On Improving the Performance of Intra- and Inter-chiplet Interconnection Networks in Multi-chiplet Systems for Accelerating FHE Encrypted Neural Network Applications

On Improving the Performance of Intra- and Inter-chiplet Interconnection Networks in Multi-chiplet Systems for Accelerating FHE Encrypted Neural Network Applications

Fully Homomorphic Encryption (FHE) is regarded as a promising way to protect data privacy with encrypted computation. Due to high computation overhead, hardware based FHE accelerators were proposed to speed up FHE applications. To support complicated FHE-encrypted neural network applications, multi-chiplet based FHE accelerators were further proposed for scaling up system size, whereas one of the challenges is designing efficient intra- and inter-chiplet interconnection networks to accelerate data transfer. Conventional regular topologies like mesh or Kite either lead to high inter-chiplet transmission latency or excessive power consumption as these topologies assume uniform bandwidth or radix for nodes/links, ignoring the highly irregular distribution of inter-chiplet communication volumes. On the other hand, the problem of generating customized intra- and inter-chiplet interconnection networks has high complexity and previous network-on-chip topology generation works cannot efficiently improve the intra- and inter-chiplet interconnection networks. In this paper, the intra- and inter-chiplet interconnection optimization problem is defined, aiming to minimize the execution time of FHE applications under cost and power constraints. To efficiently solve this problem, we propose a bilevel optimization algorithm, which decomposes the problem into three sub-problems: (1) FHE parameters selection, (2) task-to-core mapping, and (3) intra-/inter-chiplet interconnection network topology generation. These sub-problems are then solved iteratively. Experimental results demonstrate that our proposed method reduces execution time by 51.66%, 43.16%, 39.44%, 43.34%, and 27.70% compared to REED and four multi-chiplet based FHE accelerators with mesh, Kite, Butterfly, and Florets as inter-chiplet networks. Therefore, the proposed method can effectively accelerate FHE applications on large-scale multi-chiplet systems.

Additional Key Words and Phrases: Multi-chiplet Systems; Intra- and Inter-chiplet Interconnection Network optimization

ACM Reference Format:
. 2025. On Improving the Performance of Intra- and Inter-chiplet Interconnection Networks in Multi-chiplet Systems for Accelerating FHE Encrypted Neural Network Applications. *ACM Trans. Embedd. Comput. Syst.* 1, 1 (August 2025), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Fully Homomorphic Encryption (FHE) has been regarded as a promising solution to privacy computing. It enables computations on encrypted data, supporting privacy-preserving machine learning. However, due to ciphertext expansion (up to a million times), FHE operations are 10,000 to 100,000 times slower than plaintext computation [45], with some operations requiring tens of MBs to GBs of auxiliary data [45]. Therefore, hardware based FHE accelerators were proposed to speed up the computation, including F1 [45] , Ark [29] , BTS [31] , REED [3], etc. To further sustain the scaling of big chips, multi-chiplet integration technology enables multiple dies/chiplets to be integrated with interposers to overcome the area wall. Multi-chiplet based FHE accelerators were

Author's Contact Information:

ACM 1558-3465/2025/8-ART
https://doi.org/XXXXXXX.XXXXXXX

proposed [18] where complicated FHE applications (e.g. FHE encrypted neural networks) can be supported.

One of the design challenges is to find an optimal intra- and inter-chiplet interconnection network architecture, which has key impact on the overall system performance [3, 18]. As shown in Fig 1, the topologies of (a) ring-ring and (b) mesh-mesh as the inter-chiplet and intra-chiplet interconnection networks. Although there are widely used topologies like mesh, they lead to sub-optimal transmission efficiency. Fig. 2(a) shows that for a 9-chiplet FHE accelerator running CKKS_ResNet50 with each chiplet having 9 Ark [29] style processors and using mesh/Kite [6] as inter-chiplet interconnection network topologies, Kite [6] reduces execution time by 10.4% compared to mesh. To see the reason, Fig. 3 shows the inter-chiplet communication volumes, showing a high degree of non-uniformity, whereas the ratio of the maximum and minimum volume of inter-chiplet communication demands is as much as 10.4x. Therefore, regular topologies like mesh or ring cannot effectively accelerate communication flows with high volumes as all links are uniform. More recently proposed inter-chiplet interconnection network topologies like Kite [6] increase the number of links and thus improve transmission performance. However, the increase in link number and radix of each router also increases power consumption by 44.3% as shown in Fig. 2(b).

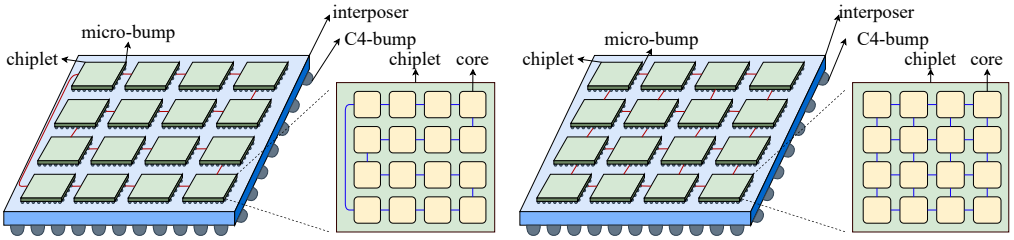

Fig. 1. The topologies of (a) ring-ring and (b) mesh-mesh as the inter-chiplet and intra-chiplet interconnection networks.

Therefore, customized inter-chiplet interconnection network design becomes a must that achieves high power efficiency by considering the irregularity in the traffic distribution of FHE applications. However, the problem of generating customized intra- and inter-chiplet interconnection network has high complexity due to its huge design space. Assume that there are n core/D2D units, there are $O(2^{n^2})$ different topologies, which leads to huge number of design options. Although there are customized network topology generation works in the network-on-chip (NoC) area [12, 39, 44, 63], they cannot be used for generating intra- and inter-chiplet interconnection networks due to the following reasons: (1) They are unaware of the hierarchical nature of intra- and inter-chiplet interconnection networks and the constraints of the D2D interfaces and interposers. (2) Beside topology generation, task-to-core mapping and FHE parameters selection are key to the traffic distribution of inter-chiplet interconnection networks, which are not considered in these works.

To solve this problem, in this paper, an intra- and inter-chiplet interconnection network optimization problem is defined and solved. We start by modeling the latency and application execution time w.r.t. to intra- and inter-chiplet interconnection network parameters and task-to-core mapping via graph models. An optimization problem is formulated to minimize FHE application execution time under cost and power constraints. To solve this problem, we propose an efficient algorithm using a bilevel optimization strategy to (1) generate intra- and inter-chiplet interconnection network, (2) perform task-to-core mapping, and (3) select key FHE parameters. Experimental results confirm that our approach improves performance for FHE applications. Compared to REED [3] and four

multi-chiplet systems with mesh, Kite, Butterfly and Florets [48] as their respective inter-chiplet interconnection network topologies, our proposed scheme reduces the application execution time by 51.66%, 43.16%, 39.44%, 43.34%, and 27.70% respectively. Therefore, the proposed scheme is suitable to improve the performance of large-scale multi-chiplet systems.

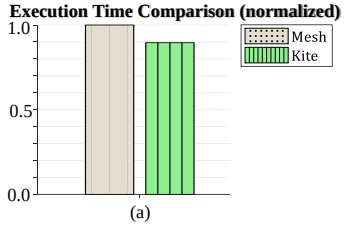

Fig. 2. Execution time and power consumption comparison between multi-chiplet systems, with intra- and inter-chiplet interconnection networks set to be mesh and Kite. The execution time and power consumption of Kite are normalized to those of mesh. The benchmark is CKKS_Resnet50.

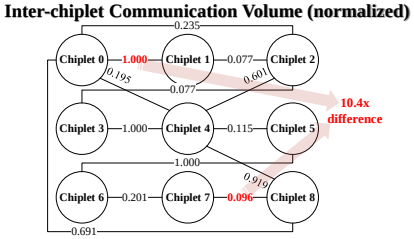

Fig. 3. The inter-chiplet communication volumes. All the inter-chiplet communication volumes are normalized to that between chiplets 0 and 1.

The reminder of the paper is organized as follows. Section II provides a survey of related work. In Section III, the system model for FHE applications running on multi-chiplet systems is formally defined, followed by the problem formulation. Section IV details the bilevel optimization algorithm, solving the sub-problems of (1) FHE parameters selection, (2) task-to-core mapping, and (3) intra-/inter-chiplet interconnection network generation. Section V presents the evaluation of experimental results. Finally, Section VI concludes the paper.

2 Related Work

2.1 Preliminaries on FHE

Fully Homomorphic Encryption (FHE) allows performing arbitrary arithmetic on encrypted plaintext values, via appropriate operations on their ciphertexts. Decrypting the resulting ciphertext yields the same result as if the operations were performed directly on plaintext values [45]. BGV [8], BFV [7], GSW [25], CKKS [15], and TFHE [16] are popular FHE schemes. In our work, we use CKKS as it supports float point arithmetic, making it suitable for machine learning applications.

In CKKS, a message, X, which is a vector of complex numbers, is encoded to a plaintext, $m(X) = \sum_{i=0}^{N} c_i X^i$, represented as a polynomial in a cyclotomic polynomial ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/\left[X^N + 1\right]$ [31]. The coefficients $\{c_i\}$ are integers modulo Q and the polynomial degree is N, which is a power-of-two integer, typically ranging from 2^{10} to 2^{18}. For a given N, up to $\frac{N}{2}$ complex numbers can be packed into a single plaintext, with each element called a slot. After encoding (or packing), element-wise multiplication (mult) and addition between two messages can be performed through polynomial operations. Then, the plaintext $m(X) \in \mathcal{R}_Q$ is encrypted as,

$$ct = (b(X), a(X)) = (a(X) \cdot s(X) + m(X) + e(X), a(X)) \quad (1)$$

where $s(X) \in \mathcal{R}_Q$ is a secret key, $a(X) \in \mathcal{R}_Q$ is a random polynomial, and $e(X)$is a small Gaussian error polynomial required for LWE security guarantee. In CKKS, $ct$is decrypted by computing $m'(X) = ct \cdot (1, -s(X)) = m(X) + e(X)$, which approximates to $m(X)$ with a small error.

2.2 FHE Accelerator Design

In [57], [45], [17], and [31], FHE accelerators NTTFusion, F1, Trinity, and BTS were proposed, with specially designed function units of NTT, modular addition, modular multiplication, and

bootstrapping. Alchemist [38] adopts a novel slot-based data management scheme and utilizes finer-grained low-level arithmetic units to accelerate cross-scheme FHE. HEAP [2] uses the CKKS and the TFHE schemes for the non-bootstrapping and bootstrapping steps to accelerate FHE. UFC [64] unifies cryptographic primitives required for all operations in hybrid FHE workflows, enabling high hardware utilization across multiple FHE schemes. The work in [37] accelerates NTT/INTT by designing a scalable and conflict-free memory mapping algorithm and a flexible, no-stall hardware/software pipeline. PPGNN [58] uses a high-precision arithmetic-logic integrated FHE algorithm tailored for GNN inference. CraterLake [46] is a hardware architecture that efficiently scales to very large ciphertexts, featuring novel functional units to accelerate key kernels. CiFHER [30] and REED [3] adopted MCM and multi-chiplet integration to scale system size. Poseidon [59] is an FPGA-based FHE accelerator, which decomposes fully homomorphic encryption operations into reusable fine-grained operators (e.g., modular operations, NTT), and includes an optimized NTT fusion algorithm and a hardware-friendly automorphism operation to maximize computational parallelism and bandwidth efficiency under limited resources. FAB [1] adopts algorithm-architecture co-optimization through decomposed Key-Switching operations and enhanced modular reduction techniques. The accelerator integrates HBM-driven heterogeneous memory hierarchies with multi-FPGA task parallelism, enabling full-parameter-set bootstrappable homomorphic encryption on FPGA platforms. TensorFHE [19] accelerates FHE by leveraging GPU tensor cores for optimized NTT computations and employing operation-level batching to maximize data parallelism.

2.3 Inter-chiplet Interconnection Network Design

Bharadwaj et al. proposed the Kite topology [6]. It selects either one of the straight or diagonal links as the longest link to reduce network diameter. Feng et al. proposed a method [23] to scale large high-radix inter-chiplet interconnection networks with 2D-mesh-based chiplets. Feng et al. proposed a switch-less Dragonfly topology [21]. Lakhotia et al. proposed PolarFly [32], a diameter-2 topology constructed from Erdős-Rényi polarity graphs with router radix $k = q + 1$, where q is a prime power. However, it restricts radix choices and thus fails to generate flexible and application-specific topologies. Wan et al. proposed a five-level Butterfly fat tree (BFT-like) topology [54] that is suitable for wafer-scale computing systems. Cao et al. proposed an automated design framework called CINT-AD [11] for inter-chiplet interconnection network topology generation. However, it assumes symmetric traffic patterns, which might lead to low performance with high irregular communication patterns of FHE applications.

Unlike prior efforts such as [10, 13, 34, 48, 60, 61] which focus on accelerating plaintext neural network inference with different traffic patterns, our work targets the unique characteristics of homomorphically encrypted applications, including irregular inter-chiplet communication patterns of ciphertext computations.

2.4 Design Space Exploration For Plaintext Neural Networks

Gemini [10] uses a layer-centric encoding method and a co-exploration framework for architecture and mapping, leveraging simulated annealing to optimize inter-chiplet communication. HPPI [34] is a reconfigurable photonic interconnection network with wavelength division multiplexing (WDM) offering four communication patterns. It dynamically selects the optimal pattern via a lightweight neural network for communication-aware customization across convolutional layers. INDM [60] is a hierarchical interconnection network combining a multi-ring intra-chiplet network and a cluster-based inter-chiplet network, along with inter-chiplet communication-aware dataflow mapping to minimize traffic congestion during DNN layer switching. M2M [61] is designed as a fine-graine mapping framework that integrates temporal-spatial task scheduling, communication-aware

mapping, and a QoS management policy for inter-chiplet links, aiming at improving communication efficiency in multi-DNN workloads.

However, these works cannot optimize FHE parameters, task-to-core mapping, and inter-chiplet interconnection networks simultaneously.

3 System Model and Problem Definition

3.1 System Model

In CKKS, a ciphertext is represented by a polynomial pair $\langle c_0, c_1 \rangle$, where N denotes the polynomial degree. The modulus q is decomposed into L primes $q_1, \ldots, q_L$ for leveled operations. The Key-Switching is the most costly operation. The evaluation key (evk) for Key-Switching used in homomorphic multiplication and rotation can be further decomposed into *dnum* slices. The choice of *dnum* impacts the performance, which is considered as a decision variable in this paper.

The CKKS encrypted neural network inference application is modeled by a task graph $G_F(V, S)$, where V is the set of tasks and S is the set of communications between tasks. Each task $v_i \in V$ is a CKKS operator such as homomorphic addition (HAdd), homomorphic multiplication (HMult), and Key-Switching operation etc. Each task has two weights, $\omega(v_i)$ is the execution time of the task. The edge $s_{i,j} = (v_i, v_j) \in S$ is the communication between tasks v_i and v_j, and the weight $\omega(s_{i,j})$ is the traffic volume between them. The second weight $t(v_i, v_j)$ is the communication latency from task v_i to task v_j.

In a multi-chiplet system architecture, multiple chiplets in the system are connected by an inter-chiplet network. Chiplets in the system are denoted as $R_1, R_2 \ldots, R_n$. The number of cores in R_j is denoted as $|R_j|$, and n_R is the total number of chiplets in the system, b_j is the memory capacity of chiplet j. A multi-chiplet system is modeled by a graph $G_C(U, E)$, where U is the set of cores following the architecture in Ark [29] and each core has NTT, INTT, Automorphism, Multiply-Add units, etc., and a global buffer. E is the set of links between cores. Each vertex $u_{i,j} \in U$ is core i in chiplet j, and $e_{i_1,j_1,i_2,j_2} \in E$ is the link between u_{i_1,j_1} and u_{i_2,j_2} whose weight $\omega(e_{i_1,j_1,i_2,j_2})$ is the transmission latency between u_{i_1,j_1} and u_{i_2,j_2} if they are connected. Otherwise, $\omega(e_{i_1,j_1,i_2,j_2})$ is set to be $+\infty$. A binary variable c_{i_1,j_1,i_2,j_2} indicating the connectivity between nodes is defined as follows.

$$c_{i_1,j_1,i_2,j_2} = \begin{cases} 1, & \text{there is a link between } u_{i_1,j_1} \text{ and } u_{i_2,j_2} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The count of die to die (D2D) interfaces in chiplet k is denoted as d_k, and the D2D interfaces in the chiplet k are indexed by $u_{|R_k|+1,k}, \ldots, u_{|R_k|+d_k,k}$ within chiplet k. According to [10], we assume that each chiplet has no more than 4 D2D interfaces for inter-chiplet communication. The D2D interfaces within a chiplet can possibly connect to any core within the same chiplet. This connectivity is indicated by binary variables $c_{i,k,|R_k|+m,k}$, which represent the connectivity of the D2D interface of index m in chiplet k.

A variable $\pi_{i,j,k}$ is defined to denote the mapping of task v_i to core $u_{j,k}$:

$$\pi_{i,j,k} = \begin{cases} 1, & \text{if } v_i \text{ is mapped to } u_{j,k} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The function $M(v_i)$ returns the core $u_{j,k}$ that runs task v_i:

$$\text{if } \forall \pi_{i,j,k} = 1, M(v_i) = u_{j,k} \tag{4}$$

3.2 Latency Model

The latency model estimates the transmission latency between two cores assuming a deterministic routing algorithm. The latency between two cores that run v_i and v_j respectively is composed of: 1) zero load latency $t_{zero}\left(v_i, v_j\right)$, which is related to the distance of the shortest path between the source and destination cores, and 2) queuing latency $t_{queuing}\left(v_i, v_j\right)$, which is modeled by queuing theory.

1) Zero load latency: Given the multi-chiplet system graph $G_C\left(U, E\right)$, and the communication flow $s_{i,j} = \left(v_i, v_j\right)$ that is mapped to cores u_{i_1,j_1} and u_{i_2,j_2}, the zero load latency $t_{zero}\left(v_i, v_j\right)$ consists of two parts: the latency of head packet and the serialization latency:

$$t_{zero}\left(v_i, v_j\right) = t_{head}\left(v_i, v_j\right) + t_{serialization}\left(s_{i,j}\right) \tag{5}$$

The serialization latency $t_{serialization}\left(s_{i,j}\right)$ is related to the packet size (filt number). The latency of the head packet is determined by the length of the shortest path $\Pi\left(M\left(v_i\right), M\left(v_j\right)\right)$ between these two cores $M(v_i)$ and $M(v_j)$ as follows:

$$t_{head}(v_i, v_j) = k_p \cdot l(M(v_i), M(v_j)) = k_p \cdot l(u_{i_1,j_1}, u_{i_2,j_2}) \tag{6}$$

where k_p is the router pipeline stage number, $l\left(M\left(v_i\right), M\left(v_j\right)\right)$ is the length of the shortest path from nodes $M(v_i)$ to $M(v_j)$, that is, from u_{i_1,j_1} to u_{i_2,j_2}.

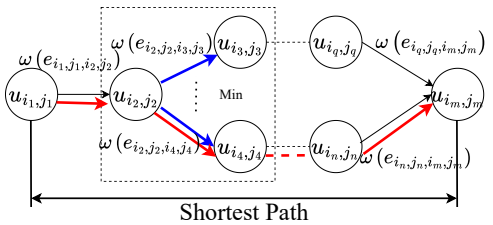

Fig. 4. Shortest path calculation in the latency model. The red line represents the shortest path from source core to destination core. The blue lines are the decisions at u_{i_2,j_2} node.

As shown in Fig. 4, the shortest path can be computed by Bellman equation [50], where the shortest path from core $u_{i_1,j_1} = \widehat{u_0}$ to core $u_{i_2,j_2} = \widehat{u_m}$ is formulated as a sequence of cores $\Pi\left(u_{i_1,j_1}, u_{i_2,j_2}\right) = \Pi\left(\widehat{u_0}, \widehat{u_m}\right) = \left\{\widehat{u_0} = u_{i_1,j_1}, \widehat{u_1}, ..., \widehat{u_m} = u_{i_2,j_2}\right\}$. $\omega\left(\widehat{u_i}, \widehat{u_j}\right)$ is the transmission latency between the $\widehat{u_i}$ and $\widehat{u_j}$.

The shortest path is obtained by recursively using the Bellman equation. In the k th step of this process, the distance from source node $\widehat{u_0}$ to node $\widehat{u_k}$ is recursively updated until the optimum is reached, which is formulated by:

$$l^k(\widehat{u_0}, \widehat{u_k}) = \min_{\forall \widehat{u_k} \in U} \{l^{k-1}(\widehat{u_0}, \widehat{u_{k-1}}) + \omega(\widehat{u_{k-1}}, \widehat{u_k})\} \text{ at node } \widehat{u_k} \tag{7}$$

where U is the set of cores, l^k is the shortest path length in the k th step. The length of the shortest path is the sum of link weights along the path:

$$l\left(u_{i_1,j_1}, u_{i_2,j_2}\right) = l\left(\widehat{u_0}, \widehat{u_m}\right) = \sum_{q=1}^{m} \omega\left(\widehat{u_{q-1}}, \widehat{u_q}\right) \tag{8}$$

To make the performance model differentiable, it is smoothed by the log-exp smoothing method [4]:

$$l(\widehat{u_0}, \widehat{u_m}) = -\ln \sum_{\forall \widehat{u_m} \in U} \exp\{-(l^{m-1}(\widehat{u_0}, \widehat{u_{m-1}}) + \omega\left(\widehat{u_{m-1}}, \widehat{u_m}\right))\} \tag{9}$$

2) Queuing latency: The queuing latency of two cores u_{i_1,j_1} and u_{i_2,j_2} is the summation of the queuing latency of each router in the path $\Pi(u_{i_1,j_1}, u_{i_2,j_2})$, which is obtained by the routing algorithm.

Given the flows $s_{p_0,q_0}, s_{p_1,q_1}, ..., s_{p_m,q_m}$ traversing router $x_{i,j}$, the queuing latency is modeled by the G/G/1 model. In the G/G/1 model, the router $x_{i,j}$ is considered as multiple input channels $IC_{\alpha}^{x_{i,j}} \in \left\{IC_1^{x_{i,j}}, IC_2^{x_{i,j}}, ..., IC_p^{x_{i,j}}\right\}$ competing for a single output channel $OC_{\beta}^{x_{i,j}} \in OC^{x_{i,j}}$. The average queuing latency $\tau_{\alpha\to\beta}^{x_{i,j}}$ in router $x_{i,j}$ is [28]:

$$\tau_{\alpha\to\beta}^{x_{i,j}} = \begin{cases} \dfrac{\rho_{\beta}^{x_{i,j}}\left(C_{A^{x_{i,j}}}^2 + C_{S_{\beta}^{x_{i,j}}}^2\right)}{2\left(\mu_{\beta}^{x_{i,j}} - \lambda_{1\to\beta}^{x_{i,j}}\right)}, \alpha = 1 \\ \dfrac{\lambda_{\beta}^{x_{i,j}}\left(C_{A^{x_{i,j}}}^2 + C_{S_{\beta}^{x_{i,j}}}^2\right)}{2\left(\mu_{\beta}^{x_{i,j}} - \sum_{k=1}^{p-1}\lambda_{k\to\beta}^{x_{i,j}}\right)}, 2 \le \alpha \le p \end{cases} \tag{10}$$

where $\rho_{\beta}^{x_{i,j}}$ is the proportion of time the output channel OC_{β} being occupied by packets, $\lambda_{\alpha\to\beta}^{x_{i,j}}$ and $\mu_{\beta}^{x_{i,j}}$ are the packet arrival rate from input channel $IC_{\alpha}^{x_{i,j}}$ to output channel $OC_{\beta}^{x_{i,j}}$ and the average arrival rate of output channel $OC_{\beta}^{x_{i,j}}$ respectively, $C_{A^{x_{i,j}}}^2$ and $C_{S_{\beta}^{x_{i,j}}}^2$ are the coefficients of variation for the packet arrival rate and service rate that is obtained based on the Allen-Cunneen approximation equation [28].

Based on this formulation, the queuing latency $t_{queuing}$ from tasks v_i to v_j is formulated by the sum of the queuing latency of routers along the path:

$$t_{queuing}\left(v_i, v_j\right) = \sum_{x_{i,j}\in\Pi\left(M(v_i),M(v_j)\right)} \sum_{\alpha=1}^{p}\sum_{\beta=1}^{q} \tau_{\alpha\to\beta}^{x_{i,j}} \tag{11}$$

The communication latency $t(v_i, v_j)$ between tasks v_i and v_j is the sum of zero load latency $t_{zero}\left(v_i, v_j\right)$ and queuing latency $t_{queuing}\left(v_i, v_j\right)$:

$$t\left(v_i, v_j\right) = t_{zero}\left(v_i, v_j\right) + t_{queuing}\left(v_i, v_j\right) \tag{12}$$

3.3 Performance Model

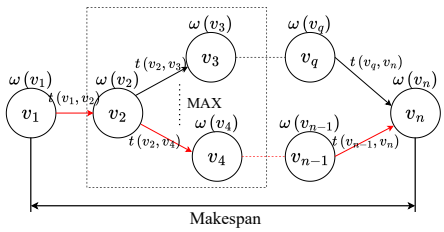

Fig. 5. Makespan computation for a task graph

As shown in Fig. 5, the execution time of the FHE application is estimated by computing the makespan of task graph $G_F(V, S)$, which is obtained by using the Bellman equation. In the k th step of the Bellman equation, the path length from source node v_0 to node v_k is recursively updated to obtain the optimum, which is formulated with:

$$\sigma^k(v_0, v_k) = \max_{\forall v_k\in V}\{\sigma^{k-1}(v_0, v_{k-1}) + t(v_{k-1}, v_k) + \omega(v_k)\} \tag{13}$$

where $\sigma^k(v_0, v_k)$ is the maximum execution time in k th step, V is the set of nodes in the task graph, $\omega\left(v_k\right)$ is the execution time of task v_k, $t(v_{k-1}, v_k)$ is the communication latency from the core running task $k-1$ to the core running task k, defined in Eqn. (12) by the latency model.

When the equation is from the source to the sink tasks in the task graph, the makespan is obtained:

$$\sigma(v_0, v_m) = \sum_{i=1}^{m} \omega(v_0) + \max_{\{v_0,\ldots,v_m\} \in \Pi(v_0,v_m)} (t(v_{i-1}, v_i) + \omega(v_i)) \tag{14}$$

where $\Pi\,(v_0, v_m)$ is the set of paths from task v_0 to v_m, which refers to the source and the sink tasks in the task graph. To make the performance model differentiable, it is smoothed by the log-exp smoothing method [4]:

$$\sigma\,(v_0, v_m) = \ln \sum_{\{v_0, v_1, \ldots, v_k\} \in \Pi(v_0, v_m)} \exp\left(\omega\,(v_0) + \sum_{i=1}^{m} (t\,(v_{i-1}, v_i) + \omega\,(v_i))\right) \tag{15}$$

The coefficients in Eqns. (5) to (15) are computed by the maximum likelihood method [61] as follows. The simulator (detailed in section 5.1) is configured with different settings of intra/inter-chiplet link bandwidth, chiplet core count, D2D interfaces per chiplet, intra/inter-chiplet topology (variables) to obtain t_{zero}, $t_{queuing}$, and σ. The latency and performance models use these data for regression by the maximum likelihood method.

3.4 Cost and Power Models

The multi-chiplet system cost and power consumption are modeled as follows.

Cost Model. The cost of a multi-chiplet system has two components [20]: chiplet silicon cost and packaging cost [52].

The silicon area for all dies/chiplets ($A_{tot} = \sum_{j=1}^{n_R} A(R_j)$) is measured in monetary cost. For each chiplet R_j, $j = 1, 2, \ldots, n_R$, its area is

$$A\left(R_j\right) = \sum_{i=1}^{\left|R_j\right|} A_1\left(u_{i,j}\right) + \sum_{i=\left|R_j\right|+1}^{\left|R_j\right|+d_j} A_2\left(u_{i,j}\right) + \sum_{j=1}^{n} A_3\left(R_j\right) + \sum_{i=1}^{n_j} A_4\left(u_{i,j}\right), \forall 0 < j \leq n_R \tag{16}$$

where $A_1\left(u_{i,j}\right)$, $A_2\left(u_{i,j}\right)$, $A_3\left(R_j\right)$ and $A_4\left(u_{i,j}\right)$ are the areas of each core (i from 1 to $\left|R_j\right|$), D2D interface (i from $\left|R_j\right| + 1$ to $\left|R_j\right| + d_j$), memory unit, and each router respectively, and n_j is the number of routers on each chiplet.

The silicon cost of chiplet R_j is $A(R_j)/Y(R_j) \cdot C_s$, where $Y(R_j)$ is the yield of production per die area and C_s is the cost per silicon area. $Y(R_j) = {Y_{unit}}^{A(R_j)/A_{unit}}$, where Y_{unit} is the yield per unit area.

The packaging cost is $C_p\,(A_{tot})$. The production cost of the system is:

$$Cost = \sum_{j=1}^{n_R} \frac{A(R_j)}{Y_{unit}^{A(R_j)/A_{unit}}} \cdot C_s + C_p\,(A_{tot}) \tag{17}$$

Power Model. The power consumption of a chiplet R_j is:

$$P\left(R_j\right) = \sum_{i=1}^{\left|R_j\right|} P_1\left(u_{i,j}\right) + \sum_{i=\left|R_j\right|+1}^{\left|R_j\right|+d_j} P_2\left(u_{i,j}\right) + \sum_{j=1}^{n} P_3\left(R_j\right) + \sum_{i=1}^{n_j} P_4\left(u_{i,j}\right), \forall 0 < j \leq n_R \tag{18}$$

where $P_1\left(u_{i,j}\right)$, $P_2\left(u_{i,j}\right)$, $P_3\left(R_j\right)$ and $P_4\left(u_{i,j}\right)$ are the powers of each core (i from 1 to $\left|R_j\right|$), D2D interface (i from $\left|R_j\right| + 1$ to $\left|R_j\right| + d_j$), memory unit, and each router respectively, and n_j is the number of routers on each chiplet.

3.5 Problem Formulation

The problem is to minimize the execution time of FHE applications under the cost and power constraints. Mathematically, the problem is formulated as follows:

$$\min \sigma\left(v_0, v_m\right) = \ln \sum_{\{v_0, v_1, \ldots, v_k\} \in \Pi\left(v_0, v_m\right)} \exp\left(\omega\left(v_0\right) + \sum_{i=1}^{m}\left(t\left(v_{i-1}, v_i\right) + \omega\left(v_i\right)\right)\right) \tag{19}$$

where v_i is the node in the task graph, $\Pi\left(v_i, v_j\right)$ is the set of tasks from the source task to the sink task, $\omega\left(v_i\right)$ is the execution time of task v_i, $t(v_i, v_{i-1})$ is the data transmission time between the cores running tasks v_{i-1} and v_i after mapping. The constraints include:

(1) The cost constraint of the system is within a threshold $Cost_0$:

$$Cost \leq Cost_0. \tag{20}$$

(2) The power consumption of the system is within their bounds P_0:

$$\sum_{i=1}^{n_R} P\left(R_i\right) \leq P_0. \tag{21}$$

where $Cost$ and $P(R_i)$ are defined in Eqn. (17) and Eqn. (18) respectively, $Cost_0$ and P_0 are the upper bounds of cost and power respectively.

(3) For the cores in a chiplet, each core cannot connect with cores or D2Ds in another chiplet.

$$c_{i_1, j_1, i_2, j_2} = 0, \text{ if } j_1 \neq j_2 \text{ and } 0 < i_1 < \left|R_{j_1}\right| \tag{22}$$

(4) For the D2Ds in a chiplet, each D2D cannot connect D2Ds in the same chiplet but can only connect D2Ds in other chiplets.

$$c_{i_1, j_1, i_2, j_2} = 0, \text{ if } j_1 = j_2 \text{ and } \left|R_{j_1}\right| < i_1, i_2 < \left|R_{j_1}\right| + d_{j_1} \tag{23}$$

(5) The last constraints indicate one-to-one task mapping:

$$\sum_{k=1}^{n_R} \sum_{j=1}^{|R_k|} \pi_{i,j,k} = 1, \forall 0 < i \leq |V| \tag{24}$$

$$\sum_{i=1}^{|R|} \pi_{i,j,k} = 1, \forall 0 < j < \left|R_k\right|, \forall 0 < k < n_R \tag{25}$$

where n_R is the total number of chiplets, $|R_k|$ is the number of cores in chiplet k, $|V|$ is the number of tasks.

4 Bilevel Optimization Algorithm

4.1 Overview

The bilevel optimization algorithm works as follows. The problem defined in Eqns. (19)-(25) is decomposed into three sub-problems: (1) FHE parameters selection sub-problem (P1), (2) task-to-core mapping sub-problem (P2), and (3) intra- and inter-chiplet interconnection network topology generation (P3).

The process of bilevel optimization is shown in Fig. 6, which is performed iteratively. In each iteration, each sub-problem is solved with the corresponding decision variables improved, while decision variables of other sub-problems are fixed as constants.

The convergence condition is as follows. The iteration terminates when the optimization target is converged, or the number of an iteration reaches the upper bound $n_{\max}$, that is,

$$\sigma_i - \sigma_{i-1} \leq \delta \text{ or } i \leq n_{\max} \tag{26}$$

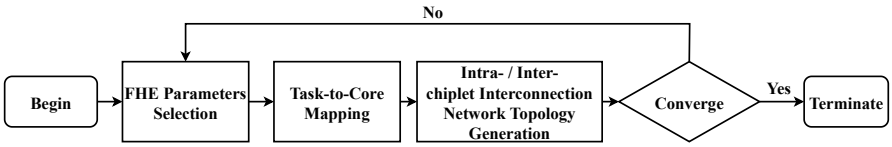

Fig. 6. The process of bilevel optimization.

where σ_i is the application execution time in the i th iteration, δ is the convergence threshold, $n_{\max}$ is the maximum iteration number set by users.

4.2 FHE Parameters Selection (P1)

In this work, FHE parameters are determined through exhaustive searches to balance computational efficiency and noise growth. The parameters include:

- L budget controls the number of supported sequential multiplications, and $L \in [28, 44]$ [14].
- q_i is the prime modulus in the residue number system (RNS), and $q_i \in [2^{40}, 2^{60}]$ [17].
- $dnum$ is the decomposition bits in the Key-Switching operation. It is used to decompose large polynomial coefficients into smaller blocks of bit-length parameters, and $dnum \in \{8, 16, 32, 64\}$.
- n_{slot} is the number of plaintext data units that can be independently encoded and manipulated within a single ring learning with errors (RLWE) ciphertext, and $n_{slot} \in \{2, 8, 32\}$ as in Trinity [17].

4.3 Task-to-Core Mapping (P2)

In task-to-core mapping, the inter-chiplet communication latency is minimized. Existing approaches like [56] rely on complex integer programming, which scales poorly for large-scale FHE applications. In contrast, this work proposes a lightweight and adaptable solution for various topologies.

The input to the task-to-core mapping algorithm is a given network topology. The proposed task-to-core mapping has two steps: 1) The task graph is divided into cuts, which are then allocated to each chiplet, ensuring that the size of each cut is equal to or smaller than the respective chiplet's core count, while minimizing the volumes of communications between chiplets. 2) Within each chiplet, tasks are mapped to cores to reduce communication latency.

(1) Task graph partition. In the first step, the task graph $G_F(V, S)$ is partitioned into n cuts $G_1, G_2, \cdots, G_n$, such that each chiplet R_i can host G_i, i.e., $|R_i| \geq |G_i|$, and the inter-cut communication volume $\sum_{1 \leq p < q \leq n} \sum_{v_i \in G_p} \sum_{v_j \in G_q} \omega(v_i, v_j)$ is minimized.

The algorithm works as follows. The tasks $v_1, v_2, v_3, \cdots, v_{|V|}$ are sorted in descending order by their total communication volume, denoted as $\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_{|V|}$, which are pushed into a queue $\hat{V}$ and assigned to $G_1, G_2, \cdots, G_n$. For each G_i , its size is set to be $|R_i|$, and the following steps are performed.

(1) If G_i is empty, the first element $\hat{v}_i \in \hat{V}$ (the task with the highest communication volume) is popped from $\hat{v}_i$ and assigned to G_i .
(2) If $|G_i| < |R_i|$, task $\hat{v}_j \in \hat{V}$ is selected such that $\underset{\hat{v}_j \text{ is unassigned}}{\arg\max} \left(\sum_{\hat{v}_k \in G_i} \omega(\hat{v}_k, \hat{v}_j) \right)$. That is, the unassigned task which has the highest communication volume with tasks in G_i is selected and assigned to G_i.
(3) If $|G_i| = |R_i|$, the algorithm continues to assign task nodes to G_{i+1} in the same way.

The steps above may not lead to minimal inter-chiplet communication volume. Therefore, an adjustment step is proposed as follows.

In this step, tasks $v_j \in G_p$ and $v_k \in G_q$ are randomly selected and temporarily swapped. In iteration i, the total inter-cut communication volume $C^i = \sum_{1\le p<q\le n} \sum_{v_r\in G_p} \sum_{v_s\in G_q} \omega(v_r, v_s)$ (before swapping) and $C^{i+1} = \sum_{1\le p<q\le n} \sum_{v_r\in G'_p} \sum_{v_s\in G'_q} \omega(v_r, v_s)$ (after swapping) are computed to derive $\Delta^i(v_j, v_k) = C^{i+1} - C^i$, where G'_p and G'_q are cuts after swapping v_j and v_k.

Therefore, in iteration i, if $\Delta^i(v_j, v_k) > 0$, this swapping is actually performed, i.e., v_j is assigned to G_q and v_k is assigned to G_p; otherwise, the swapping is not performed. The iteration terminates when $\max_{r\neq s} \Delta^i(v_r, v_s) \le 0$ or a maximum number of iterations N is reached.

The aforementioned algorithm is demonstrated through an example, as shown in Fig. 7. In the first iteration, v_2 and v_6 are temporarily swapped, after which $\Delta^1(v_2, v_6)=C^2 - C^1 = 21$ is computed. Based on the fact that $\Delta^1(v_2, v_6) > 0$, this swapping is actually performed, which is shown in (b). In the second iteration, v_1 and v_5 are swapped in the same way, as shown in (c). In the third iteration, the algorithm terminates because $\max_{r\neq s} \Delta^3(v_r, v_s) \le 0$.

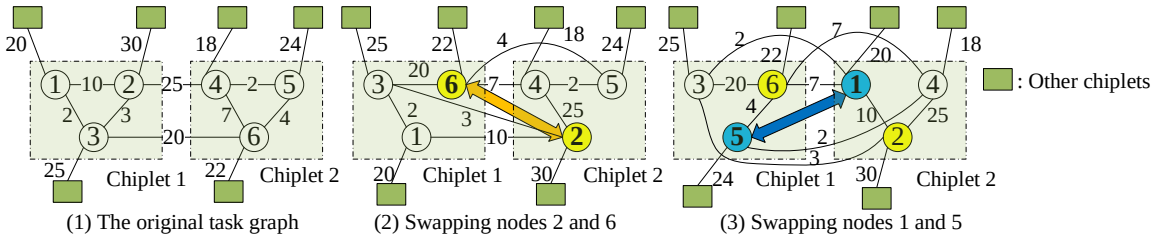

Fig. 7. The procedure of swapping nodes.

(2) Intra-chiplet task mapping. In the last step, a heuristic algorithm is adopted to map tasks in G_j to the cores in chiplet R_j, $j = 1, 2, \cdots, n$ to minimize communication latency.

For each task node $v_i \in G_j$, we define $F(v_i, G_j)$ as the ratio of intra-chiplet to inter-chiplet communication volumes:

$$F(v_i, G_j) = \frac{\sum_{k=1,2,\ldots,n,k\neq j} \sum_{v_r\in G_k} \omega(v_i, v_r)}{\sum_{v_i\in G_j, v_r\neq v_i} \omega(v_i, v_r)}. \tag{27}$$

where $\sum_{k=1,2,\ldots,n,k\neq j} \sum_{v_r\in G_k} \omega(v_i, v_r)$ is the total inter-chiplet communication volume of v_i and $\sum_{v_i\in G_j, v_r\neq v_i} \omega(v_i, v_r)$ is the total intra-chiplet communication volume of v_i.

For each G_j , each task v_i within G_j is sorted in descending order by $F(v_i, G_j)$ and stored in a queue que_i . The following iterations are performed. In iteration k, the head $\hat{v}_p$ and tail elements $\hat{v}_q$ in que_i are popped. For $\hat{v}_p$, if none of its neighbors in G_j is mapped, $\hat{v}_p$ is mapped to the core with maximum degree in R_j . Otherwise, suppose its mapped neighbors in G_j are all in set $N_{p,j}$, it is mapped to a core $u_{x,j}$ such that

$$\arg\min_{u_{x,j}\in R_j} \sum_{\hat{v}_y\in N_{p,j}} \omega(M(\hat{v}_y), u_{x,j}) \cdot \omega(\hat{v}_y, \hat{v}_p) \tag{28}$$

i.e., find a core that minimizes the partial intra-chiplet communication latency in G_j which has high inter-chiplet communication volume.

For $\hat{v}_q$, it is mapped to a core that is closest to a D2D interface $u_{k,j}$ in G_j.

The complexity of the mapping algorithm is $O(\max\{c\times|S|, N\}\times|V|^2)$, where $c = \max_{v_i,v_j\in V, i\neq j} \omega(v_i, v_j)$ and N is the iteration number set by user.

4.4 Intra-/Inter-chiplet Interconnection Network Topology Generation (P3)

After mapping, the communication volumes for each pair of cores/D2Ds is known. In order to obtain an improved intra- and inter-chiplet interconnection network for CKKS-encrypted homomorphic applications, a reinforcement learning (RL) algorithm is proposed since RL was used to solve many network generation problems like [65], [24], [9]. An initial solution is found by an efficient heuristic algorithm. After that, the system continues with the topology generation by reinforcement learning. The algorithm selects a network structure to minimize execution time under the cost and power constraints.

(1) Definition

The topology generation problem is formed as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{Z}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $z_i \in \mathcal{Z}$ is a state, representing i-th intra-/inter-chiplet interconnection network topology, i.e.,

$$z_i = \begin{bmatrix} c^{(i)}_{1,1,1,1} & c^{(i)}_{1,1,2,1} & \cdots & c^{(i)}_{1,1,|R_{n_R}|+d_{n_R},n_R} \\ c^{(i)}_{2,1,1,1} & c^{(i)}_{2,1,2,1} & \cdots & c^{(i)}_{2,1,|R_{n_R}|+d_{n_R},n_R} \\ \cdots & \cdots & \cdots & \cdots \\ c^{(i)}_{|R_{n_R}|+d_{n_R},n_R,1,1} & c^{(i)}_{|R_{n_R}|+d_{n_R},n_R,2,1} & \cdots & c^{(i)}_{|R_{n_R}|+d_{n_R},n_R,|R_{n_R}|+d_{n_R},n_R} \end{bmatrix}, 0 \le i \le N_{TG} \tag{29}$$

where N_{TG} is the number of maximum iterations in the RL algorithm, $a_i \in \mathcal{A}$ is an action, the i-th action corresponds to link insertion and bandwidth enhancement between nodes (cores or D2D interfaces), i.e.,

$$a_i = \begin{cases} c^{(i)}_{j_1,k_1,j_2,k_2} = 1 & , c^{(i-1)}_{j_1,k_1,j_2,k_2} = 0 \\ c^{(i)}_{j_1,k_1,j_2,k_2} = 2\times c^{(i-1)}_{j_1,k_1,j_2,k_2} & , c^{(i-1)}_{j_1,k_1,j_2,k_2} \neq 0 \end{cases} \quad \exists j_1, k_1, j_2, k_2, 1 \le i \le N_{TG} \tag{30}$$

$\mathcal{P}(z_{i+1}|z_i, a_i)$ is the state transition probability, $\mathcal{R}(z_i, a_i, z_{i+1})$ is the reward function, $\gamma \in [0, 1)$ is the discount factor. The policy and value functions are denoted by $\pi_\theta(a_i|z_i)$ and $V_\phi(z_i)$.

(2) Initial Solution

Given $|U|$ cores and n_R (chiplet count), the algorithm first divides $|U|$ cores evenly into n_R sets $R_1, R_2, \cdots, R_{n_R}$. Both the intra- and inter-chiplet networks are initialized to be mesh.

A heuristics algorithm is proposed to generate topology according to the inter-chiplet communication volumes. In the intra-chiplet interconnection network, for each chiplet R_j, $j = 1, 2, \cdots, n_R$, all edges within R_j are sorted in descending order and the top t of which are stored in a queue que_{intra} , where t is a parameter. In each iteration, when que_{intra} is not empty, the first element $< u_{i_1,j}, u_{i_2,j} >$ (edge with the highest communication volume) is popped. A long link is added between $u_{i_1,j}$ and $u_{i_2,j}$ if they are not directly connected and the power and cost constraints are respected. This step is demonstrated through an example, as shown in Fig. 8. The communication volumes between cores after mapping in R_j is shown in Fig. 8 (a) and the original intra-chiplet topology is shown in Fig. 8 (b). The top two edges with the highest communication volume $< u_{1,j}, u_{4,j} >$ and $< u_{2,j}, u_{3,j} >$ are popped into que_{intra}. Links are added between $u_{1,j}$ and $u_{4,j}$, $u_{2,j}$ and $u_{3,j}$ respectively, as shown in Fig. 8 (c).

For the inter-chiplet interconnection network, the inter-chiplet communication volume between each pair of chiplets $< R_{j_1}, R_{j_2} >$ (where $j_1, j_2 = 1, 2, ..., n_R, j_1 \neq j_2$) is denoted as $W(R_{j_1}, R_{j_2}) = \sum_{u_{i_1,j_1}\in R_{j_1}} \sum_{u_{i_2,j_2}\in R_{j_2}} \omega(u_{i_1,j_1}, u_{i_2,j_2})$. The top t pairs of chiplets with the highest communication volumes

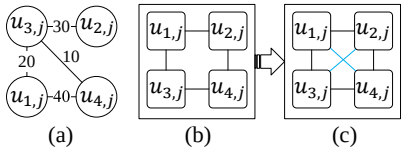

Fig. 8. An example of intra-chiplet interconnection network topology improvement with $t = 2$.

are sorted and stored in a queue que_{inter}. In each iteration, when que_{inter} is not empty, the first element $W(R_{j_1}, R_{j_2})$ in que_{inter} is popped. A serial long link (via interposer [22]) is added between the D2D units of R_{j_1} and R_{j_2} if no direct connection exists between them. For already connected chiplet pairs, the D2D communication bandwidth is doubled provided that power and cost constraints are respected. This methodology is exemplified in Fig. 9, where the initial inter-chiplet interconnection network topology and communication volumes are illustrated in Fig. 9 (a) and (b) respectively. In this example, the top two edges $< R_4, R_5 >$ and $< R_1, R_6 >$ with the highest communication volumes are popped from que_{inter}. It results in the addition of a serial long link between R_1 and R_6, with link bandwidth between R_4 and R_5 being doubled, as demonstrated in Fig. 9 (c).

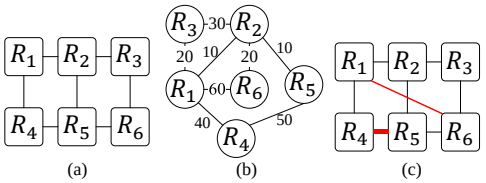

Fig. 9. An example of inter-chiplet interconnection network topology improvement with $t = 2$.

(3) Reward Function

The action a_i corresponds to inserting a link between two nodes (core or D2D) or doubling the bandwidth between them. Each a_i changes the intra-/inter-chiplet interconnection network topology, transforming from the states z_i to z_{i+1}. The reward of inserting edge or doubling bandwidth from state z_i to z_{i+1} is

$$R(z_i, a_i, z_{i+1}) = \sigma(z_{i+1}) - \sigma(z_i) + \lambda_1 \cdot (C(z_{i+1}) - C(z_i)) + \lambda_2 \cdot (P(z_{i+1}) - P(z_i)) \tag{31}$$

where $\sigma(z_i)$, $C(z_i)$ and $P(z_i)$ are execution time, the cost and power consumption of state z_i, respectively. λ_1, λ_2 are the Lagrange multiplier coefficients of the cost and power consumption, respectively, where $\lambda_1, \lambda_2 \in \{0.1, 0.2, ..., 1.0\}$ and selected via exhaustive search. The optimal coefficients are determined by minimizing the execution time under power and cost constraints.

(4) Policy and Value Functions

(4.1) Policy Network

Following the RL algorithm designed for the protein phosphorylation sites network prediction [55], the policy network is designed with the following layers.

- **Problem Encoding:** The intra-/inter-chiplet interconnection network topology can be encoded by an adjacent matrix, denoted as $z_i \in R^{N_d \times N_d}$, where $N_d = \sum_{j=1}^{n_R} (|R_j| + d_j)$.
- **Input Layer:** After encoding, we use ResGCN [43] and self-attention modules for the policy and value functions in the RL algorithm. Fig. 10 (a) is the proposed ResGCN-Attention policy and value networks. The first layer is the input layer, whose input data is the encoded intra-/inter-chiplet interconnection network topology $Z \in \mathbb{R}^{N_d \times N_d}$.
- **ResGCN Layers:** The second layer is ResGCN, as shown in Fig. 10 (b), $H^{(1)} = ReLU(AZW_g^{(1)} + ZW_r^{(1)}) \in \mathbb{R}^{N_d \times 128}$. Graph convolution is used to aggregate neighbor node features. Residual

connections are used to capture local topological relationships. The third layer is also ResGCN for dimension identity mapping, i.e., $H^{(2)} = ReLU(AH^{(1)}W_g^{(2)} + H^{(1)}W_r^{(2)}) \in \mathbb{R}^{N_d \times 512}$. The two-layer ResGCN further integrates high-order neighbor information to enhance the feature expression ability.

- **Global Average Pooling Layer:** The next layer is global average pooling layer, i.e., $H^{(global)} = \frac{1}{N_d}\sum_{i=1}^{N_d} H^{(2)} \in \mathbb{R}^{512}$.
- **Multi-head Self-attention Layer:** The fourth layer is multi-head self-attention layer. In this layer, Query(QH), Key(KH), Value(VH) are computed by $QH = H^{(2)}W_Q$, $KH = H^{(2)}W_H$, $VH = H^{(2)}W_V$, respectively. The multi-head attention mechanism divides the QH, KH, VH into 8 heads, $Q_h, K_h, V_h \in \mathbb{R}^{N_d \times 64}, 1 \le h \le 8$. The attention $Attention_h$ can be computed by $Attention_h = Softmax(\frac{Q_h K_h^\top}{\sqrt{64}})V_h \in \mathbb{R}^{N_d \times 64}, 1 \le h \le 8$. Then $H^{(3)} = Concat(Attention_1, ..., Attention_8)W_O \in \mathbb{R}^{N_d \times 512}$. The multi-head mechanism dynamically adjusts node weights through self-attention, capturing long-range dependencies and enhancing the adaptability of model to various interaction patterns.
- **Global Average Pooling Layer:** The fifth layer is global average pooling layer, i.e., $H^{(4)} = \frac{1}{N_d}\sum_{i=1}^{N_d} H^{(3)} \in \mathbb{R}^{512}$. The $H^{(4)}$ concats the $H^{(global)}$ to be $H^{(5)} \in \mathbb{R}^{1\times 1024}$.
- **Fully Connected Layers:** The sixth layer and seven layer are both fully connnected layers. $H^{(6)} = ReLU(H^{(5)}W_{fc1} + b_{fc1}) \in \mathbb{R}^{256}$. $H^{(7)} = ReLU(H^{(6)}W_{fc2} + b_{fc2}) \in \mathbb{R}^{C_{N_d}^2}$ is the link probability distribution of the edge of each pair of core/D2D.

Since the target of the policy network is to maximize the expected cumulative reward, we optimize the action probability distribution using policy gradients. The loss function of the policy network is

$$\mathcal{L}_{policy} = -\mathbb{E}(\log \pi_\theta(a_i|z_i) \cdot A(z_i, a_i)) \tag{32}$$

where $\log \pi_\theta(a_i|z_i)$ is the probability of selecting action a_i given state z_i, $A(z_i, a_i)$ is the advantage function to measure the advantage of action a_i compared to average as follows.

$$A(z_i, a_i) = SA(z_i, a_i) - V_\phi(z_i) \tag{33}$$

where $SA(z_i, a_i)$ is the state-action function, $V_\phi(z_i)$ is the value of states predicted by the value network.

(4.2) Value Network

The value network is similar to the policy network as shown in Fig 10 (a). The value network does not have self-attention layer following the design in [43]. The high-dimensional features are average pooled directly. After two fully connected layers, the state value estimate function $\in \mathbb{R}^1$ can be obtained. The loss function of value network is

$$\mathcal{L}_{value} = \mathbb{E}((V_\phi(z_i) - V_{target}(z_i))^2) \tag{34}$$

where $V_\phi(z)$, $V_{target}(z)$ are the predicted value of the value network and target value, respectively. $V_{target}(z)$ can be approximated by

$$V_{target}(z_i) = R_i + \gamma \cdot V_\phi(z_{i+1}) \tag{35}$$

where R_i is the instant rewards, γ is the discount factor parameter, z_{i+1} is the next state.

(5) Reinforcement Learning Algorithm

The reinforcement learning (RL) algorithm improves the intra- and inter-chiplet interconnection network topology by iteratively exploring and exploiting the design space. The algorithm works as follows:

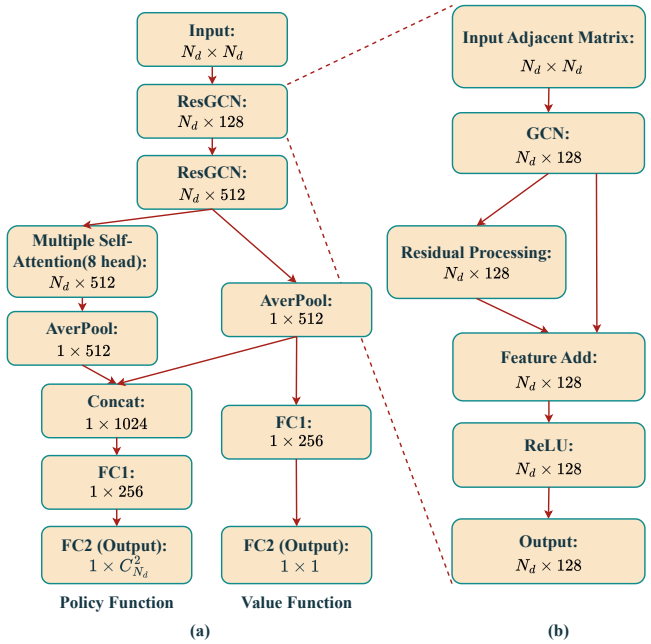

Fig. 10. (a) Policy and Value Networks, (b) ResGCN Module

- **Initialization:** The policy network $\pi_\theta(z_0, a_0)$ and value network $V_\phi(z_0)$ are initialized with parameters θ and ϕ, respectively, where $\theta \in \mathbb{R}^\theta$ and $\phi \in \mathbb{R}^\phi$ define the trainable weights of the ResGCN-based architectures. An experience replay buffer $\mathcal{D}$ is initialized to be $\emptyset$. z_0 is the intra-/inter-chiplet interconnection network topology after step (2) (initial solution).
- **Action Selection:** The agent generates a policy function $\pi_\theta(z_i, a_i)$ through the ResGCN encoder and multi-head attention layers. An action $a_i \in \mathcal{A}$ is sampled from the discrete random distribution $\pi_\theta(z_i, a_i)$.
- **State Transition and Reward Calculation:** The action a_i is executed to modify the topology, yielding a new state z_{i+1} with reward $R_i = R(z_i, a_i, z_{i+1})$ computed following Eqn. (31).
- **Experience Storage:** The transition (z_i, a_i, R_i, z_{i+1}) is stored to $\mathcal{D}$.
- **Policy and Value Networks Update:** A batch of transitions (z_j, a_j, R_j, z_{j+1}) are periodically sampled from $\mathcal{D}$. The policy network is updated to minimize the policy loss (Eqn. (32)) and the value network is updated to minimize the value loss (Eqn. (34)).
- **Termination:** The algorithm terminates when a predefined number of iterations is reached.

5 Experimental Results

5.1 Experimental Setup

Experiments were conducted using the CKKS homomorphic encryption scheme, with the polynomial degree N_{CKKS} being 2^{16} and the scaling factor Δ_{scale} being 2^{64}. Experiments were performed on a multi-chiplet simulator [62] with the cores being those in Ark [29]. The power consumption and areas of router, D2D interface, core, SRAM are from DSENT [49], synthesis from DC following the core design in Ark [29] with TSMC 45nm technology, and Cacti 6.0 [40], respectively. The cost model parameters follow those in [20]. The configurations of the FHE parameters, intra-/inter-chiplet interconnection network, and interposer are tabulated in Table 1.

Table 1. The Configurations of FHE Parameters, Intra-/Inter-chiplet Network, and Interposer

FHE Parameters	
Polynomial Degree N_{CKKS}	2^{16}
Scaling Factor Δ_{scale}	2^{64}
Intra- and Inter-chiplet Network Configuration	
Flit size	256 bits
Intra-chiplet communication latency	Router: 2 cycles Link: 1 cycle
Inter-chiplet communication latency	Router: 2 cycles
Routing algorithm	Routing table based
Buffer depth	4 flits per router
Interposer Configuration	
Thickness	0.1 mm
Heat capacity	1.81×10^6 J/(m$^3\cdot$ K)
Thermal conductivity	35 W/(m$\cdot$ K)
Capacitance density	300 nF/mm^2
Inter-chiplet transmission power consumption	1.17 PJ/bit

Fig. 11 shows an example of an interposer-based 2.5D design and its cross-sectional view. In this design, micro-bumps are created across the surface of chiplets to establish connections with the interposer. The inter-chiplet routing relies on connecting the corresponding mirco-bumps using wires routed across the interposer's metal layers. External signals are routed through the metal layers and then exit the package via C4-bumps, using through-silicon visas (TSVs). The transmission energy consumption between adjacent chiplets is 1.17 PJ/bit [51], and the transmission delay between adjacent chiplets is composed of the following three parts: 1) the processing overhead of packetization and depacketization times, which can be obtained from [53]; 2) the transceivers' transmission delay, which is adopted from [47]; and 3) the interposer wire delay, which is adopted from [27]. In addition, the area and power consumption of D2D interface are based on [22].

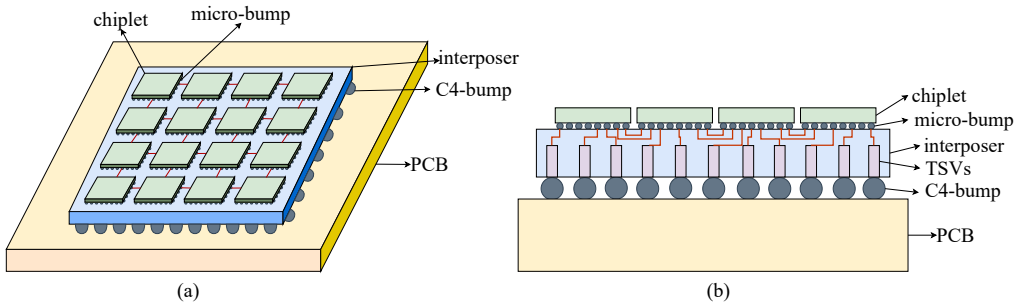

Fig. 11. The packaging of interposer-based 2.5-D system. (a) interposer-based 2.5D IC and (b) Cross-sectional view.

To evaluate the bilevel algorithm proposed in this paper, five previously proposed architectures, REED [3] and multi-chiplet systems with mesh, Kite [6], Butterfly, and Florets [48] as inter-chiplet interconnection networks were selected as the baseline for comparison. REED [3] uses ring as inter-chiplet interconnection network, which is one of the pioneering works for multi-chiplet FHE system design. Kite [6] reduces the average hop count by shortening diagonal long links, and enhances the bi-section bandwidth by adopting the ButterDonut structure. Florets [48] adopts a

layered petal-shaped topology, whereas each subnet (or "petal") corresponds to a chiplet and is connected by the ring topology as intra-/inter-chiplet interconnection network. The experiments include four CKKS-encrypted benchmarks including CKKS_ResNet50, CKKS_MLP, CKKS_CNN, and CKKS_VGG [33].

5.2 Validation of the Performance Model

The performance model defined in Eqn. (15) estimates the application execution time $\sigma(v_0, v_m)$. To validate its accuracy, $\sigma(v_0, v_m)$ derived from Eqn. (15) is compared against the execution time σ_s from simulation. The regression error is defined as:

$$e = \left| \frac{\sigma(v_0, v_m) - \sigma_s}{\sigma_s} \right| \times 100\%.$$

Table 2 compares the average errors of the proposed performance model against other regression models, where all coefficients in the four models are computed via the maximum likelihood method [61], with $\omega(v_i)$ being the execution time of task v_i, $t(v_{i-1}, v_i)$ being the communication latency between the cores running on v_{i-1} and v_i after mapping. As shown in Table 2, one can see that the proposed performance model has the lowest error rate (less than 2%), while the other regression models have errors exceeding 35%. Therefore, the performance model proposed in this work achieves high prediction accuracy.

Table 2. Average Errors of Different Regression Models

Model	Formula	Average Error
Proposed	Eqn. (15)	1.79%
Linear	$\sigma(v_0, v_m) = \alpha \cdot \sum_{i=1}^{m} (t(v_{i-1}, v_i) + \omega(v_i)) + \beta$	37.06%
Exponential	$\sigma(v_0, v_m) = \gamma \cdot \exp\left(\lambda \cdot \sum_{i=1}^{m} (t(v_{i-1}, v_i) + \omega(v_i))\right)$	46.21%
Power-Law	$\sigma(v_0, v_m) = \mu \cdot \left(\sum_{i=1}^{m} (t(v_{i-1}, v_i) + \omega(v_i))\right)^{\nu}$	50.35%

5.3 Ablation Experiment

In this subsection, the contributions of the sub-problems, task-to-core mapping (MA) and topology generation (TG), are evaluated.

As shown in Fig. 12, four configurations are defined:

- **Ring/Mesh**: The multi-chiplet systems have intra- and inter-chiplet interconnection network topologies set to be ring/mesh, using random task-to-core mapping.
- **MA+Ring/MA+Mesh**: The multi-chiplet systems have intra- and inter-chiplet topology set to be ring/mesh, using the *proposed* task-to-core mapping.
- **TG**: The intra- and inter-chiplet interconnection network topology of the multi-chiplet system is generated by the *proposed* topology generation algorithm, using random task-to-core mapping.
- ***Proposed* (MA+TG)**: Both the *proposed* topology generation and task-to-core mapping algorithms are used.

The two MA configurations demonstrate significant latency reduction. As shown in Fig. 12, MA+Ring reduces execution time by 59.77%, 65.13%, 37.78%, and 45.12% for CKKS_CNN, CKKS_MLP, CKKS_ResNet50, and CKKS_VGG respectively compared to random mapping. MA+Mesh reduces

execution time by 24.20%, 31.92%, 12.22%, and 12.79% for CKKS_CNN, CKKS_MLP, CKKS_ResNet50, and CKKS_VGG respectively compared to random mapping. The reason is as follows. Tasks with high communication volume are mapped in close proximity, which decreases inter-chiplet communication volume and latency.

TG also demonstrates significant latency reduction. From Fig. 12, one can see that TG reduces execution time by 35.90%, 46.82%, 65.92%, and 65.76% for CKKS_CNN, CKKS_MLP, CKKS_ResNet50, and CKKS_VGG respectively compared to mesh topology. The proposed topology generation algorithm allocates more links/bandwidth to communications with high volume, thus improving resource utilization.

The *proposed* method on average reduces execution time by 21.9% and 5.1% over MA+Mesh and TG, respectively. Therefore, iteratively improving MA and TG (*proposed*) further reduces execution time than individually performing MA and TG.

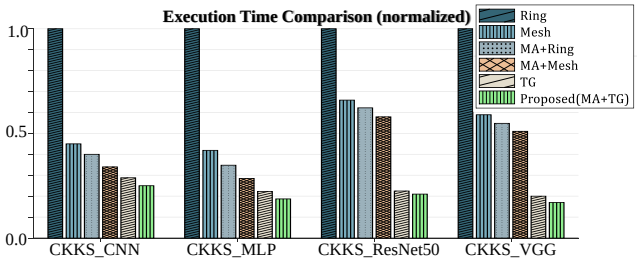

Fig. 12. Comparison of execution times with different configurations. For each FHE application, the execution time of different topologies are normalized to that of ring.

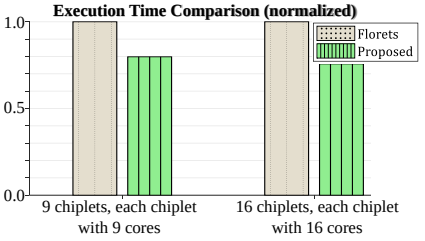

Fig. 14. Comparison of execution times on different multi-chiplet systems with 9 and 16 chiplets, running CKKS_Resnet50 under 200W power budget. The execution time is normalized to that of Florets [48].

5.4 Performance Evaluation

The proposed method is compared with 5 previous works: REED [3], multi-chiplet systems with inter-chiplet network set to be mesh, Kite [6], Butterfly and Florets [48]. All execution times are normalized to that of REED.

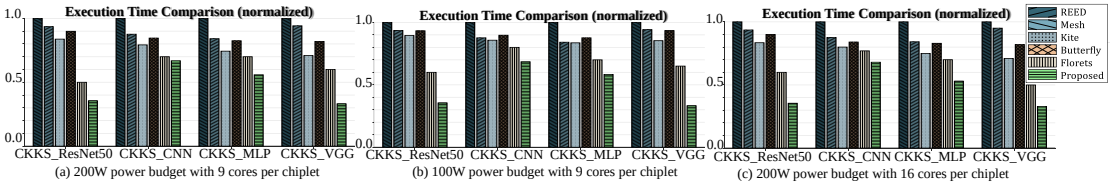

Fig. 13. The execution time comparison with CKKS_ResNet50, CKKS_CNN, CKKS_MLP and CKKS_VGG respectively under various configurations. For each FHE application, the execution times of different methods are normalized to that of REED.

As shown in Fig. 13 (a) with each chiplet having 9 cores and a total of 200W power budget, on average, our proposed method reduces the execution time by 51.66%, 43.16%, 39.44%, 43.34%, and 27.70% compared to REED [3], mesh, Kite [6], Butterfly, and Florets [48] respectively. On average, the power consumption of the proposed method is 0.998×, 0.976×, 0.982×, 0.979×, and 0.984× of REED [3], mesh, Kite [6], Butterfly, and Florets [48], respectively.

As shown in Fig. 13 (b) with each chiplet having 9 cores and a total of 100W power budget, on average, our proposed method reduces the execution time by 50.61%, 43.91%, 43.78%, 45.87%, and 24.62% compared to REED [3], mesh, Kite [6], Butterfly, and Florets [48], respectively.

As shown in Fig. 13 (c) with each chiplet having 16 cores and a total of 200W power budget, on average, our proposed method reduces the execution time by 52.19%, 43.76%, 40.39%, 43.88%, and 30.20% compared to REED [3], mesh, Kite [6], Butterfly, and Florets [48], respectively.

In addition, we have implemented a new accelerator whose PEs resemble those in REED. The inter- and intra-chiplet interconnection network in this accelerator is generated using our proposed algorithm. The accelerator has been synthesized using Synopsys Design Compiler with the SMIC 28nm HKC Plus process standard cell library (HS C35 P140 RVT, version 0.1b). The D2D PHY follows that described in [22].

For comparison, with 9 PEs per chiplet and in total 9 chiplets for running the CKKS_Resnet50 benchmark, the new accelerator achieves an execution time of 0.36× of that of REED, while the power consumption and area are 0.99× and 0.91× those of REED, respectively. The Verilog code for the PE is implemented based on our understanding of REED, with on-chip network routers sourced from the OpenPiton project [5]. Our accelerator has also been validated using a 24-FPGA system (each FPGA is Xilinx vu13p), with each FPGA communicating via QSFP 28 links, and the D2D interfaces mapped to those of QSFP 28 PHY in FPGAs.

The reason is that our proposed method can optimize both the intra- and inter-chiplet network topology and task-to-core mapping, therefore improving the overall performance.

5.5 Scalability Analysis

The proposed method is evaluated on different multi-chiplet systems with 9 and 16 chiplets to validate its scalability. As shown in Fig. 14, the proposed method reduces execution time by 20.2% and 24.3% for CKKS_ResNet50 compared to Florets [48], on a 9-chiplet and a 16-chiplet systems respectively. Therefore, the proposed method can effectively accelerate FHE applications in multi-chiplet systems of varying system sizes.

5.6 Applicability Beyond FHE

To validate the usability of the proposed method for applications other than FHE, we have conducted three sets of experiments (DLRM [42], BEVFusion [36], and DeepSeek (one decoding block in D phase) [35] in plaintext for comparison. These benchmarks are parallelized according to [41], [36] and [35]. The PEs are adopted from [26], and the area and power models for D2D and routers are based on [49]. As shown in Fig. 15, with each chiplet having 9 cores and a total power budget of 100W, our proposed method reduces execution time by an average of 46.50% and 46.20% compared to ring and mesh inter-chiplet topologies (with corresponding intra-chiplet topologies).

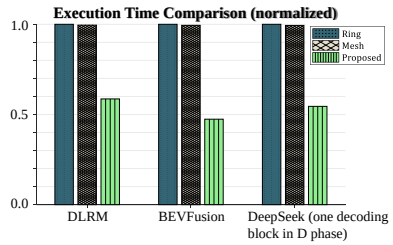

Fig. 15. The execution time comparison of different inter-chiplet network topologies.

6 Conclusion

In this paper, an intra- and inter-chiplet interconnection network optimization problem was proposed under the power and cost constraints, which was solved by a bilevel optimization algorithm. In essence, three sub-problems were solved iteratively: (1) FHE parameters selection, (2) task-to-core

mapping, and (3) intra-/inter-chiplet interconnection network topology generation. Experimental results demonstrate that our proposed method reduces execution time by 51.66%, 43.16%, 39.44%, 43.34%, and 27.70% compared to REED and four multi-chiplet systems with mesh, Kite, Butterfly, and Florets as inter-chiplet networks. While the proposed method is tailored for FHE workloads, it can be adapted to other applications by substituting task graphs and adjusting parameters. This flexibility is enabled by the bilevel optimization framework, which decouples application-specific constraints from topology generation. Therefore, the proposed method can effectively accelerate FHE applications on large-scale multi-chiplet systems.

References

[1] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Tugce Yazicigil, Anantha P. Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. FAB: An FPGA-based Accelerator for Bootstrappable Fully Homomorphic Encryption. In *IEEE International Symposium on High-Performance Computer Architecture*. 882–895.

[2] Rashmi S. Agrawal, Anantha P. Chandrakasan, and Ajay Joshi. 2024. HEAP: A Fully Homomorphic Encryption Accelerator with Parallelized Bootstrapping. In *ACM/IEEEInternational Symposium on Computer Architecture*. 756–769.

[3] Aikata, Ahmet Can Mert, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. 2023. REED: Chiplet-Based Scalable Hardware Accelerator for Fully Homomorphic Encryption. *International Association for Cryptologic Research Cryptology ePrint Archive* (2023), 1190.

[4] Arnold O. Allen. 1990. *Probability, Statistics and Queueing Theory - with Computer Science Applications (2. ed.)*.

[5] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, et al. 2016. OpenPiton: An Open Source Manycore Research Framework. *ACM SIGPLAN Notices* 51, 4 (2016), 217–232.

[6] Srikant Bharadwaj, Jieming Yin, Bradford M. Beckmann, and Tushar Krishna. 2020. Kite: A Family of Heterogeneous Interposer Topologies Enabled via Accurate Interconnect Modeling. In *ACM/IEEE Design Automation Conference*. 1–6.

[7] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Annual Cryptology Conference*. 868–886.

[8] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory* 6, 3 (2014), 1–36.

[9] Van-Phuc Bui, Shashi Raj Pandey, Pedro Maia de Sant Ana, and Petar Popovski. 2024. Value-Based Reinforcement Learning for Digital Twins in Cloud Computing. In *IEEE International Conference on Communications*. 1413–1418.

[10] Jingwei Cai, Zuotong Wu, Sen Peng, Yuchen Wei, Zhanhong Tan, Guiming Shi, Mingyu Gao, and Kaisheng Ma. 2024. Gemini: Mapping and Architecture Co-exploration for Large-scale DNN Chiplet Accelerators. In *IEEE International Symposium on High-Performance Computer Architecture*. 156–171.

[11] Zhipeng Cao, Qinrang Liu, Zhiquan Wan, Wenbo Zhang, Ke Song, and Wenbin Liu. 2025. Enhancing Interconnection Network Topology for Chiplet-based Systems: An Automated Design Framework. *Future Generation Computer Systems* 163 (2025), 107547.

[12] Karam S. Chatha, Krishnan Srinivasan, and Goran Konjevod. 2008. Automated Techniques for Synthesis of Application-Specific Network-on-Chip Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 8 (2008), 1425–1438.

[13] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *ACM/IEEE International Symposium on Computer Architecture*. 367–379.

[14] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography*, Vol. 11349. 347–368.

[15] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Applications of Cryptology and Information Security*. 409–437.

[16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption over the Torus. *Journal of Cryptology* 33, 1 (2020), 34–91.

[17] Xianglong Deng, Shengyu Fan, Zhicheng Hu, Zhuoyu Tian, Zihao Yang, Jiangrui Yu, Dingyuan Cao, Dan Meng, Rui Hou, Meng Li, Qian Lou, and Mingzhe Zhang. 2024. Trinity: A General Purpose FHE Accelerator. In *IEEE/ACM International Symposium on Microarchitecture*. 338–351.

[18] Yibo Du, Ying Wang, Bing Li, Fuping Li, Shengwen Liang, Huawei Li, Xiaowei Li, and Yinhe Han. 2024. Chiplever: Towards Effortless Extension of Chiplet-based System for FHE. In *ACM/IEEE Design Automation Conference*. 243:1–243:6.

[19] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2023. TensorFHE: Achieving Practical Computation on Encrypted Data Using GPGPU. In *IEEE International Symposium on High-Performance Computer Architecture*. 922–934.

[20] Yinxiao Feng and Kaisheng Ma. 2022. Chiplet actuary: A Quantitative Cost Model and Multi-chiplet Architecture Exploration. In *ACM/IEEE Design Automation Conference*. 121–126.

[21] Yinxiao Feng and Kaisheng Ma. 2024. Switch-Less Dragonfly on Wafers: A Scalable Interconnection Architecture based on Wafer-Scale Integration. In *International Conference for High Performance Computing, Networking, Storage, and Analysis*. 96.

[22] Yinxiao Feng, Dong Xiang, and Kaisheng Ma. 2023. Heterogeneous Die-to-Die Interfaces: Enabling More Flexible Chiplet Interconnection Systems. In *International Symposium on Microarchitecture*. 930–943.

[23] Yinxiao Feng, Dong Xiang, and Kaisheng Ma. 2023. A Scalable Methodology for Designing Efficient Interconnection Network of Chiplets. In *IEEE International Symposium on High-Performance Computer Architecture*. 1059–1071.

[24] Daniele Gammelli, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C. Pereira, and Marco Pavone. 2021. Graph Neural Network Reinforcement Learning for Autonomous Mobility-on-Demand Systems. In *IEEE Conference on Decision and Control*. 2996–3003.

[25] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-simpler, Asymptotically-faster, Attribute-based. In *Cryptology Conference*. 75–92.

[26] Tom Glint, Mithil Pechimuthu, and Joycee Mekie. 2024. DeepFrack: A Comprehensive Framework for Layer Fusion, Face Tiling, and Efficient Mapping in DNN Hardware Accelerators. In *Design, Automation & Test in Europe Conference & Exhibition*. 1–6.

[27] M. D. Arafat Kabir and Yarui Peng. 2020. Chiplet-Package Co-Design For 2.5D Systems Using Standard ASIC CAD Tools. In *Asia and South Pacific Design Automation Conference*. 351–356.

[28] Abbas Eslami Kiasari, Zhonghai Lu, and Axel Jantsch. 2013. An Analytical Latency Model for Networks-on-Chip. *IEEE Transaction on Very Large Scale Integration Systems* 21, 1 (2013), 113–123.

[29] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn. 2022. ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse. In *IEEE/ACM International Symposium on Microarchitecture*. 1237–1254.

[30] Sangpyo Kim, Jongmin Kim, Jaeyoung Choi, and Jung Ho Ahn. 2024. CiFHER: A Chiplet-Based FHE Accelerator with a Resizable Structure. In *International Symposium on Secure and Private Execution Environment Design*. 119–130.

[31] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. 2022. BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption. In *International Symposium on Computer Architecture*. 711–725.

[32] Kartik Lakhotia, Maciej Besta, Laura Monroe, Kelly Isham, Patrick Iff, Torsten Hoefler, and Fabrizio Petrini. 2022. PolarFly: A Cost-Effective and Flexible Low-Diameter Topology. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 12:1–12:15.

[33] Hyunhoon Lee and Youngjoo Lee. 2023. Optimizations of Privacy-Preserving DNN for Low-Latency Inference on Encrypted Data. *IEEE Access* 11 (2023), 104775–104788.

[34] Guanglong Li and Yaoyao Ye. 2024. HPPI: A High-Performance Photonic Interconnect Design for Chiplet-Based DNN Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 3 (2024), 812–825.

[35] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 Technical Report. *ArXiv Preprint ArXiv:2412.19437* (2024).

[36] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela L Rus, and Song Han. 2023. Bevfusion: Multi-task Multi-sensor Fusion with Unified Bird's-eye View Representation. In *International Conference on Robotics and Automation*. 2774–2781.

[37] Zhaojun Lu, Weizong Yu, Peng Xu, Wei Wang, Jiliang Zhang, and Dengguo Feng. 2024. An NTT/INTT Accelerator with Ultra-High Throughput and Area Efficiency for FHE. In *ACM/IEEE Design Automation Conference*. 158:1–158:6.

[38] Jianan Mu, Husheng Han, Shangyi Shi, Jing Ye, Zizhen Liu, Shengwen Liang, Meng Li, Mingzhe Zhang, Song Bian, Xing Hu, Huawei Li, and Xiaowei Li. 2024. Alchemist: A Unified Accelerator Architecture for Cross-Scheme Fully Homomorphic Encryption. In *ACM/IEEE Design Automation Conference*. 26:1–26:6.

[39] Srinivasan Murali and Giovanni De Micheli. 2004. SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs. In *Design Automation Conference*. 914–919.

[40] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A Tool to Model Large Caches. *HP laboratories* 27 (2009), 28.

[41] Krishnakumar Nair, Avinash-Chandra Pandey, Siddappa Karabannavar, Meena Arunachalam, John Kalamatianos, Varun Agrawal, Saurabh Gupta, Ashish Sirasao, Elliott Delaye, Steve Reinhardt, et al. 2024. Parallelization Strategies for DLRM Embedding Bag Operator on AMD CPUs. *IEEE Micro* (2024).

[42] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *ArXiv Preprint ArXiv:1906.00091* (2019).

[43] Yulong Pei, Tianjin Huang, Werner van Ipenburg, and Mykola Pechenizkiy. 2022. ResGCN: Attention-based Deep Residual Modeling for Anomaly Detection on Attributed Networks. *Machine Learning* 111, 2 (2022), 519–541.

[44] Md Farhadur Reza, Tung Thanh Le, Bappaditya Dey, Magdy A. Bayoumi, and Dan Zhao. 2018. Neuro-NoC: Energy Optimization in Heterogeneous Many-Core NoC using Neural Networks in Dark Silicon Era. In *IEEE International Symposium on Circuits and Systems*. 1–5.

[45] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald G. Dreslinski, Christopher Peikert, and Daniel Sánchez. 2021. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In *IEEE/ACM International Symposium on Microarchitecture*. 238–252.

[46] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sánchez. 2022. CraterLake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data. In *International Symposium on Computer Architecture*. 173–187.

[47] Debendra Das Sharma, Gerald Pasdast, Zhiguo Qian, and Kemal Aygun. 2022. Universal Chiplet Interconnect Express (UCIe): An Open Industry Standard for Innovations with Chiplets at Package Level. *IEEE Transactions on Components, Packaging and Manufacturing Technology* 12, 9 (2022), 1423–1431.

[48] Harsh Sharma, Lukas Pfromm, Rasit Onur Topaloglu, Janardhan Rao Doppa, Ümit Y. Ogras, Ananth Kalyanaraman, and Partha Pratim Pande. 2023. Florets for Chiplets: Data Flow-aware High-Performance and Energy-efficient Network-on-Interposer for CNN Inference Tasks. *ACM Transactions on Embedded Computing Systems* 22, 5s (2023), 132:1–132:21.

[49] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason E. Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *IEEE/ACM International Symposium on Networks-on-Chip*. 201–210.

[50] Ebadollah Taheri, Sudeep Pasricha, and Mahdi Nikdast. 2022. DeFT: A Deadlock-Free and Fault-Tolerant Routing Algorithm for 2.5D Chiplet Networks. In *Design, Automation & Test in Europe Conference & Exhibition*. 1047–1052.

[51] Zhanhong Tan, Hongyu Cai, Runpei Dong, and Kaisheng Ma. 2021. NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators. In *ACM/IEEE International Symposium on Computer Architecture*. 1013–1026.

[52] Tianqi Tang and Yuan Xie. 2022. Cost-Aware Exploration for Chiplet-Based Architecture with Advanced Packaging Technologies. *CoRR* abs/2206.07308 (2022).

[53] Andreas Traber, Florian Zaruba, Sven Stucki, Antonio Pullini, Germain Haugou, Eric Flamand, Frank K Gurkaynak, and Luca Benini. 2016. PULPino: A Small Single-core RISC-V SoC. In *RISCV Workshop*. 15.

[54] Zhiquan Wan, Zhipeng Cao, Shunbin Li, Peijie Li, Qingwen Deng, Weihao Wang, Kun Zhang, Guandong Liu, Ruyun Zhang, and Qinrang Liu. 2025. Architectural Exploration for Waferscale Switching System. *IEEE Transactions on Very Large Scale Integration Systems* 33, 2 (2025), 512–524.

[55] Minghui Wang, Jihua Jia, Fei Xu, Hongyan Zhou, Yushuang Liu, and Bin Yu. 2024. Res-GCN: Identification of Protein Phosphorylation Sites Using Graph Convolutional Network and Residual Network. *Computational Biology and Chemistry* 112 (2024), 108183.

[56] Xiaohang Wang, Yifan Wang, Yingtao Jiang, Amit Kumar Singh, and Mei Yang. 2025. On Task Mapping in Multi-chiplet Based Many-Core Systems to Optimize Inter- and Intra-chiplet Communications. *IEEE Trans. Comput.* 74, 2 (2025), 510–525.

[57] Zhiwei Wang, Peinan Li, Rui Hou, and Dan Meng. 2023. NTTFusion: Efficient Number Theoretic Transform Acceleration on GPUs. In *IEEE International Conference on Computer Design*. 357–365.

[58] Yuntao Wei, Xueyan Wang, Song Bian, Yicheng Huang, Weisheng Zhao, and Yier Jin. 2024. PPGNN: Fast and Accurate Privacy-Preserving Graph Neural Network Inference via Parallel and Pipelined Arithmetic-and-Logic FHE Accelerator. In *ACM/IEEE Design Automation Conference*. 273:1–273:6.

[59] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. 2023. Poseidon: Practical Homomorphic Encryption Accelerator. In *IEEE International Symposium on High-Performance Computer Architecture*. 870–881.

[60] Jinming Zhang, Xi Fan, Yaoyao Ye, Xuyan Wang, Guojie Xiong, Xianglun Leng, Ningyi Xu, Yong Lian, and Guanghui He. 2024. INDM: Chiplet-Based Interconnect Network and Dataflow Mapping for DNN Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 4 (2024), 1107–1120.

[61] Jinming Zhang, Xuyan Wang, Yaoyao Ye, Dongxu Lyu, Guojie Xiong, Ningyi Xu, Yong Lian, and Guanghui He. 2024. M2M: A Fine-Grained Mapping Framework to Accelerate Multiple DNNs on a Multi-Chiplet Architecture. *IEEE Transactions on Very Large Scale Integration Systems* 32, 10 (2024), 1864–1877.

[62] Haocong Zhi, Xianuo Xu, Weijian Han, Zhilin Gao, Xiaohang Wang, Maurizio Palesi, Amit Kumar Singh, and Letian Huang. 2021. A Methodology for Simulating Multi-chiplet Systems Using Open-source Simulators. In *ACM International Conference on Nanoscale Computing and Communication*. 1–6.

[63] Wei Zhong, Bei Yu, Song Chen, Takeshi Yoshimura, Sheqin Dong, and Satoshi Goto. 2011. Application-specific Network-on-Chip Synthesis: Cluster Generation and Network Component Insertion. In *International Symposium on Quality Electronic Design*. 144–149.

[64] Minxuan Zhou, Yujin Nam, Xuan Wang, Youhak Lee, Chris Wilkerson, Raghavan Kumar, Sachin Taneja, Sanu Mathew, Rosario Cammarota, and Tajana Rosing. 2024. UFC: A Unified Accelerator for Fully Homomorphic Encryption. In *IEEE/ACM International Symposium on Microarchitecture*. 352–365.

[65] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. 2020. Causal Discovery with Reinforcement Learning. In *International Conference on Learning Representations*.