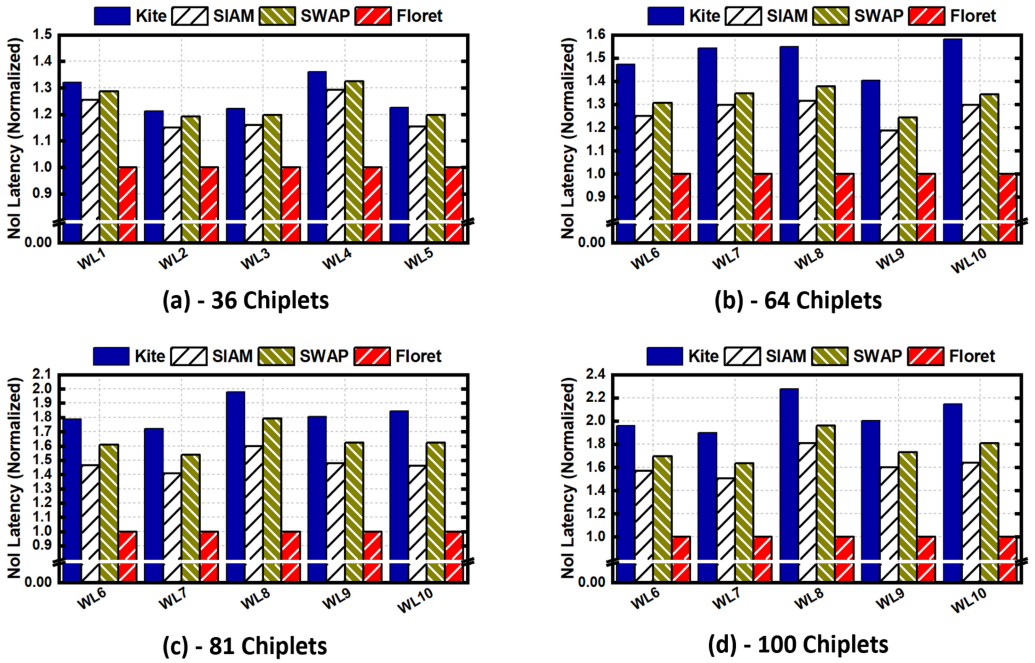

Fig. 6. Comparison of NoI latency for 2.5D system with (a) 36 chiplets, (b) 64 chiplets, (c) 81 chiplets, and (d) 100 chiplet system.

to Kite. For all other CNN workloads, Floret consistently outperforms the existing NoI counterparts in performance. Figures 5(b)-(d) show the latency improvements for Floret compared to the other architectures for 64-, 81- and 100-chiplet systems. The average latency improvements for these system sizes for Floret are: 34%, 21%, and 24% with respect to Kite, SIAM and SWAP for the 64 chiplet system; 45%, 32%, and 38% with respect to Kite, SIAM and SWAP for the 81 chiplet system; 51%, 38%, and 45% with respect to Kite, SIAM and SWAP for the 100 chiplet system.

Floret not only reduces the inference latency of DL workloads but also achieves significant energy consumption savings. For example, Floret reduces the energy consumption by about 22%, 18%, and 20% compared to Kite, SIAM, and SWAP, with a 36 chiplet system for workload WL2 (shown in Table 2(a)). On average, Floret reduces energy consumption by 47%, 20% and 34% for Kite, SIAM and SWAP on 36-chiplet system. Figures 7(b)-(d) show the reductions in energy consumption improvements from Floret compared to the other architectures for 64-, 81-, and 100-chiplet systems. The average energy reductions for these system sizes for Floret are: 51%, 23%, and 35% with respect to Kite, SIAM and SWAP respectively for the 64 chiplet system; 54%, 25%, and 44% with respect to Kite, SIAM and SWAP respectively for the 81 chiplet system; 59%, 29%, and 52% with respect to Kite, SIAM and SWAP respectively for the 100 chiplet system. Both the energy and latency improvements of Floret for biger system sizes demonstrate the scalability of the Floret architecture for datacenter-scale DL application workloads.

We map each CNN layer in Kite, SIAM and SWAP following a greedy mapping algorithm that allocate each incoming CNN layer to the next available chiplet. However, as these three architectures have multi-hop paths between chiplets, it is not possible to get contiguous available chiplets as the number of CNNs increase. Hence, it becomes imperative to map the consecutive neural layers to far-apart chiplets through multi-hop paths. Most importantly, for bigger system sizes

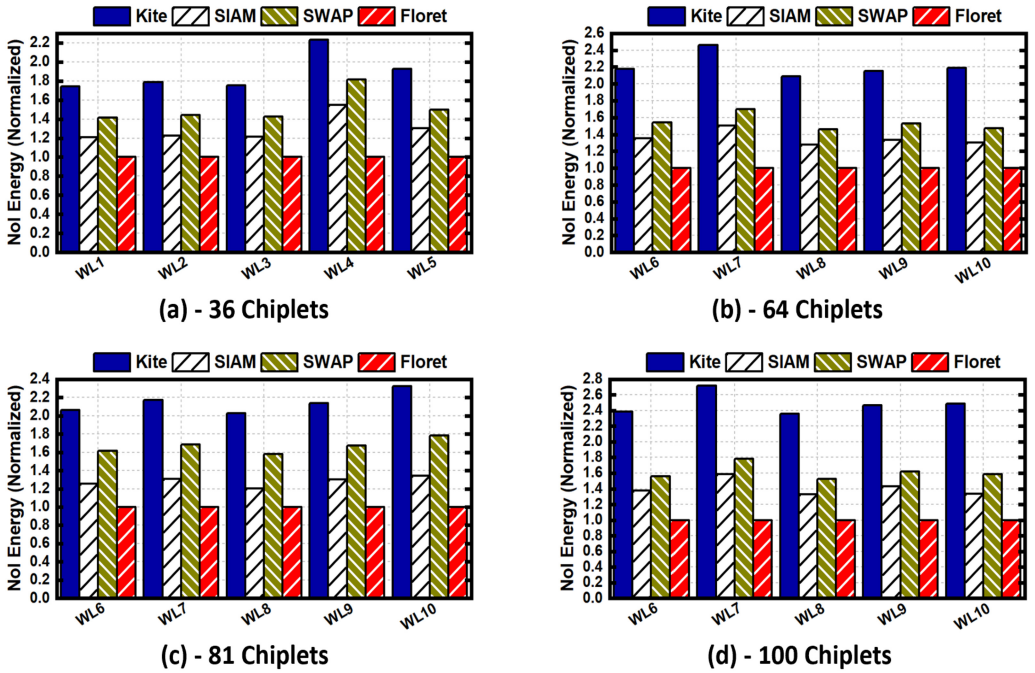

Fig. 7. Comparison of NoI energy for 2.5D system with (a) 36 chiplets, (b) 64 chiplets, (c) 81 chiplets, and (d) 100 chiplet system.

the multi-hop paths increase even more. On contrary, Floret always ensures communicating CNN layers get mapped to contiguous chiplets. Hence, Floret achieves better performance with lower energy consumption compared to other state-of-the-art NoI architectures.

5 CONCLUSION

The emergence of 2.5D chiplet platforms provides a new avenue for compact scale-out implementations of emerging compute- and data-intensive applications. Conventional NoI architectures have a limited computational throughput due to the inherent multi-hop nature of the topology. We presented a novel space-filling curve-based NoI architecture, called Floret, which optimizes task mapping and inter-chiplet data exchange to extract high performance for concurrent CNN inference tasks representing data-center scale scenarios. We demonstrated that the data-flow aware Floret architecture outperforms the state-of-the-art 2.5D manycore architectures with significantly lower energy consumption and fabrication cost. Floret reduces the latency and energy up to 58% and 64%, respectively, compared to state-of-the-art NoI architectures while executing a diverse workload of CNN inference tasks. We also demonstrate that Floret reduces the fabrication costs by up to 82% compared to existing NoI architectures. Optimized top-level network while complimenting the mapping along the space-filling path is the key to Floret's benefits over its counterparts.

REFERENCES

[1] A. Kannan, N. Jerger, and G. Loh. 2015. Enabling interposer-based disintegration of multi-core processors. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*, 2015.

[2] D. Stow et al. 2017. Cost-Effective design of scalable high-performance systems using active and passive interposers. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[3] J. A. Cunningham et al. 1990. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Transaction of Semiconductor Manufacturing* (1990).
[4] International technology roadmap for semiconductors 2.0, 2015 edition, system integration. Report Ch 1, 2015., Semiconductor Industry Association, 2015.
[5] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna. 2020. Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling. In *Proceedings of 57th ACM/IEEE Design Automation Conference (DAC)*.
[6] G. Krishnan et al. 2021. SIAM: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks. *In ACM Transaction of Embedded Computer Systems* 20, 5 (2021).
[7] Y. Shao et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.
[8] Z. Tan, H. Cai, R. Dong, and K. Ma. 2021. NN-Baton: DNN workload orchestration and chiplet granularity exploration for multichip accelerators. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
[9] S. Bergsma, T. Zeyl, A. Senderovich, and J. Beck. 2021. Generating complex, realistic cloud workloads using recurrent neural networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 376–391.
[10] https://www.cloudera.com/content/dam/www/marketing/resources/ebooks/how-to-take-ai-applications-from-concept-to-reality-with-cml-on-aws.pdf.landing.html
[11] A. Verma, M. Korupolu, and J. Wilkes. 2014. Evaluating job packing in warehouse-scale computing. In *Proceedings of the International Conference on Cluster Computing (CLUSTER)*.
[12] D. C. Juan, L. Li, H. K. Peng, D. Marculescu, and C. Faloutsos. Beyond Poisson: Modeling inter-arrival time of requests in a datacenter. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*.
[13] N. Jerger, A. Kannan, Z. Li, and G. Loh. 2014. NoC architectures for silicon interposer systems: Why Pay for more Wires when you Can Get them (from your interposer) for Free?. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 458-470.
[14] B. Zimmer et al. 2020. A 0.32–128 TOPS, scalable Multi-Chip-Module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm. *IEEE Journal of Solid-State Circuits* 55, 4 (2020).
[15] F. Li et al. 2022. GIA: A reusable general interposer architecture for agile chiplet integration. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*.
[16] P. Vivet et al. 2021. IntAct: A 96-Core processor with six chiplets 3D-stacked on an active interposer with distributed interconnects and integrated power management. *IEEE Journal of Solid-State Circuits* 56, 1 (2021).
[17] H. Sharma et al. 2022. SWAP: A server-scale communication-aware chiplet-based manycore PIM accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4145–4156.
[18] S. Mittal. 2019. A survey of ReRAM-based architectures for processing-in-memory and neural networks. *Machine Learning and Knowledge Extraction* 1, 1 (2019).
[19] A. Shafiee et al. 2016. Crossbars., ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 14–26.
[20] L. Song, X. Qian, H. Li, and Y. Chen. 2017. PipeLayer: A pipelined ReRAM-Based accelerator for deep learning. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*.
[21] M. Giordano et al. 2021. CHIMERA: A 0.92 TOPS, 2.2 TOPS/W Edge AI accelerator with 2 MByte On-Chip foundry resistive RAM for efficient training and inference. In *IEEE Symposium on VLSI Circuits*. 1–2.
[22] B. Li et al. 2020. 3D-ReG: A 3D ReRAM-based heterogeneous architecture for training deep neural networks. In *the Journal of Emerging Technology of Computer Systems* 16, 20 (2020).
[23] P. Chi et al. 2016. PRIME: A novel processing- in-memory architecture for neural network computation in ReRAM-Based main memory. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*.
[24] M. Imani, S. Gupta, Y. Kim, and T. Rosing. 2019. Floatpim: In-memory acceleration of deep neural network training with high precision. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*.
[25] T. Ebadollah, S. Pasricha, and M. Nikdast. 2022. ReSiPI: A reconfigurable silicon-photonic 2.5 D chiplet network with PCMs for Energy-Efficient Interposer Communication. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*.
[26] S. V. R. Chittamuru et al. 2018. BiGNoC: Accelerating big data computing with application-specific photonic network-on-chip architectures. *IEEE Transactions on Parallel and Distributed Systems* 29, 11 (2018), 2402–2415.
[27] H. Sagan. 2012. Space-filling curves. *Springer Science & Business Media* (2012).
[28] D. Hilbert. 1891. Uber die stegie abbildung einer linie auf flachenstuck. *Mathematische Annalen* 38 (1891), 459–460.
[29] G. Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. In *IBM*, Ottawa, Canada, 1966.
[30] P. Xu and S. Tirthapura. 2012. A lower bound on proximity preservation by space filling curves. In *Proceedings of the 26th International Parallel and Distributed Processing Symposium*. 1295–1305.

[31] P. Xu, N. Cuong, and S. Tirthapura. 2018. Onion curve: A space filling curve with near-optimal clustering. In 2018). In *Proceedings of the 34th International Conference on Data Engineering (ICDE)*.
[32] D. DeFord and A. Kalyanaraman. 2013. Empirical analysis of space-filling curves for scientific computing applications. In *Proceedings of the 42nd International Conference on Parallel Processing*.
[33] M. Lindenbaum and C. Gotsman. 1996. The metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing* 5, 5 (1996), 794–797.
[34] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. 2001. Analysis of the clustering properties of Hilbert spacefilling curve. *IEEE Transactions on Knowledge and Data Engineering* 13, 1 (2001).
[35] S. Tirthapura, S. Seal, and S. Aluru. 2006. A formal analysis of space filling curves for parallel domain decomposition. In *Proceedings of the International Conference on Parallel Processing (ICPP'06)*.
[36] H. Jagadish. 1990. Linear clustering of objects with multiple attributes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
[37] S. Aluru and F. E. Sevilgen. 1997. Parallel domain decomposition and load balancing using space-filling curves. In *Proceedings of the Fourth International conference on High-Performance Computing*.
[38] E. W. Bethel, D. Camp, D. Donofrio, and M. Howison. 2015. Improving performance of structured-memory, data-intensive applications on multi-core platforms via a space-filling curve memory layout. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS) Workshop*.
[39] M. M. Haque, A. Kalyanaraman, A. Dhingra, N. Abu-Lail, and K. Graybeal. 2009. DNAjig: A new approach for building DNA nanostructures. In *Proceedings of the International Conference on Bioinformatics and Biomedicine*.
[40] S. Sarkar, G. R. Kulkarni, P. P. Pande, and A. Kalyanaraman. 2009. Network-on-chip hardware accelerators for biological sequence alignment. *IEEE Transactions on Computers* 59, 1 (2009), 29–41.
[41] T. Majumder, P. P. Pande, and A. Kalyanaraman. 2013. High-throughput, energy-efficient network-on-chip-based hardware accelerators. In *Proceedings of the Sustainable Computing: Informatics and Systems* 3, 1 (2013), 36–46.
[42] S. Pati et al. 2023. Computation vs. Communication Scaling for Future Transformers on Future Hardware. In *arXiv:2302.02825*, 2023.
[43] G. Karunaratne et al. 2020. In-memory hyperdimensional computing. *Nature Electron* 3 (2020), 327–337.
[44] W. Chen et al. 2019. CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nature Electron* 2 (2019), 420–428.
[45] X. Dong, C. Xu, Y. Xie, and N. Jouppi. 2012. NVSim: A Circuit-Level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (2012).
[46] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal. 2020. In-memory computing in emerging memory technologies for machine learning: An overview. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference (DAC'20)*.
[47] Y. Kim, W. Yang, and O. Mutlu. 2015. RAMULATOR: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters* 15, 1 (2015).
[48] K. K. S. Murty. 1987. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming* 39 (1987), 117–129.
[49] T. K. Hazra and A. Hore. 2016. A comparative study of Travelling Salesman Problem and solution using different algorithm design techniques. In *Proceedings of the 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*.
[50] X. Peng et al. 2019. DNN+NeuroSim: An End-to-End benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *Proceedings of the International Electron Devices Meeting (IEDM)*.
[51] N. Jiang et al. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 86–96.
[52] Intel. 2019. Intel Foveros Interconnect. [Online].
[53] G. Gad et al. 2022. Deep learning-based context-aware video content analysis on IoT devices. *Electronics* 11, 11 (2022).
[54] S. Kumar, L. Bhagat, and J. Jin. 2022. Multi-neural network based tiled 360° video caching with Mobile Edge Computing. *Journal of Network and Computer Applications* (2022).
[55] U. Gupta et al. 2021. Chasing Carbon: The elusive environmental footprint of computing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 854–867

Received 23 March 2023; revised 2 June 2023; accepted 13 July 2023