M2M: A Fine-Grained Mapping Framework to Accelerate Multiple DNNs on a Multi-Chiplet Architecture

Jinming Zhang, Xuyan Wang, Yaoyao Ye, *Member, IEEE*, Dongxu Lyu, Guojie Xiong, Ningyi Xu, Yong Lian, *Fellow, IEEE*, and Guanghui He, *Member, IEEE*

***Abstract*— With the advancement of artificial intelligence, the collaboration of multiple deep neural networks (DNNs) has been crucial to existing embedded systems and cloud systems, especially for automatic driving applications as well as augmented and virtual reality (AR/VR) applications. To trade off between cost and performance, chiplet-based DNN accelerators have emerged as a promising solution for accelerating DNN workloads. However, most existing mapping methods for multiple DNNs target for the monolithic chip, which fail to solve the problems faced by the emerging multi-chiplet architecture, such as the problems of distributed memory access, complex heterogeneous interconnect network, and the scaling-up of computing resources. In this work, we propose M2M, a fine-grained mapping framework for accelerating multiple DNNs on a multi-chiplet architecture. It includes a temporal and spatial task scheduling for reconfigurable dataflow accelerators and a communication-aware task mapping in a heterogeneous interconnect network. To enhance communication efficiency and reduce the overall latency, we further propose a fine-tuned quality-of-service (QoS) policy for network-on-package (NoP) links. To the best of our knowledge, this is the first fine-grained mapping framework for multiple DNNs on a multi-chiplet architecture. We implemented the proposed fine-grained mapping framework using genetic algorithm and simulated annealing algorithm. Experimental results show that our work achieves 7.18%–61.09% latency reduction under vision, language, and mixed workloads when compared with the state-of-the-art related work.**

***Index Terms*— Chiplet, deep neural network (DNN), mapping, multitenancy.**

NOMENCLATURE

Notation	Meaning
$G = (V, E)$	Computational graph of DNN.
$G' = (V', E')$	Simplified computational graph of DNN after PCA and hierarchical clustering.
B	Number of block in one DNN model after hierarchical clustering.
$\mathrm{TS}(G')$	Topological sorting of G'.
v	DNN block in the $\mathrm{TS}(G')$.
TP	Total number of time frames.
SP	Total number of spatial partition.
$T(v)$	Processing latency of v.
T_{total}	Overall processing latency.
$t_{i,j}$	Processing latency of the ith time frame in the jth spatial partition.
$K = \lvert C \rvert$	Total number of chiplet.
$K_{i,j}$	Number of chiplet of the ith time frame in the jth spatial partition.
N	Number of DNN model.
C	Set of chiplet.
C_v	Subset of chiplet allocated for V.
M	Number of iterations in the simulated annealing.

I. INTRODUCTION

WITH the development of artificial intelligence, the existing embedded systems and cloud systems require multiple deep neural networks (DNNs) to work in collaboration, especially for automatic driving applications and augmented and virtual reality (AR/VR) applications [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]. For example, perception in automatic driving systems leverages sensors including light detection and ranging (LiDAR) and camera for object detection and semantic segmentation. AR/VR requires multiple DNN models for face tracking, pose estimation, depth estimation, etc. What is more, cloud servers provide a multiple tenancy solution executing multiple workloads on shared infrastructures. Considering the trend toward increasingly larger and deeper networks, the cost barrier of monolithic (single die) DNN accelerators has limited the deployment of complex applications. Meanwhile, chiplet has been emerging as a promising technology which balances performance and cost [13], [14], [15], [16], [17], [18], [19]. Different from the monolithic chip, multi-chiplet architectures address the demanding needs of computing and storage by 2.5-D/3-D integration [20].

As shown in Fig. 1, the existing multi-DNN mapping algorithms were primarily designed for traditional monolithic chip

Manuscript received 26 December 2023; revised 6 April 2024 and 31 May 2024; accepted 15 July 2024. Date of publication 12 August 2024; date of current version 27 September 2024. This work was supported by the National Natural Science Foundation of China under Grant 62074097 and Grant 62072298. *(Corresponding authors: Yaoyao Ye; Guanghui He.)*

Jinming Zhang, Xuyan Wang, Yaoyao Ye, Dongxu Lyu, Guojie Xiong, Ningyi Xu, and Yong Lian are with the Department of Micro/Nano Electronics, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: yeyaoyao@sjtu.edu.cn).

Guanghui He is with the Department of Micro/Nano Electronics and the MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: guanghui.he@sjtu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2024.3438549.

Digital Object Identifier 10.1109/TVLSI.2024.3438549

architectures [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11] or multi-FPGA systems with fully connected network [12]. The memory access is uniform in bus or systolic-array-based interconnect network. Thus, the focus of mapping lies in the partitioning between shared resources of computation and memory bandwidth. PEs are split only along the temporal or spatial dimension. However, mapping multiple DNNs to a multi-chiplet architecture efficiently is still a challenging problem that cannot be solved effectively by the existing methods. The increasing number of chiplets leads to problems such as distributed memory access, nonuniform latency, and imbalanced traffic patterns [21], [22]. Moreover, partitioning a DNN model on a multi-chiplet architecture involves extra interdie communication on the network-on-package (NoP).

Specifically, the fine-grained mapping of DNN model includes loop tiling, loop exchange, task partitioning, and task scheduling. The complexity of mapping multiple DNNs on a multi-chiplet architecture is $\mathcal{O}(K^{(N*L)})$, where N represents the number of DNNs, L denotes the number of layers in a DNN, and K represents the total number of chiplets. Although recent work [22] has proposed a fine-grained mapping framework from DNN computation graphs to chiplet communication graphs, it is still limited to a single DNN. Different DNNs can be allocated to different numbers of chiplets, meaning each vertex in the DNN computation graph can be mapped to multiple vertices in the communication graph, adding complexity to the problem.

To alleviate the above issues, we propose M2M, a fine-grained mapping framework designed to accelerate multiple DNNs on a multi-chiplet architecture. As shown in Fig. 2, our proposed framework includes network partitioning [Fig. 2(b)], genetic loop optimization within each block [Fig. 2(c)], temporal and spatial scheduling [Fig. 2(d)], and communication-aware mapping [Fig. 2(e)]. The major contributions of this work are the following.

1) A fine-grained temporal and spatial block scheduling algorithm is proposed for chiplet-based accelerators, which explores different task execution orders and numbers of allocated chiplets. Flexible scheduling in both the temporal and spatial dimensions is fully explored in the whole design space, achieving 7.18%–61.09% latency reduction when compared with the state-of-the-art work.
2) A communication-aware block mapping is proposed based on the simulated annealing algorithm to map each DNN block to a specific chiplet. Various interconnect network (mesh, CMesh, and ring) and communication patterns are considered to mitigate link congestion during block mapping.
3) A fine-tuned quality of service (QoS) policy for NoP link is conducted to improve the communication efficiency. About 4.67%–34.63% latency reduction is achieved by our proposed fine-tuned bandwidth allocation policy.

To the best of our knowledge, this is the first fine-grained mapping framework for accelerating multiple DNNs on a multi-chiplet architecture. Our mapping framework covers aspects such as problem modeling, algorithm design, and simulator evaluation, providing a unified framework to systematically explore and map multi-DNNs to multi-chiplet.

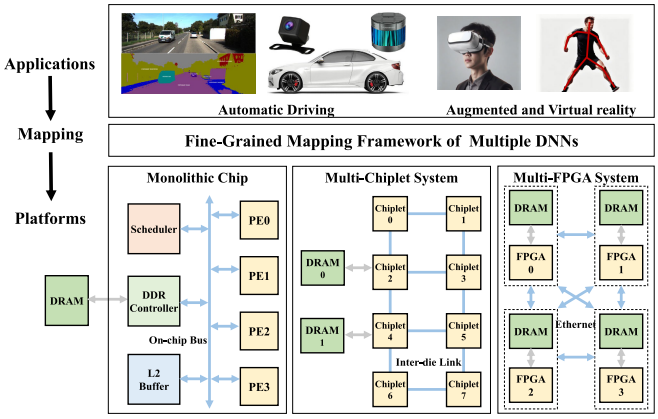


Fig. 1. Mapping framework of multiple DNNs for applications such as automatic driving and AR/VR to platforms such as monolithic chip [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], multi-chiplet system (this article), and multi-FPGA system [12]. The challenges of mapping multiple DNNs on a multi-chiplet architecture include distributed memory access, nonuniform latency, imbalanced traffic patterns, and increasing design space.

To accommodate increasingly large-scale DNN computing demands, we expand the existing multinetwork mapping algorithms in chiplet systems with more complex interconnect networks. With the increase in the number of chiplets, our algorithm achieved a better performance. Our work also provides insights for network mapping on monolithic chip and multi-FPGA systems, especially when complex interconnect networks become bottlenecks in the system.

This article is organized as follows. Section II introduces related work about multi-DNN mapping in monolithic chips and heterogeneous platforms. Section III introduces the workflow of M2M, and Section IV describes the implementation details of our work. Section V presents the experimental results and comparisons to the state-of-the-art. Finally, Section VI concludes this article.

II. Related Work

As shown in Table I, the existing multi-DNN mapping algorithms can be divided into three types: compute-centric, memory-centric, and communication-aware. Among them, PREMA [1] is the first work proposed for multi-DNN scheduling. PREMA prioritizes short jobs and adds preemption within a certain network layer. These modifications allow the scheduler to meet the latency requirements of high-priority task, while maintaining high throughput. Instead of running multiple DNNs through time multiplexing, spatial multiplexing-based scheduling algorithms were proposed for multitenancy [2], [3]. In [3], a 4 × 4 systolic array can run four DNNs simultaneously by adding additional interconnect links. An exhaustive search was conducted for resource reallocation. Moreover, a heuristic algorithm named Herald was proposed for heterogeneous accelerators [4], whose subaccelerators include NVDLA [23] and Shi-diannao [24].

Computing-centric scheduling algorithms allocate resources based on the dataflow styles of accelerators. However, these methods overlook the memory access requirements and may lead to data access bottlenecks. To address this issue, memory-centric mapping algorithms such as AI-MT [5] and

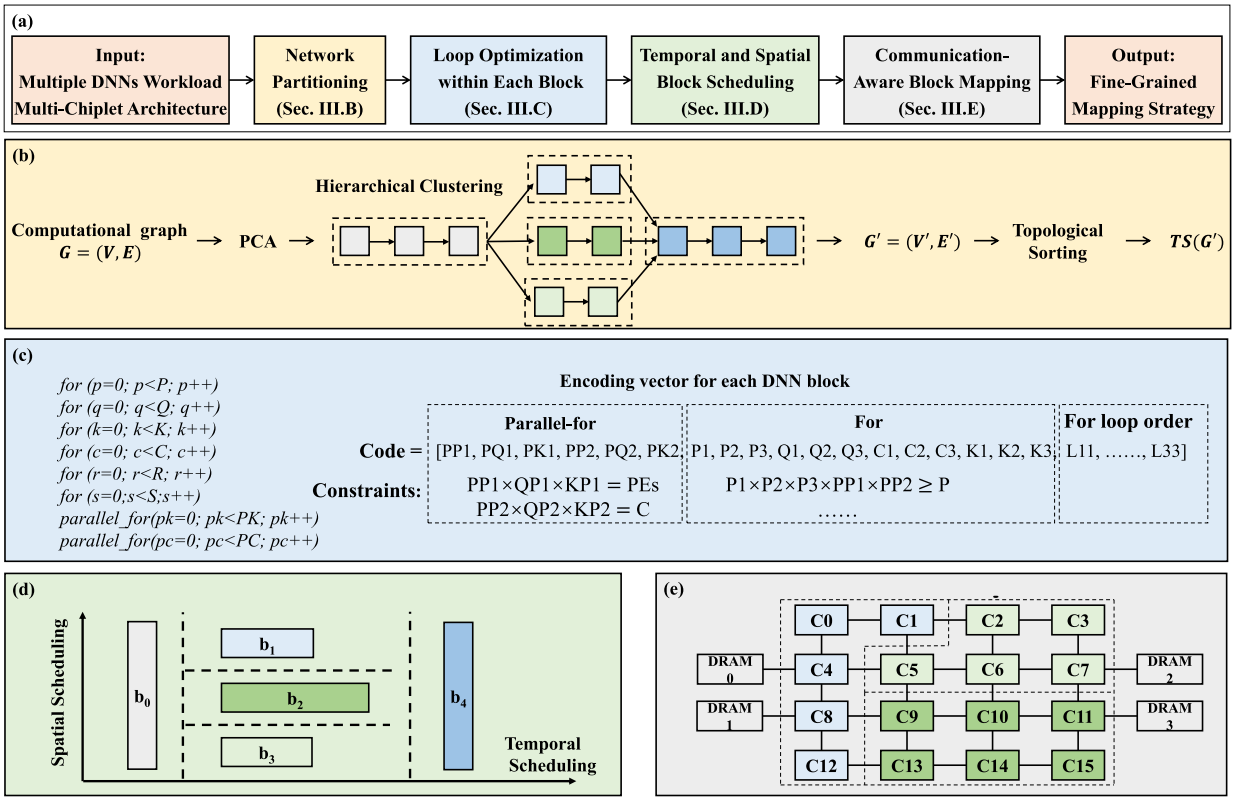


Fig. 2. M2M: a fine-grained mapping framework of multiple DNNs on a multi-chiplet architecture. (a) High-level overview of the framework. (b) Network partitioning for the computational graph $G = (V, E)$ including PCA, hierarchical clustering, and topological sorting. (c) Loop optimization within each block based on genetic algorithm. (d) Temporal and spatial block scheduling. (e) Communication-aware block mapping.

TABLE I
COMPARISONS OF MULTI-DNN MAPPING ALGORITHMS

Work	Type	Mapping algorithm	Architecture	Interconnect Network
PREMA [1]	Compute-centric	greedy algorithm	monolithic chip	Bus
Planaria [2]		exhaustive search	monolithic chip	Systolic array
Dataflow mirroring [3]		exhaustive search	monolithic chip	Systolic array
Herald [4]		greedy algorithm	monolithic chip	Bus
AI-MT [5]	Memory-centric	greedy algorithm	monolithic chip	Bus
Layerweaver [6]		greedy algorithm	monolithic chip	Bus
MAGMA [7]		genetic algorithm	monolithic chip	Bus
H3M [8]		CMA-ES	monolithic chip	Bus
MoCA [9]		dynamic algorithm	monolithic chip	Systolic array
V10 [10]		greedy algorithm	monolithic chip	Systolic array
H2H [12]	Communication-aware	greedy algorithm	multi-FPGA system	Fully-connected network
M2M		**genetic algorithm & simulated annealing algorithm**	**monolithic chip & multiple chiplets**	**Mesh/CMesh/Ring**

Layerweaver [6] use memory block prefetching based on the greedy algorithm, which loads weight and activation in advance to reduce the idle time of PE and improve hardware utilization. In addition, an automatic mapping framework called MAGMA was proposed in [7] to explore the overall mapping space using a custom genetic algorithm. Similarly, H3M [8] used the covariance-matrix adaptation evolution strategy (CMA-ES) to map multiple DNNs on multicore heterogeneous architectures. Similar to [4], all the subaccelerators in H3M share OFF-chip bandwidth through the global bus.

Differently, communication-aware mapping algorithms such as H2H [12] proposed a greedy algorithm to map multiple DNNs on multi-FPGA system. In H2H, each FPGA has a local DRAM memory, and each FPGA is connected by Ethernet in a fully connected network to achieve uniform bandwidth and latency. As compared in Table I, most traditional multi-DNN scheduling and mapping algorithms are generally designed for monolithic chips or multi-FPGA system. However, for multi-chiplet architectures, the interdie communication between chiplets can have a significant impact on performance. Problems of distributed memory access, nonuniform latency, and unbalanced traffic patterns cannot be solved by the above mapping algorithms.

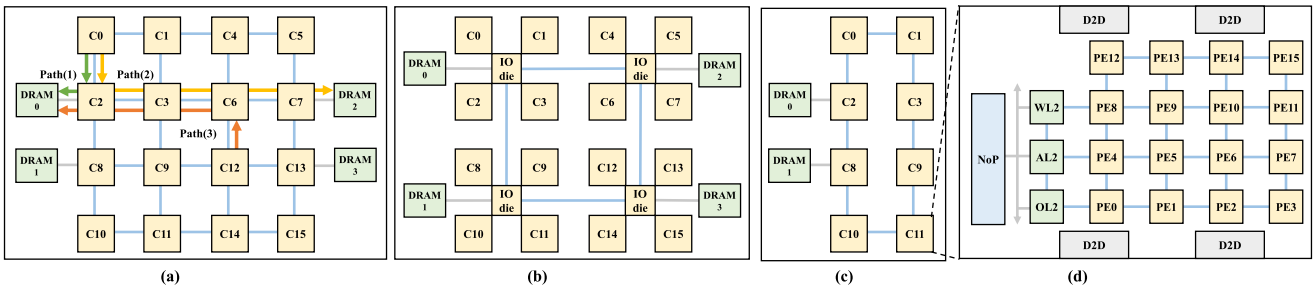


Fig. 3. Multi-chiplet architectures with different NoP networks. (a) Mesh. (b) CMesh. (c) Ring. (d) Chiplet architecture similar to SIMBA [21]. Compared with monolithic chips, multi-chiplet architectures face problems such as distributed memory access, nonuniform latency, and imbalanced traffic patterns.

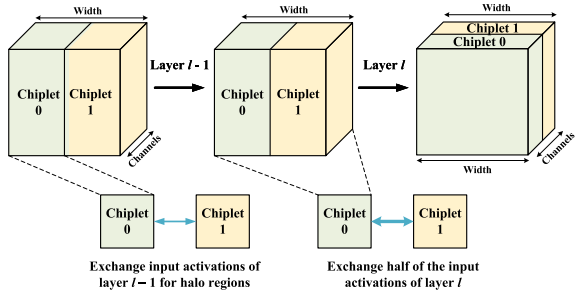


Fig. 4. Example of data coherence between two chiplets for layer $l-1$ and layer l.

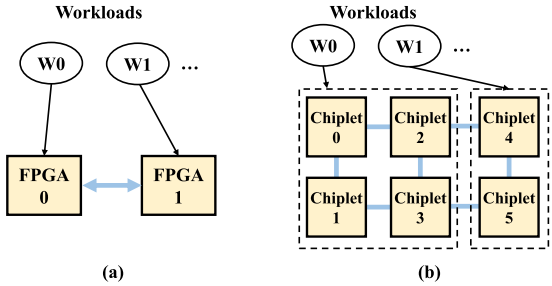


Fig. 5. Differences of existing multiaccelerator/multi-FPGA mapping algorithms and M2M. The number of chiplets is typically much higher than the number of accelerators/FPGAs. (a) Mapping in multi-FPGA system. (b) Mapping in multi-chiplet system.

For example, PREMA [1] prioritizes short jobs and adds preemption within a certain network layer. Efficient preemption mechanisms can be used similarly in multi-chiplet architecture. The problem is how to measure the preemption overhead in a complex interconnect network. As shown in Fig. 3(a), path (2) (from c0 to DRAM2) has a larger latency than path (1) (from c0 to DRAM0). In addition, the overlapping between path (1) and path (3) increases the interdie communication between c2 and DRAM0, resulting in imbalanced traffic patterns in the NoP.

Moreover, the presence of multiple DRAM memories further increases the complexity of the problem. Partitioning a DNN model on a multi-chiplet architecture will have extra interdie communication. As shown in Fig. 4, half of the input activations of layer l need to be exchanged between chiplets 0 and 1 for channel parallelism. Therefore, for chiplet-based architectures, we need to consider both the physical mapping positions of multiple DNNs and the interchiplet communication, which significantly expand the design space.

More importantly, merely regarding each accelerator/FPGA as a chiplet and migrating the multiaccelerator/multi-FPGA mapping algorithms to multi-chiplet cannot address the issue. The number of chiplets (36 chiplets in [21] and 32 chiplets in [19]) is typically much higher than the number of accelerators or FPGAs. As shown in Fig. 5, H2H [12] is limited to assigning each workload to a single computing node, while M2M assigns each workload to a set of computing nodes (from one to n nodes are possible). The allocation of chiplet will increase the complexity of the problem.

To alleviate these problems, we propose M2M, a fine-grained mapping framework for accelerating multiple DNNs on a multi-chiplet architecture. Temporal and spatial task scheduling algorithm and communication-aware task mapping algorithm are adopted for heterogeneous interconnect network.

III. Fine-Grained Mapping of Multiple DNNs on a Multi-Chiplet Architecture

A. Overall Framework

As shown in Fig. 2, we propose a fine-grained mapping framework for accelerating multiple DNNs on a multi-chiplet architecture, which mainly includes four steps.

1) *Network Partitioning [Fig. 2(b)]:* First, to reduce the complexity, we partition the heterogeneous network model into several DNN blocks using principal component analysis (PCA) and hierarchical clustering. Each DNN block is the basic granularity of exploration in scheduling and mapping.
2) *Genetic Loop Optimization Within Each Block [Fig. 2(c)]:* For each DNN block, we use genetic mapping algorithms [25] to explore the fine-grained loop optimization strategies including loop unrolling, loop tiling, and loop interchange. Since each DNN block can be processed in a different number of chiplets (1–16), the genetic algorithms should be repeated under different package-level parallelism.
3) *Temporal and Spatial Block Scheduling [Fig. 2(d)]:* The block execution order and the number of allocated chiplets for each DNN block are determined during temporal and spatial block scheduling. Data dependency between network layers has been taken into account along the temporal and spatial dimensions.
4) *Communication-Aware Block Mapping [Fig. 2(e)]:* After determining the execution order of each DNN block and the number of allocated chiplet, we map the network to the corresponding chiplets. The nonuniform latency of distributed memory access is taken into account for

TABLE II
NOTATIONS OF TENSOR DIMENSION

Tensor Dimension	Notation
Width of output activation	P
Height of output activation	Q
Channels of input activation	C
Channels of output activation	K
Width of kernel	R
Height of kernel	S

various interconnect networks such as mesh, CMesh, and ring.

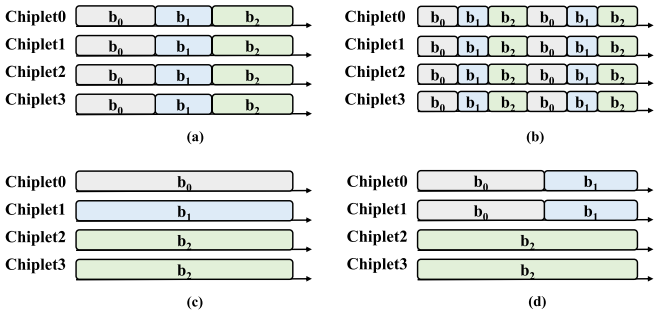


Fig. 6. Block scheduling for multi-DNN workloads. (a) Temporal scheduling. (b) Temporal preemption. (c) Spatial scheduling. (d) Temporal and spatial scheduling.

B. Network Partitioning Based on Hierarchical Clustering

Although it is easy to partition a DNN model along each layer, the computational complexity of fine-grained mapping is too large for networks with numerous layers. For example, there are 10^{850} mapping candidates for ResNet-50 deployed on EdgeTPU [26]. In addition, the computational complexity will grow exponentially for mapping multiple DNNs on a multi-chiplet architecture.

In fact, some adjacent layers share similar features size and similar optimal mapping strategy. The proposed network partitioning aims to consolidate similar network layers to simplify the complexity of mapping algorithm. Hierarchical clustering is adopted to group similar DNN layers into clusters based on a distance metric. We start with each DNN layer as its own cluster and find the closest (most similar) pair of clusters and then merge them.

To better extract features of DNN layers and characterize the distance metric, PCA is conducted for feature dimension reduction. The input features of PCA include size information of DNN layers. In PCA, eigenvalue decomposition is conducted on the covariance matrix and we pick the highest eigenvalues (principal components) to capture the most variance in the data.

The computational graph can be denoted as $G = (V, E)$, where V represents operation vertices and E contains directed edges indicating task dependencies. After PCA and hierarchical clustering, the simplified computation graph is represented as $G' = (V', E')$. A topological sorting is derived for G' to generate the execution order of operators

$$\mathrm{TS}(G') = \left[v_{\sigma(1)}, v_{\sigma(2)}, \ldots\right]. \tag{1}$$

Here, topological sorting is used for ordering the operators of a directed acyclic graph (G') in a linear order, such that for every directed edge $(v_i, v_j) \in E$, vertex v_i comes before v_j in the ordering.

C. Genetic Loop Optimization Within Each Block

The state-of-the-art DNN dataflow mapping strategies mainly include loop unrolling, loop tiling, and loop interchange. Among them, loop unrolling adopts different spatial parallelism dimensions for DNN accelerators. Loop tiling partitions one layer into small tiles for local buffering to increase the chance of data reuse.

In this work, we adopt a genetic algorithm similar to GAMMA [25] to explore the optimized mapping strategy for each DNN block. Different from mapping algorithms for a monolithic chip, both the parallelism parameters at the PE level and the chiplet level need to be explored. So we use parallel P1 (PP1), parallel Q1 (PQ1), parallel C1 (PC1), and parallel K1 (PK1) to represent the spatial parallelism parameters at the PE level. Similarly, PP2, PQ2, PC2, and PK2 represent the spatial parallelism parameters at the chiplet level.

The mapping encoding vector of DNN block is shown in Fig. 2(c), and related notations of tensor dimension are listed in Table II. During the crossover and mutation in genetic algorithms, these parameters are changed to explore different mapping strategies. We determine the loop order based on importance values. The loop with the highest importance value is executed in the innermost order. Thus, the optimization of loop order is conducted in a continuous domain.

We define the number of generations as 50. The population size is set as $20/8 * \min(\text{chiplet\_num}, 8)$. This is because as the number of chiplets increases, the candidates for loop parallelism and loop tiling grow exponentially. In each iteration, fitness is evaluated by our proposed performance prediction model (Section IV-A).

D. Temporal and Spatial Block Scheduling

Fig. 6(a) shows an example of block scheduling for multi-DNN workloads, where DNN blocks $b0$, $b1$, and $b2$ run sequentially. Each network block is allocated all the available hardware resources through time multiplexing. So the processing latency of a single network is minimized. However, as each layer can be either compute-intensive or memory-intensive, running only one network at a time may lead to inefficient resource utilization. Therefore, Fig. 6(b) demonstrates time multiplexing with preemption, achieving a balance between computation and memory access through network switching. However, context switching would potentially cause performance degradation. Moving activations and weights between ON-chip and DRAM will result in increased communication overhead.

Therefore, block scheduling along the spatial dimension, as depicted in Fig. 6(c), enables concurrent execution of

Algorithm 1 Temporal and Spatial Block Scheduling Algorithm

```
1: bestSchedule ← []
2: bestMakespan ← ∞
3: scheduleCandidates ← generateCandidates(Blocks)
4: for each tp in scheduleCandidates do
5:    currentSchedule ← []
6:    i ← 0
7:    for each sp in tp do
8:       block ← blockList[i++]
9:       minLoad ← ∞
10:      minChiplet ← null
11:      for each chiplet in C do
12:         if chiplet.load < minLoad then
13:            minLoad ← chiplet.load
14:            minChiplet ← chiplet
15:         end if
16:      end for
17:      currentSchedule.append((minChiplet, block))
18:      minChiplet.load ← minChiplet.load + block.time
19:   end for
20:   currentMakespan ← calculateMakespan(currentSchedule)
21:   if currentMakespan < bestMakespan then
22:      bestSchedule ← currentSchedule
23:      bestMakespan ← currentMakespan
24:   end if
25: end for
26: return bestSchedule
```

different DNNs. Compute-intensive networks can use more computing resources, while memory-intensive networks can occupy a larger share of memory bandwidth, thereby enhancing hardware utilization. However, when multiple DNNs are running concurrently, the ON-chip SRAM is partitioned, resulting in increased memory access for weights and activations from DRAM.

As shown in Fig. 6(d), in this work, we consider both temporal and spatial block scheduling in M2M. As modeled in (2), the function of block scheduling can be defined as the mapping from a computational graph G' to the design space of temporal and spatial block scheduling $\mathbb{R}^{\mathrm{TP}\times\mathrm{SP}}$

$$\begin{aligned}&\textbf{schedule}: \mathrm{TS}(G') \rightarrow \mathbb{R}^{\mathrm{TP}\times\mathrm{SP}}\\ &\textbf{schedule}(v) = [t_v, s_v]^\top\\ &\qquad 1 \le t_v \le \mathrm{TP}\\ &\qquad 1 \le s_v \le \mathrm{SP}.\end{aligned} \tag{2}$$

Here, v denotes a DNN block in G', and t_v is a short time frame for v to execute. We suppose there are TP time frames in total. Between different time frames, context switching is necessary, incurring OFF-chip communication overhead. Within each time frame, some DNN blocks run simultaneously, using a spatial partition of the hardware resources. s_v denotes the index of spatial partition, which is allocated with a certain number of chiplets (K_{t_v,s_v}). Detailed notations used in fine-grained mapping of multiple DNNs are summarized in Nomenclature.

In line 20 in Algorithm 1, the processing latency of each DNN block has already been calculated in Section III-C. Suppose $t_{i,j}$ denotes the processing latency of the ith time frame in the jth spatial partition. Since v is scheduled to $[t_v, s_v]^\top$ in (2), the latency of time frame t_{t_v,s_v} can be modeled as follows:

$$T(v) = t_{t_v,s_v}. \tag{3}$$

Within each time frame, the processing latency depends on the slowest workload

$$\min T_{\mathrm{total}} = \min \sum_{i=1}^{\mathrm{TP}} \left\{ \max_j t_{i,j} \right\}. \tag{4}$$

In (4), T_{total} represents the overall processing latency. The spatial partitioning is established on the granularity of chiplets. Assuming there are a total of K chiplets, the spatial partitioning can be expressed as follows:

$$\sum_{j=1}^{\mathrm{SP}} K_{i,j} = K. \tag{5}$$

In (5), $K_{i,j}$ denotes the number of chiplet of the ith time frame within the jth spatial partition. The sum of the number of chiplets within all the spatial partitions in each time frame should equal the total number of chiplets (K).

During the temporal and spatial block scheduling, data dependency between network layers should be considered in the following equation:

$$\begin{aligned}&\forall i < j\\ &\textbf{schedule}(v_{\sigma(i)})[0] < \textbf{schedule}(v_{\sigma(j)})[0].\end{aligned} \tag{6}$$

Here, $\sigma(i)$ represents the topological sorting order as defined in (1). Equation (6) ensures $\forall i < j$ in the topological order, the scheduling time of block i ($\textbf{schedule}(v_{\sigma(i)})[0]$) should be earlier than the scheduling time of block j ($\textbf{schedule}(v_{\sigma(j)})[0]$).

In this work, an exhaustive search is conducted to find the optimal scheduling solution which minimizes the overall latency (T_{total}). The detailed algorithm is presented in Algorithm 1. The time and space complexity is $\mathcal{O}(N * B * \mathrm{TP} * \mathrm{SP} * K)$. Here, N denotes the number of DNN model. B denotes the number of block in one DNN model. TP and SP denote the number of time frames and spatial partition, respectively. K denotes the number of chiplet.

E. Communication-Aware Block Mapping

As defined in (7), the function of block mapping is to map each DNN workload onto a specific chiplet subset $C_v \subseteq C$

$$\begin{aligned}&\textbf{map}: \mathbb{R}^{\mathrm{TP}\times\mathrm{SP}} \rightarrow C\\ &\textbf{map}([t_v, s_v]^\top) = C_v \subseteq C\\ &\qquad 1 \le t_v \le \mathrm{TP}\\ &\qquad 1 \le s_v \le \mathrm{SP}.\end{aligned} \tag{7}$$

For a multi-chiplet architecture, distributed memory leads to nonuniform access latency and imbalanced traffic loads. The interdie bandwidth is considerably lower than the on-chiplet bandwidth, thus may become a potential bottleneck for the system. Consequently, mapping DNN workloads onto different chiplets can lead to varying performance. Taking

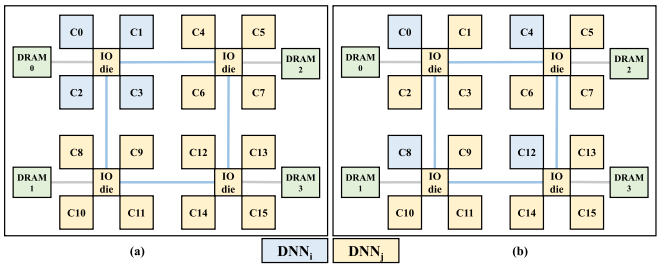


Fig. 7. Example of communication-aware mapping for DNN$_i$ and DNN$_j$. (a) Mapping DNN$_i$ in one cluster. (a) Mapping DNN$_j$ in one cluster.

Algorithm 2 Communication-Aware Block Mapping Based on Simulated Annealing Algorithm

```
Mapping ← InitialMapping
bestMapping ← Mapping
Temp ← InitialTemp
while not StoppingCondition do
    newMapping ← GenerateNeighbor(Mapping)
    delta ← Latency(newMapping) − Latency(Mapping)
    Mapping ← newMapping
    if Latency(Mapping) < Latency(bestMapping) then
        bestMapping ← Mapping
    end if
    Temp ← Cooling(Temp, CoolingRate)
end while
return bestMapping
```

CMesh in Fig. 3(c) as an example, every four compute chiplets share a DDR controller. Fig. 7 shows an example of communication-aware block mapping. In Fig. 7(a), each network model is assigned to the same cluster ($C_{\text{DNN}_i} = \{c0, c1, c2, c3\}$), whereas in Fig. 7(b), each network model is allocated to different clusters ($C_{\text{DNN}_i} = \{c0, c4, c8, c12\}$). Notably, Fig. 7(b) enables effective utilization of DDR bandwidth between the compute-intensive DNN$_i$ and the memory-intensive DNN$_j$ within a single cluster. Combining different types of workloads is better for enhancing bandwidth utilization. However, as illustrated in Fig. 4, interdie communication is necessary within the same network model to ensure data coherence for halo regions, while no communication is required between DNN$_i$ and DNN$_j$. Consequently, the mapping strategy depicted in Fig. 7(a) diminishes interdie communication between clusters.

To alleviate this problem, we propose a communication-aware mapping algorithm (Algorithm 2) based on the simulated annealing, which assigns each DNN block to the corresponding chiplet based on the scheduling results. As shown in Fig. 7(b), for architectures with multiple DRAM chips, $c8$ writes the output activation to the nearest DRAM (DRAM 1), while loading the input activation from the remote DRAM (DRAM 0, 2, 3) and local DRAM (DRAM 1). The final execution time will be the processing latency of each DNN block plus the additional overhead caused by interdie communication. Section IV-A will describe our proposed performance prediction model for multi-DNN workload.

As shown in Algorithm 2, simulated annealing uses a temperature parameter to control the probability of accepting worse solutions. The temperature decreases over time according to a predefined schedule. Initially, the high temperature allows the algorithm to explore a broad solution space. As the temperature decreases, the algorithm focuses on refining the solutions. At each iteration, a neighboring solution of swapping the assignments of blocks between chiplets is generated. We keep track of the best block mapping encountered during the optimization process. Using communication-aware block mapping based on simulated annealing algorithm, different block mappings are explored, allowing it to escape local optima and find a solution that minimizes communication cost, leading to improved overall system performance. The time complexity of the proposed algorithm is $\mathcal{O}(M * \text{TP})$, where M denotes the number of iterations and TP denotes the number of time frames.

IV. Implementation of the Fine-Grained Mapping Framework

A. Performance Prediction for Multi-Chiplet Architecture

When mapping multiple DNNs to a multi-chiplet architecture, a fast and accurate prediction model is required to evaluate the performance of different mapping strategies. However, traditional DNN simulators such as MAESTRO [27], DNN Chip Predictor [28], and Timeloop [29] only support the traditional bus architecture. These simulators cannot support the modeling of complex interconnect networks in multi-chiplet architectures. Meanwhile, cycle-accurate simulators such as Booksim [30] and gem5 [31] are mainly aimed at general-purpose processors. Running DNN workloads on these cycle-accurate simulators is time-consuming due to multiple loop nesting and large computational complexity.

In this work, we propose a performance prediction model for multi-chiplet architectures which supports complex networks such as mesh, CMesh, and ring.

As shown in Fig. 8, we can analyze the latency from the following three aspects.

1) Latency of chiplet computation (MAC_cycle$_i$).
2) Latency of DRAM access (DRAM_cycle$_i$).
3) Latency of interdie communication (NoP_cycle$_i$).

Complex connection networks have no impact on computational latency, and thus we can obtain MAC_cycle$_i$ using traditional simulators such as MAESTRO. The latency of DRAM access (DRAM_cycle$_i$) can be determined by the total DRAM traffic (traffic$_{\text{DRAM}}$) and DRAM bandwidth (BW$_{\text{DRAM}}$). Similar to a monolithic chip, traffic$_{\text{DRAM}}$ can be calculated based on the parallelism configurations and the total data volume of weights and activations [27], [32], [33]

$$\text{DRAM\_cycle}_i = \text{traffic}_{\text{DRAM}}/\text{BW}_{\text{DRAM}} * f_{\text{clk}}. \quad (8)$$

As for the latency of interdie communication (NoP_cycle$_i$), imbalanced traffic patterns can result in varying loads on different NoP links, leading to differences in transmission times for each chiplet. However, due to synchronization operations among chiplets within the same network, communication latency depends on the slowest link (the most congested link).

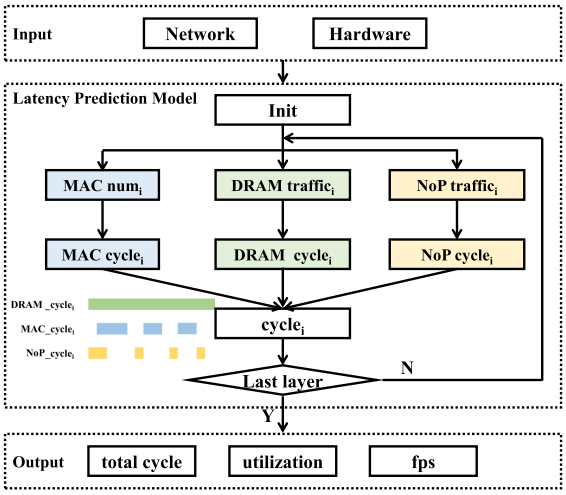


Fig. 8. Proposed performance prediction model for multi-DNN workloads.

TABLE III
INTERDIE COMMUNICATION OF ACTIVATION BETWEEN LAYER $l-1$ AND LAYER l FOR FOUR CHIPLETS [33]

Parallelism type		layer l		
		P-Parallel	PK-Parallel	K-Parallel
layer l-1	P-Parallel	(R-1)*Q*K	(P/4+R-1)*Q*K	3/4*P*Q*K
	PK-Parallel	1/8*P*Q*K	(P/4+R-1)*Q*K	3/4*P*Q*K
	K-Parallel	3/16*P*Q*K	3/4*(P/2+R-1)*Q*K	3/4*P*Q*K

TABLE IV
ENERGY CONSUMPTION OF DIFFERENT OPERATIONS OF CHIPLETS AT 16 nm [32]

Operation	Energy (pJ/bit)
DRAM access	8.75
Inter-die communication (GRS)	1.17
SRAM access	0.81
8 bit MAC	0.024

Based on the routing algorithm, we calculate the traffic load of each link, including weight loading, input activation loading, and output activation storing. Thus, the latency of interdie communication (NoP_cycle_i) can be determined by the NoP traffic ($\text{traffic}_{\text{NoP}}$) and NoP bandwidth ($\text{BW}_{\text{NoP}}$)

$$\text{NoP\_cycle}_i = \max_{\text{link}}\left(\sum \text{traffic}_{\text{NoP}}\right)/\text{BW}_{\text{NoP}} * f_{\text{clk}}. \quad (9)$$

To quantize the effect of multiple DRAM memories, we partition the process of each DNN layer into three parts: weight and input activation loading, computing, and output activation storing. Synchronization among different chiplets is conducted for each part. During weight and input activation loading, the interdie communication depends on the parallelism dimensions between two DNN layers (Table III). During output activation storing, since the output activations of each chiplet are only stored in the nearest DRAM die, the interdie communication occurs between chiplets and each nearest DRAM.

The implementation of ping-pong buffers can enable overlapping of computation and communication time. Thus, for layer i, the processing latency t_i can be modeled as (10). The overall processing latency of layer i is constrained by the slowest among three components: PE MAC computation (MAC_cycle_i), accessing data from DRAM (DRAM_cycle_i), and NoP communication (NoP_cycle_i)

$$t_i = \max\{\text{MAC\_cycle}_i, \text{DRAM\_cycle}_i, \text{NoP\_cycle}_i\}. \quad (10)$$

In addition, we evaluated the energy overhead similar to work [32], which includes the energy of DRAM access, L2 SRAM access, L1 SRAM access, die-to-die communication ground-referenced signaling (GRS), and PE MAC unit. Detailed energy consumption of different operations in 16 nm is shown in Table IV. The energy consumption of DRAM access ($\text{Energy}_{\text{DRAM}}$) can be determined by the total DRAM access ($\text{Access}_{\text{DRAM}}$) multiplying the energy overhead per operation (e_{DRAM})

$$\text{Energy}_{\text{DRAM}} = e_{\text{DRAM}} * \text{Access}_{\text{DRAM}}. \quad (11)$$

Similarly, the energy consumption of L2 SRAM, L1 SRAM, GRS, and PE MAC can be determined by the total L2 SRAM access (Access_{L2}), the total L1 SRAM access (Access_{L1}), the interdie communication ($\text{Access}_{\text{GRS}}$), and the number of MAC, respectively. Here, $\text{Access}_{\text{DRAM}}$, Access_{L2}, Access_{L1}, and $\text{Access}_{\text{GRS}}$ can be calculated using traditional simulators such as MAESTRO. Finally, the equation of energy consumption is summarized as follows:

$$\text{Energy}_{\text{total}} = \text{Energy}_{\text{DRAM}} + \text{Energy}_{L2} + \text{Energy}_{L1} + \text{Energy}_{\text{GRS}} + \text{Energy}_{\text{MAC}}. \quad (12)$$

B. Fine-Tuned QoS Policy for NoP Link

Due to the limited number of pins and longer wires in substrate, the interchiplet communication bandwidth is significantly lower than the ON-chip communication bandwidth [20]. Thus, NoP link tends to be a potential bottleneck. Furthermore, different networks have varying communication requirements, leading to load imbalances. For multiple DNNs, QoS policy for NoP links directly impacts the overall latency. A fine-tuning QoS policy for NoP links requires a balance between the throughput of the applications running on the network and the capabilities of the underlying network infrastructure.

Taking Fig. 9 as an example, DNN_0 and DNN_1 are running simultaneously in an eight-chiplet system. If DNN_0 is the bottleneck for the overall system performance, and interchiplet communication is the bottleneck for DNN_0, if more NoP bandwidths and DDR bandwidths are allocated to DNN_0, the overall processing latency can be reduced. More precisely, due to the varying loads on each NoP link, fine-tuned bandwidth allocation is required for each NoP router.

In a multi-chiplet architecture, each NoP router has an arbiter for packets' selection and arbitration. Instead of traditional round-robin arbitration, we adopt a fine-tuned bandwidth allocation. First, incoming traffic is classified into different priorities. We grant a higher priority for the slowest tasks, which are the performance bottlenecks. For each NoP link, a weighted fair queuing algorithm is applied to balance the allocation, ensuring that lower priority traffic gets its share of resources without compromising higher priority traffic.

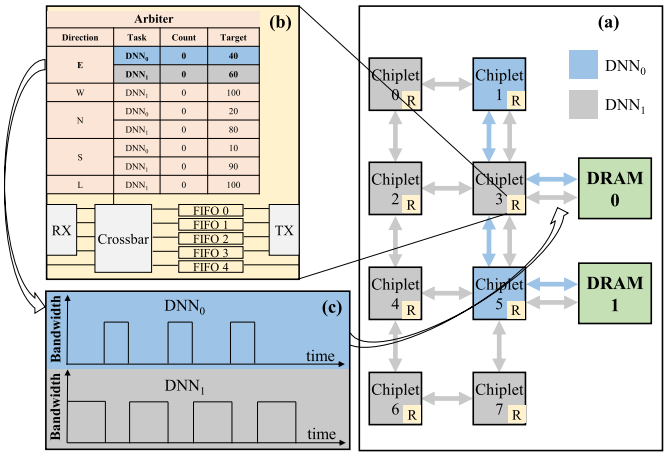


Fig. 9. Proposed fine-tuned QoS policy deployed in eight chiplets for DNN_0 and DNN_1. (a) Overall mesh network of eight chiplets. (b) NoP router of chiplet3 with fine-tuned QoS policy. (c) For DNN_0 and DNN_1, the allocated bandwidth of NoP link between chiplet3 and DRAM0 is set to 40:60.

TABLE V
MULTI-DNN WORKLOADS

Workload	DNN Models
Vision	ResNet-50 [34], YOLOv2 [35], ViT [36], Unet [37]
Language	GNMT [38], BERT [39], Ncf [40]
Mixed	VGG-16 [41], ResNet-18 [34], GNMT [38], BERT [39]

TABLE VI
INTERCONNECT NETWORK CONFIGURATIONS

Interconnect Network	Ring	Mesh	CMesh
Throughput per chiplet / TOPS	4	4	4
Frequence / GHz	2	2	2
Chiplet Num	8	16	16
Throughput / TOPS	32	64	64
DDR4 Bandwidth / Gbps	450	900	900
HBM Bandwidth / Gbps	2048	4096	4096
Inter-Die Link Bandwidth / Gbps	100	100	100
On-Die Link Bandwidth / Gbps	68	68	68

TABLE VII
MEMORY HIERARCHY OF COMPUTE CHIPLET [21]

Description		Parameter
Chiplet level	PE Number	16
	AL2 / KB	48
	WL2 / KB	32
	OL2 / KB	48
PE level	MAC num	64
	AL1 / KB	8
	WL1 / KB	32
	OL1 / KB	3

As shown in Fig. 9(b), different tasks (DNN_0 and DNN_1) are allocated distinct bandwidths across five directions: East (E), west (W), north (N), south (S) and local (L). FIFOs and loop counters are used inside the NoP router to control traffic. For example, as shown in Fig. 9(c), for the NoP link between Chiplet3 and DRAM0 [direction E for the NoP router of Chiplet3 in Fig. 9(b)], the interdie bandwidth is set to 40:60 for DNN_0 and DNN_1.

In this way, the latency of critical traffic can be reduced, since communication-limited task benefits from lower latency as it is granted priority access to the network. Usually, the DNN mapping algorithm can be conducted offline to generate instruction before use. Therefore, static QoS policy can be applied for each NoP link to achieve lower overall processing latency.

V. RESULTS ANALYSIS AND COMPARISONS

A. Experimental Setup

Experimental workloads include typical network models in vision, language, and mixed applications (Table V). CNN, RNN, and transformer are all evaluated in this work. The hardware parameters are shown in Tables VI and VII, including mesh, CMesh, and ring network. For each compute chiplet, we assume the same configuration with the SIMBA architecture [21]. As shown in Fig. 3(d), 32-kB WL2 (L2 buffer for weight), 48-kB AL2 (L2 buffer for activation), 48-kB OL2 (L2 buffer for output), and 16 PEs are connected by on-chip links. The throughput of each chiplet is 4 TOPS when operating at 2 GHz. Similar to related works [21], [32], [42], we use GRS as NoP link. Each GRS link operates at a data rate of 25 Gb/s/pin with 1.17-pJ/bit energy efficiency. The DRAM types include DDR4 and HBM for edge and cloud scenarios, respectively. The number of compute chiplet changes from 4 to 16 for variously configured systems.

For mesh network shown in Fig. 3(a), four compute chiplets (C2, C7, C8, and C13) at the package edge are connected with DDR or HBM. The paths are determined by the YX routing algorithm. In YX routing, the packet is routed vertically along the Y dimension first until it reaches the row where the destination node resides. Then the packet is routed horizontally along the X dimension until it reaches the destination node [43]. For CMesh shown in Fig. 3(b), every four compute chiplets are connected by one IO die. Each cluster is connected by the IO die via GRS links. For ring network shown in Fig. 3(c), the paths are determined by the shortest path routing algorithm. To avoid deadlocks between different tasks, bubble flow control [44], [45], [46] is adopted for NoP links. Since the ring network is hard to scale with constant bisection bandwidth, only eight compute chiplets at most are considered in this work.

We compared M2M with the state-of-the-art related works including PREMA [1], Planaria [2], Herald [4], and H2H [12]. We implemented these baseline works according to open-source code or pseudocode described in their articles. Our proposed algorithm is implemented in python and is now open-source in https://github.com/tiaozhanzhe/M2M.

Fig. 2 describes the tool flow in M2M. The input of our tool flow is multiple DNN workloads and hardware parameters. The output of our tool flow is the optimal mapping strategy. Our proposed fine-grained mapping framework includes four steps: network partitioning (Section III-B), loop optimization within each block (Section III-C), temporal and spatial block scheduling (Section III-D), and communication-aware block mapping (Section III-E).

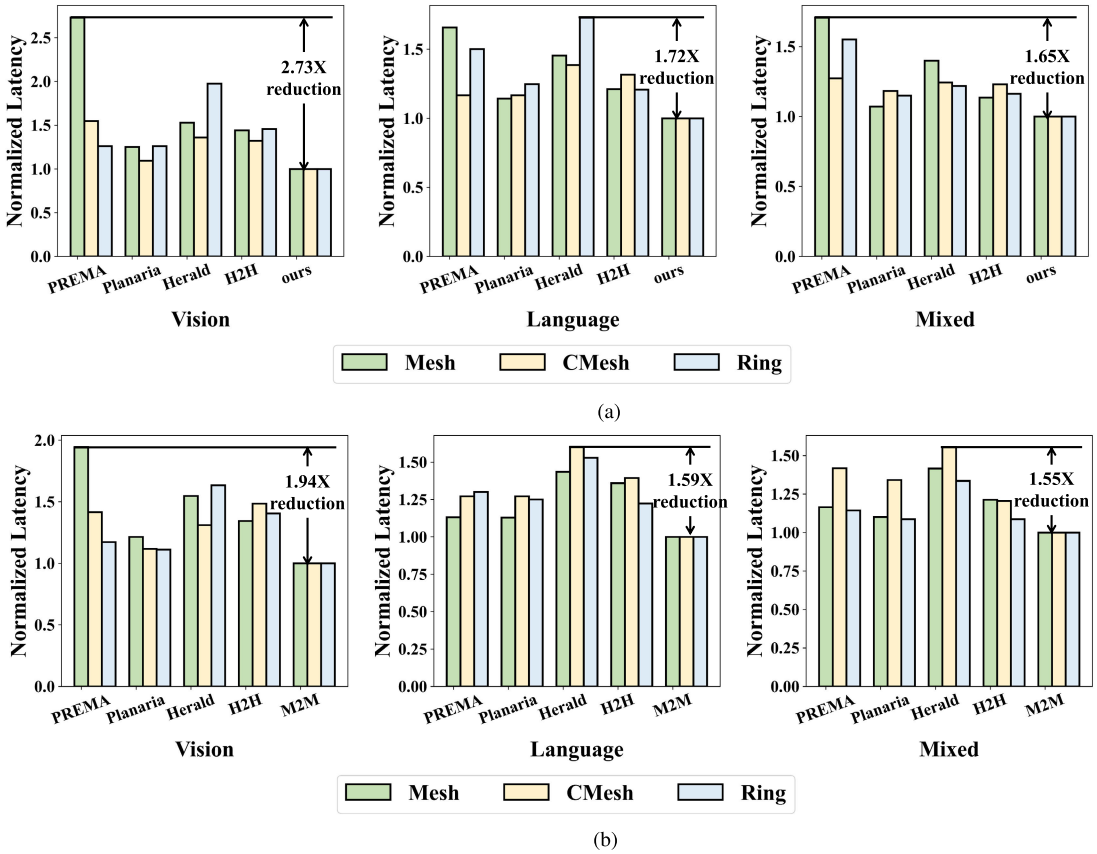


Fig. 10. Normalized latency comparisons of M2M compared with PREMA [1], Planaria [2], Herald [4], and H2H [12] for multi-chiplet architectures in mesh, CMesh, and ring NoP network. DNN workloads include vision, language, and mixed applications. (a) Normalized latency comparisons of fine-grained mapping strategies deployed in a multi-chiplet architecture with HBM. (b) Normalized latency comparisons of fine-grained mapping strategies deployed in a multi-chiplet architecture with DDR4.

TABLE VIII
RESULTS OF EDP AND ENERGY IN MESH, CMESH, AND RING NETWORK

	Mesh		CMesh		Ring	
	EDP (pJ·s)	**Energy (pJ)**	**EDP (pJ·s)**	**Energy (pJ)**	**EDP (pJ·s)**	**Energy (pJ)**
Vision	2.51E+13	2.41E+15	1.67E+12	2.89E+14	9.72E+13	4.73E+15
NLP	2.04E+09	3.23E+11	1.19E+09	3.91E+10	6.96E+09	5.96E+11
Mixed	6.10E+11	8.73E+13	1.20E+11	3.51E+12	5.01E+12	4.09E+14

B. Comparison for Processing Latency

PREMA uses temporal preemption for multitask workload. Since only one network can run at a time, PREMA can achieve the lowest latency for each DNN. However, a certain network monopolizing the entire hardware resources also results in low hardware utilization, especially for HBM-based systems with abundant DRAM bandwidth. For Herald, each compute chiplet can be regarded as a subaccelerator. Each DNN block is allocated among these subaccelerators using a greedy mapping algorithm. The greedy algorithm works well for systems with two or three subaccelerators but misses the globally optimal strategy for a larger architecture with 16 compute chiplets.

As shown in Fig. 10, our proposed mapping framework M2M achieves the lowest overall processing latency in vision, language, and mixed applications. The DRAM bandwidth ranges from 900 Gb/s (DDR4) to 4096 Gb/s (HBM), which represents memory-bound architecture and computation-bound architecture, respectively. In overall, M2M achieves 11.05%–61.09% latency reduction for the computation-bound architecture and 7.18%–48.5% latency reduction for the memory-bound architecture.

Compared with PREMA, M2M achieves 1.16× −2.73× latency reduction. Temporal scheduling like PREMA leads to inefficient resource utilization in the vision and mixed workload because the vision and mixed workload comprises both compute-intensive tasks (mostly CNN) and memory-intensive tasks (mostly transformer). But for language workload, which is primarily memory-intensive, the performance gap between PREMA and M2M is expected to significantly narrow.

Energy is also evaluated in this work. The results of energy-delay product (EDP) and energy are shown in Table VIII. CMesh offers better performance compared with mesh network due to the ability of broadcasting and multicasting through clusters. Ring is limited by constant bisection

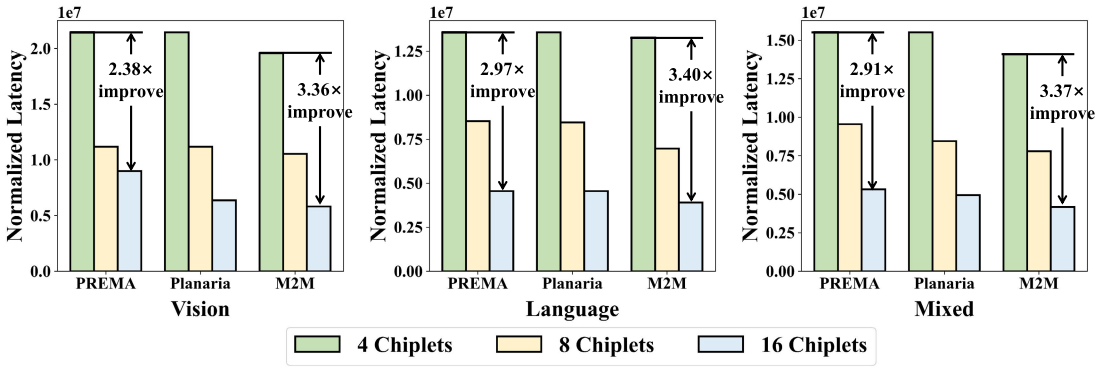


Fig. 11. Latency comparison with a different number of chiplets.

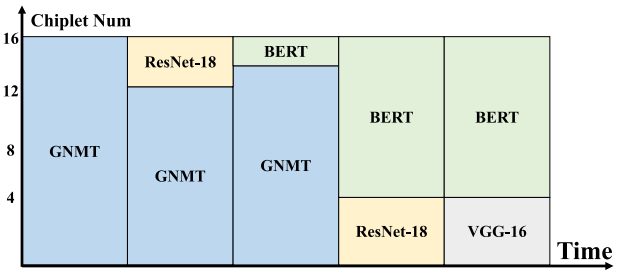


Fig. 12. Scheduling result for mixed workload in mesh network with HBM.

bandwidth and poor scalability, resulting in significantly inferior performance compared with both mesh and CMesh.

C. Latency Comparison With a Different Number of Chiplets

Fig. 11 shows comparison of the latency with a different number of chiplets. We explored a fine-grained mapping strategy for the CMesh-based architecture with HBM, ranging from four chiplets (16 TOPS) to 16 chiplets (64 TOPS). For a four-chiplet system, M2M achieves similar performance with PREMA and Planaria. However, the hardware utilization of PREMA degrades with the increase in chiplet number. When the number of chiplet increased from 4 to 16, PREMA only achieves a 2.38× −2.97× latency reduction. When compared with PREMA, multiple DNN models run spatially across different chiplets in M2M and Planaria. The processing latency of M2M is comparatively linearly reduced, achieving a 3.36× −3.40× latency reduction. Since both temporal and spatial task scheduling are considered in M2M, a latency reduction by 2.2%–35.38% is achieved when compared with Planaria. Despite the ideal 4× performance gain, interchiplet communication gradually becomes a bottleneck, leading to decreased hardware utilization. As the number of nodes increases, our proposed multi-DNN mapping algorithm tailored for multi-chiplet architectures exhibit even greater advantages compared with the existing algorithms target for the monolithic chip.

Fig. 12 shows an example of scheduling result for mixed workload in mesh network with HBM. At times, one network monopolizes all the hardware resources, while at other times, two networks share resources and compute simultaneously. Due to the significantly larger computation of BERT (96 GOPs) and GNMT (24 GOPs) when compared with ResNet-18 (1.79 GOPs) and VGG-16 (15 GOPs), BERT and GNMT occupy most of the hardware resources most of the time.

TABLE IX
LATENCY OF EACH MODEL INDIVIDUALLY IN MIXED WORKLOAD (16 CHIPLETS IN MESH NETWORK WITH HBM)

	GNMT	BERT	ResNet-18	VGG-16
PREMA	4.02E+06	6.51E+06	9.43E+05	1.33E+06
Herald	9.81E+06	7.28E+06	1.02E+06	2.27E+06
H2H	1.09E+07	1.09E+07	3.74E+06	6.10E+06
M2M	5.27E+06	1.02E+07	1.00E+07	7.94E+06

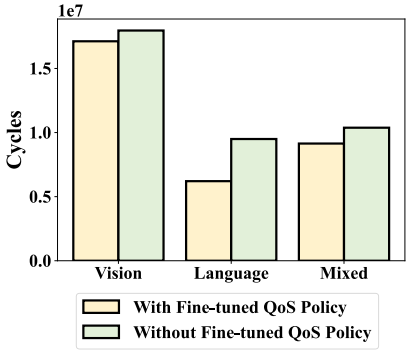


Fig. 13. Latency reduction by the fine-tuned QoS policy.

TABLE X
PREDICTION ACCURACY OF THE PROPOSED PERFORMANCE PREDICTION MODEL COMPARED WITH GEM5

	gem5	Ours	Accuracy
AlexNet	62690	61800	98.56%
LeNet	15825	15696	99.18%
ResNet-18	105743	104274	98.59%
ResNet-50	193595	194784	99.39%
VGG-16	78365	80460	97.40%
Darknet-19	159873	158982	99.44%
ViT	838776	815328	97.12%

Our optimization objective is to minimize the overall latency that all the workloads have been completed. For instance, in the field of autonomous driving, each frame of the image needs to be processed by object detection network, semantic segmentation network, path planning network, and so on. So the overall latency for all the DNNs for one frame,

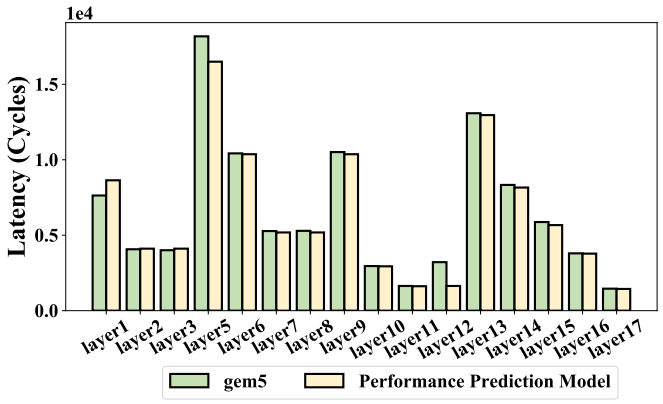


Fig. 14. Prediction error of ResNet-18 compared with gem5.

reflecting the system's throughput and frame rate. Minimizing the overall latency does not necessarily mean minimizing the processing time of each individual DNN. As discussed in Section III-D, when each network is allocated with all hardware resources using time multiplexing [Fig. 6(a)], the processing latency of a single network is minimized. Therefore, in Table IX, the latency of each model individually in PREMA outperforms other works. However, as each layer can be either compute-intensive or memory-intensive, running only one network at a time may lead to inefficient resource utilization. The concurrent execution of different network models may improve resource utilization. In M2M, we explore the possible chiplet allocation strategy [Fig. 6(d)]. While it is not possible to ensure that the latency of each network model is lowest, M2M can achieve the minimized overall latency.

D. Effect of the Fine-Tuned QoS Policy for NoP Links

Fig. 13 demonstrates the effect of the proposed fine-tuned QoS policy for NoP links. Among the vision, language, and mixed workloads, the fine-tuned link bandwidth control achieves a latency reduction by 4.67%–34.63%. It achieves a larger latency reduction under the language workload since GNMT and BERT are sensitive to the NoP link bandwidth. Therefore, we can conclude that the proposed fine-tuned QoS policy can balance the hardware resources between the memory-intensive network and the compute-intensive network.

E. Validation of the Proposed Performance Prediction Model

We validated the accuracy of our proposed performance prediction model for a multi-chiplet architecture. We generated the NoC and NoP packet traces of the workload and obtained the processing latency using the cycle-accurate simulator gem5. Fig. 14 demonstrates that the processing latency of ResNet-18 in our performance prediction model compared with gem5. Additional results beyond ResNet-18 are shown in Table X. The prediction accuracy is over 97% for AlexNet, LeNet, ResNet-18, ResNet-50, VGG-16, Darknet-19, and ViT. Behavioral-level modeling indeed introduces some inaccuracies compared WITH cycle-accurate simulation. The prediction error comes from the different NoC/NoP behavior modeling, including packet arbitration, and virtual channels. Since cycle-accurate simulator is time-consuming, which is not suitable for mapping exploration, our proposed prediction model achieves a good tradeoff between speed and accuracy.

VI. Conclusion

In conclusion, a fine-grained mapping framework named M2M was proposed to accelerate multiple DNNs on multi-chiplet architectures. The workflow of the proposed framework includes network partitioning, single-network mapping, temporal and spatial task scheduling, and communication-aware mapping. In addition, we proposed a fine-tuned QoS policy for NoP links to reduce the overall processing latency. The experimental results demonstrated that our work achieves a latency reduction by 7.18%–61.09% compared with the state-of-the-art approaches in vision, language, and mixed applications.

References

[1] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 220–233.

[2] S. Ghodrati et al., "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 681–697.

[3] J. Lee, J. Choi, J. Kim, J. Lee, and Y. Kim, "Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array NPUs," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 247–252.

[4] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Mar. 2021, pp. 71–83.

[5] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 940–953.

[6] Y. H. Oh et al., "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 584–597.

[7] S.-C. Kao and T. Krishna, "MAGMA: An optimization framework for mapping multiple DNNs on multiple accelerator cores," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 814–830.

[8] S. Zeng et al., "Serving multi-DNN workloads on FPGAs: A coordinated architecture, scheduling, and mapping perspective," *IEEE Trans. Comput.*, vol. 72, no. 5, pp. 1314–1328, May 2023.

[9] S. Kim, H. Genc, V. V. Nikiforov, K. Asanovic, B. Nikolic, and Y. S. Shao, "MoCA: Memory-centric, adaptive execution for multi-tenant deep neural networks," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 828–841.

[10] Y. Xue, Y. Liu, L. Nai, and J. Huang, "V10: Hardware-assisted NPU multi-tenancy for improved resource utilization and fairness," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–15.

[11] S. Kim, J. Zhao, K. Asanovic, B. Nikolic, and Y. S. Shao, "AuRORA: Virtualized accelerator orchestration for multi-tenant workloads," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2023, pp. 62–76.

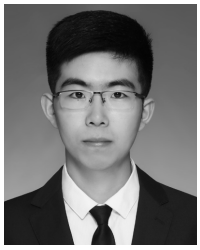
[12] X. Zhang, C. Hao, P. Zhou, A. Jones, and J. Hu, "H2H: Heterogeneous model to heterogeneous system mapping with computation and communication awareness," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 601–606.

[13] J. Kim et al., "Architecture, chip, and package codesign flow for interposer-based 2.5-D chiplet integration enabling heterogeneous IP reuse," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 11, pp. 2424–2437, Nov. 2020.

[14] S. Pal, D. Petrisko, R. Kumar, and P. Gupta, "Design space exploration for chiplet-assembly-based processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 1062–1073, Apr. 2020.

[15] F. Li et al., "GIA: A reusable general interposer architecture for agile chiplet integration," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.

[16] Z. Zhuang, B. Yu, K.-Y. Chao, and T.-Y. Ho, "Multi-package co-design for chiplet integration," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.
[17] I. H. Jiang, Y.-W. Chang, J.-L. Huang, and C.-P. Chen, "Intelligent design automation for 2.5/3D heterogeneous SoC integration," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–7.
[18] T. Burd et al., "'Zeppelin': An SoC for multichip architectures," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 133–143, Jan. 2019.
[19] M. Wang, Y. Wang, C. Liu, and L. Zhang, "Network-on-interposer design for agile neural-network processor chip customization," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Oct. 2021, pp. 49–54.
[20] Y. Feng, D. Xiang, and K. Ma, "A scalable methodology for designing efficient interconnection network of chiplets," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2023, pp. 1059–1071.
[21] B. Zimmer et al., "A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, Apr. 2020.
[22] X. Hao, Z. Ding, J. Yin, Y. Wang, and Y. Liang, "Monad: Towards cost-effective specialization for chiplet-based spatial accelerators," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2023, pp. 1–9.
[23] *NVDLA Deep Learning Accelerator*. Accessed: 2017. [Online]. Available: http://nvdla.org/
[24] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.
[25] S.-C. Kao and T. Krishna, "GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.
[26] Y. Lin, M. Yang, and S. Han, "NAAS: Neural accelerator architecture search," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1051–1056.
[27] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2019, pp. 754–768.
[28] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, "DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 1593–1597.
[29] A. Parashar et al., "Timeloop: A systematic approach to DNN accelerator evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2019, pp. 304–315.
[30] N. Jiang, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 86–96.
[31] *The Gem5 Simulator*. Accessed: 2011. [Online]. Available: https://www.gem5.org/
[32] Z. Tan, H. Cai, R. Dong, and K. Ma, "NN-baton: DNN workload orchestration and chiplet granularity exploration for multichip accelerators," in *Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1013–1026.
[33] J. Zhang et al., "INDM: Chiplet-based interconnect network and dataflow mapping for DNN accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 4, pp. 1107–1120, Apr. 2024.
[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[35] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517–6525.
[36] A. Dosovitskiy et al., "An image is worth 16 × 16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–21.
[37] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. 18th Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.*, vol. 9351, 2015, pp. 234–241.
[38] Y. Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.
[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, Jun. 2019, pp. 4171–4186.
[40] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*. New York, NY, USA: ACM, 2017, pp. 173–182.
[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.
[42] J. W. Poulton et al., "A 1.17-pJ/b, 25-Gb/s/pin ground-referenced single-ended serial link for off- and on-package communication using a process- and temperature-adaptive voltage regulator," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 43–54, Jan. 2019.
[43] D. Seo, A. Ali, W.-T. Lim, N. Rafique, and M. Thottethodi, "Near-optimal worst-case throughput routing for two-dimensional mesh networks," in *Proc. 32nd Int. Symp. Comput. Archit. (ISCA05)*, 2005, pp. 432–443.
[44] C. Carrion, R. Beivide, J. A. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proc. 4th Int. Conf. High-Perform. Comput.*, Dec. 1997, pp. 322–329.
[45] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. M. Prellezo, "The adaptive bubble router," *J. Parallel Distrib. Comput.*, vol. 61, no. 9, pp. 1180–1208, Sep. 2001, doi: 10.1006/JPDC.2001.1746.
[46] L. Chen, R. Wang, and T. M. Pinkston, "Critical bubble scheme: An efficient implementation of globally aware network flow control," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2011, pp. 592–603.

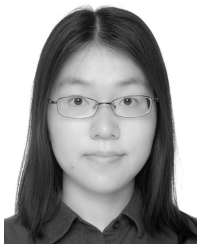


Jinming Zhang received the B.S. degree in microelectronics science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 2020, where he is currently working toward the Ph.D. degree.

His current research interests include network-on-chip and AI chip architecture design.

Xuyan Wang received the B.S. degree in microelectronics science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 2021. She is currently working toward the M.S. degree.

Her current research interests include chiplet architecture design and neural-network processing unit design.



Yaoyao Ye (Member, IEEE) received the B.S. degree from the University of Science and Technology of China, Hefei, Anhui, China, in 2008, and the Ph.D. degree from The Hong Kong University of Science and Technology, Hong Kong, in 2013.

She is currently an Associate Professor with the Department of Micro/Nano Electronics, Shanghai Jiao Tong University, Shanghai, China. Her research interests include multicore architecture, network-on-chip, and accelerator architecture.

Dr. Ye served on the Technical Program Committees of ASP-DAC 2024, DATE 2023, NOCS 2022-2023, NoCArc 2022-2023, ASP-DAC 2019-2021, ASP-DAC 2016-2017, and GLSVLSI 2017.

Dongxu Lyu received the B.S. degree in microelectronics science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 2020, where he is currently working toward the Ph.D. degree at the School of Electronic Information and Electrical Engineering.

His current research interests include intersection of autonomous driving systems and digital circuits and systems. The current focus is ASIC design for AI applications such as 3-D perception for autonomous driving.

Guojie Xiong received the B.S. and M.S. degrees in microelectronics science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 2020 and 2023, respectively.

His research interests include chiplet architecture and interface protocol.

Ningyi Xu received the B.S. and Ph.D. degrees from Tsinghua University, Beijing, China, in 2001 and 2006, respectively.

He was a Principle Architect with Baidu, Beijing, and a Lead Researcher with the Hardware Computing Group, Microsoft Research Asia, Beijing. He is currently a Professor with the Qingyuan Research Institute, Shanghai Jiao Tong University, Shanghai, China. His current research interests include domain-specific computing, computer architecture, parallel computing, and machine learning systems.

Yong Lian (Fellow, IEEE) received the B.S. degree from the College of Economics and Management, Shanghai Jiao Tong University, Shanghai, China, in 1984, and the Ph.D. degree from the Department of Electrical Engineering, National University of Singapore, Singapore, in 1994.

He has authored or co-authored more than 320 articles. His research interests include biomedical circuits and systems and signal processing.

Dr. Lian is a fellow of the Canadian Academy of Engineering and the Academy of Engineering Singapore. He was a recipient of many awards, including the IEEE Circuits and Systems (CAS) Society's Guillemin-Cauer Award in 1996, the IEEE Communications Society Multimedia Communications Best Paper Award in 2008, Singapore IES Prestigious Engineering Achievement Award in 2011, the CN Yang Award in Science and Technology for New Immigrant in 2014, and the Design Contest Award of 20th ISLPED in 2015. He is a Member-at-Large of IEEE PSPB, the Chair of the IEEE Periodicals Partnership Opportunities Committee, the IEEE Ad Hoc Committee on Accelerating Partnership with Chinese Publications, a member of the IEEE Periodicals Committee, the IEEE Periodicals Advisory and Review Committee, the IEEE Products and Services Committee, the IEEE PSPB Strategic Planning Committee, the IEEE PSPB Publishing Conduct Committee, and the IEEE Press Editorial Board. He was the President, the VP for Publications, and the VP Region ten of the IEEE Circuits and Systems Society. He is the Founder of the IEEE BioCAS Conference and the IEEE PrimeAsia Conference for postgraduate students. He was the Editor-in-Chief of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II, a member of the IEEE Fellow Committee, a member of the IEEE Biomedical Engineering Award Committee, and a member of the IEEE Medal for Innovation in Healthcare Technology Committee.

Guanghui He (Member, IEEE) received the B.S. degree in electronic engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2002, and the Ph.D. degree in electronic engineering from Tsinghua University, Beijing, China, in 2007.

He is currently a Professor with the Department of Micro/Nano Electronics and the MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China. His research interests include energy-efficient algorithms and circuits design for wireless communication, image processing, emerging memory devices, and artificial intelligent systems.