An Efficient Multi-DNN Accelerator Based on Multiple Systolic Arrays

Jianjun Chen, Han Jiao, Wenjin Huang, Yihua Huang
School of Electronics and Information Technology (School of Microelectronics)
Sun Yat-sen University
Guangzhou, China
{chenjj369, jiaoh3}@mail2.sysu.edu.cn, {huangwj96, huangyih}@mail.sysu.edu.cn

Abstract—With the rapid evolution of artificial intelligence (AI) technology, the scope of AI application has extended beyond individual Deep Neural Network (DNN) models, leading to a growing emphasis on Multi-DNN scenario. Google's TPU v1, a prominent DNN hardware accelerator, employs a systolic array as its core computing module, showcasing high data reuse and computational parallelism, thereby enabling efficient processing of DNN tasks. However, applying the fixed size of monolithic systolic array in Multi-DNN scenario leads to low hardware utilization due to the computational heterogeneity of DNN tasks, and it lacks the capability to concurrently execute multiple DNN tasks. To address these issues, we propose a multi-core accelerator, with each core designed based on systolic array. Additionally, we design a flexible Networks-on-Chip (NoC) specifically tailored for the systolic array, allowing combinations between systolic arrays to form a new systolic array with different sizes and shapes to adapt DNN tasks with varying computational characteristics. In addition, to fully leverage the composability of the multi-systolic array blocks, we design a DNN layer execution compiler and a core allocation compiler, both significantly improving performance. The experimental results show that our design significantly outperforms TPU-like architecture and DM-NPU, achieving an average reduction of 65.5% and 34.8% in the average normalized turnaround time (ANTT), and an average improvement of 104.3% and 30.1% in the system throughput (STP) under the multitasking scenario.

Index Terms—Multi-DNN Accelerators, Systolic Arrays, Hardware Accelerator

I. Introduction

In recent years, Deep Neural Networks(DNNs) have played a significant role in various artificial intelligence (AI) applications [1-7]. However, as tasks become increasingly complex, there is a growing demand for running multiple DNN workloads [10]. The escalating need for computational resources in AI systems has spurred research on DNN accelerators. Among these, the systolic array [8] stands out as a computing network comprised of homogeneous processing elements (PEs), which has high off-chip data reuse rate, parallelism and scalability [9]. However, a monolithic systolic array faces challenges in Multi-DNN scenario. It cannot simultaneously execute multiple DNN tasks, and due to the computational heterogeneity among different DNN tasks, a fixed-size systolic array struggles to efficiently adapt to convolutional operations or matrix multiplications of various sizes and types, leading to low hardware utilization, as shown in Figure 1. This example employs weight-stationary dataflow, where the weights are fixed [9]. Figure 1 shows that when using a monolithic systolic array for convolutional computation, if the output channels of the convolution kernel is less than the systolic array's columns, or if the product of kernel size and input channels is less than the systolic array's rows, it will result in a decrease in the process element (PE) utilization.

An effective solution is to employ multiple smaller systolic arrays that can be flexibly tiled together through a dedicated Networks-on-Chip (NoC). The specific fusion form is determined by the computational characteristics of different DNN tasks, as illustrated in Figure 2. In this example, four DNN tasks are processed in parallel, and the fusion configurations of the systolic arrays adopted by different DNN tasks exhibits distinct shapes and sizes. This approach effectively addresses the challenges of low hardware utilization and inability to process multiple DNN tasks in parallel using a monolithic systolic array in Multi-DNN scenario.

Although we improve hardware utilization through fine-grained systolic array blocks, due to the heterogeneity of different DNN tasks (e.g., layer count, per-layer input and output shapes) [11], it is important to properly map multiple systolic arrays to specific DNN tasks. Furthermore, given the limited number of systolic array blocks, how to allocate an appropriate number of systolic array blocks to different DNN tasks will significantly impact the overall system performance [10].

In summary, this paper makes the following contributions:

- A Multi-Systolic Array Accelerator: We design a multi-DNN accelerator based on multiple systolic arrays, incorporating a flexible NoC that enables the concatenation of multiple systolic array blocks into various shapes. This dedicated NoC for systolic arrays improves the performance of the accelerator in handling diverse DNN tasks, while maintaing a low resource overhead.
- A DNN Layer Execution Compiler for Dynamic

This work was supported by the National Natural Science Foundation of China under Grant 62276278 and GuangDong Basic and Applied Basic Research Foundation under Grant 2022A1515110006 and 2024A1515011259.

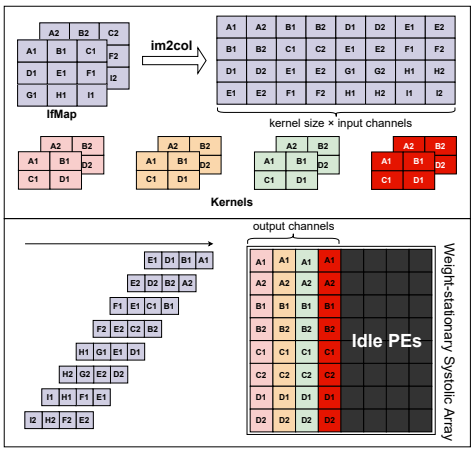

Fig. 1. Weight-stationary systolic array for convolution computation.

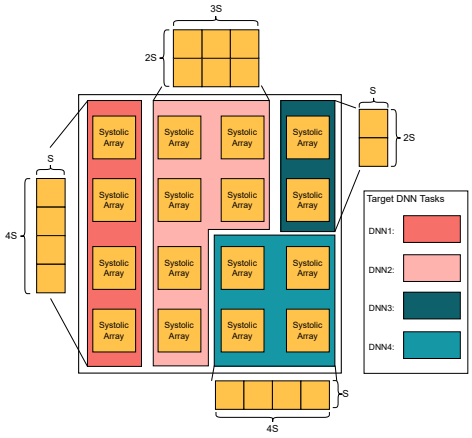

Fig. 2. Multi-DNN processing in a multi-systolic array system (assuming the size of a single systolic array is S×S).

Assembling of Multi-Systolic Arrays: We design a compiler that aims to maximize hardware utilization by exploring the optimal tiling configuration of systolic arrays for DNN layers. It will generates tiling configuration into a config table, allowing the CPU to perform table lookups based on DNN layers, thus generating systolic arrays of different shapes.

- A Core Allocation Compiler Oriented for Multi-Systolic Arrays: We design a compiler to dertermine the number of systolic array blocks that should be occupied by different DNN tasks. This compiler can generate the optimal core allocation solution based on the target performance, such as minimizing average normalized turnaround time (ANTT) or maximizing system throughput (STP).

II. Related Work

The increasing demand for accelerating execution in Multi-DNN scenario has propelled the swift evolution of Multi-DNN accelerators. PREMA [19], utilizing TPU [9] as its computational core, processes multiple DNN tasks through time-multiplexing shared computational resources. Additionally, PREMA presents a scheduling algorithm for preemptive execution of DNN tasks, ensuring adherence to the latency requirement of high-priority tasks. However, due to its reliance on a monolithic systolic array, it is not suitable for Multi-DNN scenario, as discussed earlier.

Therefore, Planaria [11] introduces a dynamic fission architecture, breaking down a large-scale systolic array into multiple small-scale systolic arrays. Each PE supports omni-directional data movement, and local crossbar units facilitate the fusion of systolic arrays, addressing the issues present in a monolithic systolic array. However, this design incurs significant overhead and lacks an efficient DNN mapping strategy to fully utilize the architectural flexibility. STfusion[12] utilizes multiple systolic arrays with clustered mapping, enabling weight and input feature sharing among multiple systolic arrays but lacks support for the transmission of the intermediate computational results generated by different systolic arrays, limiting the flexibility of the integration of multiple systolic arrays. DM-NPU[13] proposes dataflow-mirroring, a finer-grained segmentation of the systolic array compared to Planaria's coarse-grained approach, achieving higher hardware utilization. It also designs a software scheduler to allocate optimal resources based on DNN tasks. However, fine-grained segmentation incurs high overhead, limiting DM-NPU to supporting the division of a systolic array into at most four systolic arrays of different sizes. This results in a decrease in the number of parallelizable DNN tasks. Additionally, once a fixed resource allocation is made for a DNN task, all layers within that DNN must utilize the same-shaped systolic array, inevitably compromising flexibility.

Although the aforementioned work improves performance in Multi-DNN scenario through multiple fine-grained processing units, they incur substantial hardware overhead. The restricted collaboration among processing units impedes their adaptability to the heterogeneous computing characteristics of diverse DNN tasks in Multi-DNN scenario. Furthermore, while the underlying architecture boasts flexibility, the lack of an effective mapping strategy for DNN tasks prevents the complete realization of the performance potential inherent in this flexibility.

III. Multi-Core Architecture Design

The designed accelerator architecture is shown in Figure 3. The underlying hardware computation section comprises Cores, global buffer clusters (GLB Clusters), Routers, and their respective controllers. The Cores are responsible for the computation of DNN tasks, and each Core is equipped with a corresponding GLB Cluster and Router. The GLB Cluster serves to cache data retrieved from external DDR as well as intermediate results generated during computations. The Router facilitates data transmission among the Cores, and different forms of

systolic array fusion can be achieved by configuring the Router Controller during runtime.

The upper-level CPU controls the scheduling of all DNN tasks, the allocation of the number of cores to them, and the mapping of different DNN tasks onto the cores. Through the AXI interface, The CPU possesses the capability to config various controllers of the underlying hardware, inclusive of the DDR controller. The DDR controller facilitates the transfer of data from the DDR to the GLB Clusters via the Crossbar.

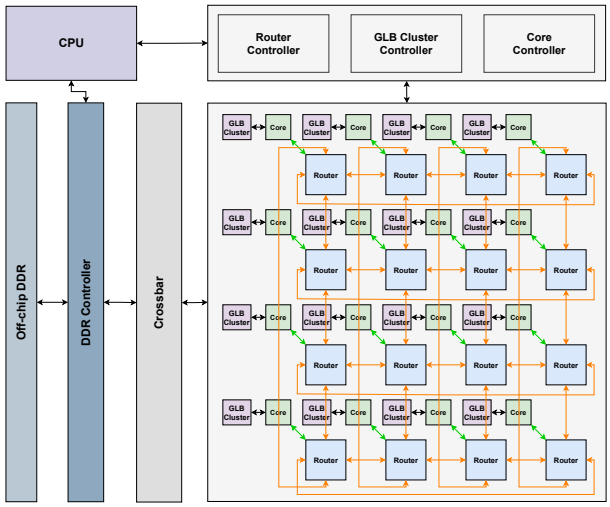

Fig. 3. Proposed accelerator architecture (an example of 4 × 4 cores).

A. Proposed Core Architecture

The architecture of the Core, as depicted in Figure 4, consists of a controller, multiple FIFOs, multiple MUXes, a systolic array, and a SIMD vector unit. Core works in independent mode or multi-core fusion mode, with each mode having different data source. In independent mode, data is selected from the GLB Cluster, while in multi-core fusion mode, data is sourced from the Router. In multi-core fusion mode, due to the varying positions of cores in the NoC, there is inconsistent data communication latency between different cores. To address this issue, FIFO buffers are strategically employed within core. The core does not need to care about when the data arrives, but only starts computation when all FIFOs contain data, which has less hardware overhead than inserting registers to balance communication delays. In this paper, we employ a systolic array with weight-stationary dataflow, which has the highest weight reuse rate and good versatility [9].

B. Flexible NoC Design

The NoC comprises Routers and Cores, and data flows between Routers to achieve the fusion of different cores. In order not to destroy the advantages brought by the regularity of the systolic array, the designed NoC retains the overall structure of the systolic array. At the same time, we introduce ring between Routers to enrich routing

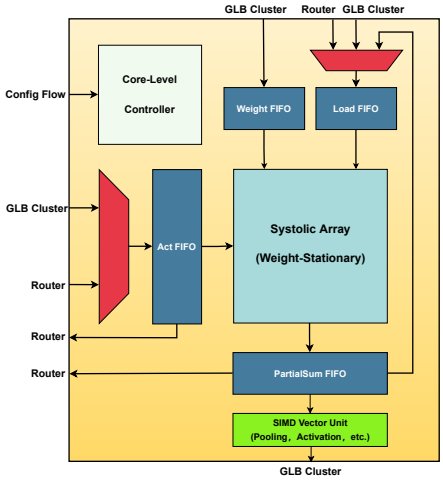

Fig. 4. Proposed core architecture.

channels and improve the NoC flexibility, as shown in Figure 3. This lays the foundation for the task mapping proposed in the next section.

Since the NoC is specifically tailored for the weight-stationary systolic array, where the weights are stationary inside the PE during runtime, the data flowing between the systolic arrays includes only input feature maps and partial sums. Consequently, we only need to focus the transfer of these two types of data, as depicted in Figure 5. Horizontally, systolic arrays aligned along the same axis share input feature maps, showcasing a horizontal fusion of two systolic arrays. Vertically, the systolic array below needs to receive partial sums generated by the systolic array above, resembling a vertical fusion of two systolic arrays. The detailed design of the Router is depicted in Figure 6, where "Act" represents input feature map, "Ps" denotes partial sum, and "U", "D", "L", "R" respectively represent data from the upper, down, left and right of the Router. During runtime, the Router receives data from its associated core and neighboring Routers. By configuring the Router Controller, the Router transfers corresponding neighboring Routers'data to the core and transfers core's data or data from the other three Routers to another Router. Since the two systolic array blocks only transmit input feature maps or partial sums, never simultaneously, the Router employs a shared bus for both, thereby significantly reducing design overhead without performance degradation.

IV. Task Mapping And Core Allocation

In data centers, there are often multiple concurrent DNN tasks, which are typically heterogeneous [10]. Applying the same number of PEs but with different shaped systolic arrays to the same DNN task can result in significant performance differences [11]. Therefore, it is crucial to specifically map DNN tasks under a specific number of systolic arrays. Additionally, since multiple

heterogeneous DNN tasks are executed in parallel, the allocation of core numbers to specific DNN tasks also needs to be explored. This section will discuss the task mapping and core allocation in detail.

A. Task Mapping

The majority of operations in DNN occur in the convolutional layers [27], therefore our discussion on task mapping focuses on the mapping of convolution on the systolic array. The convolution operation actually involves block-wise processing [15], the granularity of the blocks depends on the size of a single systolic array. Blocks in the same row share input feature map, while blocks in the same column propagate the partial sum, which facilitates the fusion of multiple systolic arrays.

To maximize hardware utilization by identifying the optimal systolic array fusion form for different layers given the available number of cores, we design a DNN layer execution compiler tailored for multiple concatenable systolic array blocks, as depicted in Figure 7. The compiler is layer-level. After the DNN layers are divided into blocks, the compiler will enumerate all possible shapes of systolic arrays and predict the execution latency based on the parameters of DNN layer (such as the number of input/output channels, the size of the input feature map, etc.), ultimately aiming to derive the optimal solution with the minimum latency, thus maximizing hardware utilization. Although the compiler adopts an exhaustive approach, it can significantly reduce the search space by eliminating some impossible solutions, such as cross-block executions that do not fully utilize data reuse.

After attaining the optimal solution, the compiler will generate a configuration table, which will be used by the CPU to lookup and configure the Router Controller for the specific tiling patterns of multiple systolic arrays. This table records the tiling shape for each execution step of the layer, along with the precedence relationship of the systolic array tiling and specfies the DDR data transfer to a particular GLB cluster. For instance, in step 2 depicted in Figure 7, Core B, C, and D will be horizontally tiled, with the input feature map data flowing from Core B through C to D. Therefore, the off-chip DDR data is stored in the GLB cluster corresponding to Core B. Meanwhile,

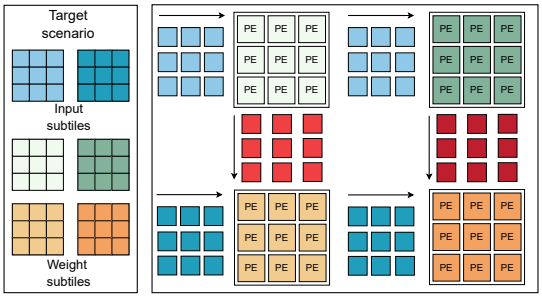

Fig. 5. Multi-systolic array fusion.

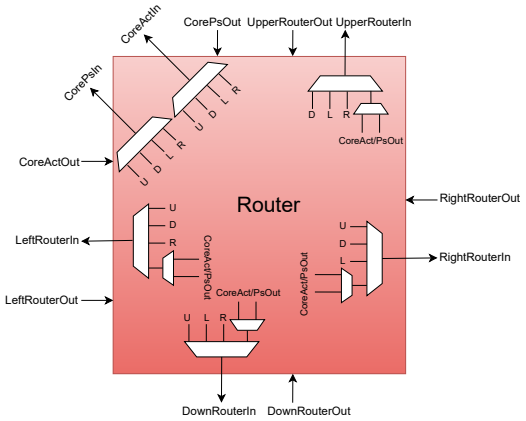

Fig. 6. Proposed router architecture.

Core A and Core B will be vertically tiled, with the results generated by Core A flowing downward into Core B, ultimately forming an 'L'-shaped configuration. The order of concatenation in each step is important, enabling the reuse of input feature maps across various execution steps. Since the structural parameters of most layers in DNNs are repetitive [21-28], the corresponding optimal solutions are identical, making the storage space occupied by the Config Table generated by compiler acceptable.

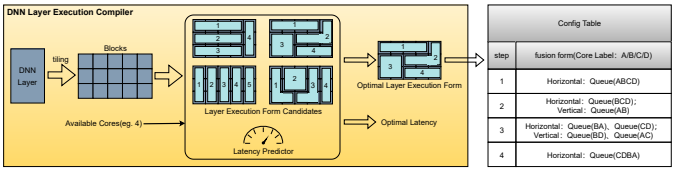

Fig. 7. DNN Layer Execution Compiler (The number inside the blocks represent the order of execution).

B. Core Allocation

Given a set of co-located DNN tasks, we should identify the optimal core allocation to maximize the target performance. We choose two widely-used multitasking performance metrics: ANTT and STP [29]. ANTT quantifies user-perceived performance, specifically measuring the slowdowns in turnaround time under multitasking scenarios. A lower value indicates that the turnaround time is less affected by multitasking. STP quantifies system-perceived performance, specifically measuring the number of DNN tasks completed per unit of time. A higher value indicates a higher system throughput.

To achieve the optimal performance, we design a core allocation compiler that generates the optimal resource allocation for each DNN task based on the optimization objectives, DNN tasks, the optimal latency of DNN tasks under different core counts (derived from the DNN Layer Execution Compiler) and the total number of available cores, as shown in Figure 8. The core allocation compiler

will enumerate all possible solutions and compare them based on ANTT or STP to obtain the optimal core allocation solution. This compiler demonstrates significant flexibility, enabling it to generate diverse solutions based on different needs. For instance, it can assign varying degrees of importance to ANTT and STP to derive the optimal core allocation solution accordingly. Since we are targeting datacenter applications, where multiple tasks are deployed and continuously executed over the long term [16], the compilation time is not a primary concern.

Once each DNN task is allocated the corresponding number of cores, all layers of that DNN task will utilize the currently allocated cores, meaning that each core will always correspond to different layers of the same DNN task. This significantly reduces complexity and makes it easier to deploy the systolic array fusion configurations generated by the DNN layer execution compiler onto the hardware architecture. Despite using a fixed number of systolic arrays, the DNN task can employ varying shapes of systolic arrays for different layers, demonstrating high flexibility.

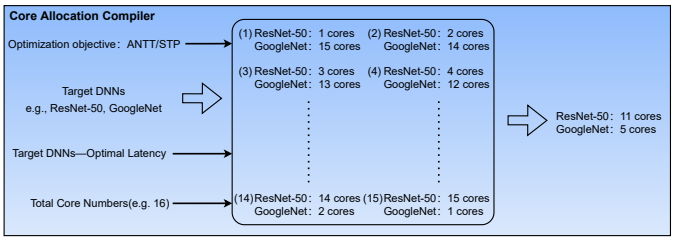

Fig. 8. Core Allocation Compiler

V. Evaluation

A. Experimental Setup

We employ task scenarios that include image classification and object detection, and select 8 different DNN workloads. The selected DNN workloads are commonly used in industry and have different computational characteristics (i.e. layer type and size), as shown in Table 1. We conduct experiments by simulating the working scenario in the data center through multithreading, maintaining the number of concurrently running DNN tasks on the accelerator [12]. We define this scenario as N-threaded, represented as NT (e.g., 8-threaded is 8T). In each threading scenario, the DNN tasks to be executed are randomly selected from Table 1, and 1000 sets of experiments are conducted to ensure that each DNN task is selected. We adopt ANTT and STP as performance metrics [29], with the final results presented as the average values of both metrics. For a fair comparison, the execution time under single-thread in ANTT and STP is benchmarked against a TPU-like architecture [19].

We implement the proposed architecture in Verilog and employ the Xilinx Alevo U250 accelerator card as the hardware platform to assess, as shown in Table 2. The

TABLE I
Workload Setting

Domain	DNN Workload
Image Classification	ResNet50[22]
	VGGNet16[25]
	GoogleNet[24]
	DenseNet121[23]
	DenseNet201[23]
Object Detection	SSD-ResNet34[22]
	Yolo-v3[26]
	SSD-MobileNet[28]

TABLE II
Hardware Configuration

Hardware Unit	Size
Processing Element Dimension	32×32
# of Cores	16
Frequency	250 MHz
LUT	1539771(89.11%)
FF	610076(17.65%)
URAM	768(60%)
BRAM	4480(60%)
DSP	10432(84.9%)

number of cores is 4×4, and the PE quantity of a single core is 32×32 to benchmark the size of PREMA [19] and DM-NPU [13] for a fair comparison. The NoC hardware overhead is registers and LUTs, accounting for about 10.9% and 5.7% of the total respectively. We also model PREMA and DM-NPU, where PREMA uses TPU as its hardware accelerator, essentially consisting of a 128×128 systolic array [19]. In contrast, DM-NPU enables a large systolic array to be isolated horizontally and vertically, forming up to four different-sized systolic array blocks.

B. User-Perceived and System-Perceived Performance

We conduct experiments to evaluate the performance of proposed architecture against the TPU-like architecture [19] and DM-NPU [13], focusing on ANTT and STP metrics. A lower ANTT is better, and a higher STP is better. The experimental results, presented in Figure 9, demonstrate that our design achieves an average reduction of 65.5% and 34.8% in ANTT compared to the TPU-like architecture and DM-NPU, respectively, and an average STP improvement of 104.3% and 30.1%, respectively.

Compared to TPU-based architecture, our design divides the systolic arrays into multiple smaller blocks, resulting in finer processing granularity. Additionally, these systolic array blocks can be concatenated through the NoC based on the computational characteristics of various DNN tasks, leading to higher hardware utilization. In contrast to DM-NPU, our architecture allows for the switching of concatenation forms across different DNN layers, while DM-NPU is constrained to a single concatenation form for all layers of DNN tasks due to its complexity, making our approach more fine-grained in the processing of DNN tasks. Furthermore, due to the complexity of its architecture, DM-NPU is incapable of concurrently executing

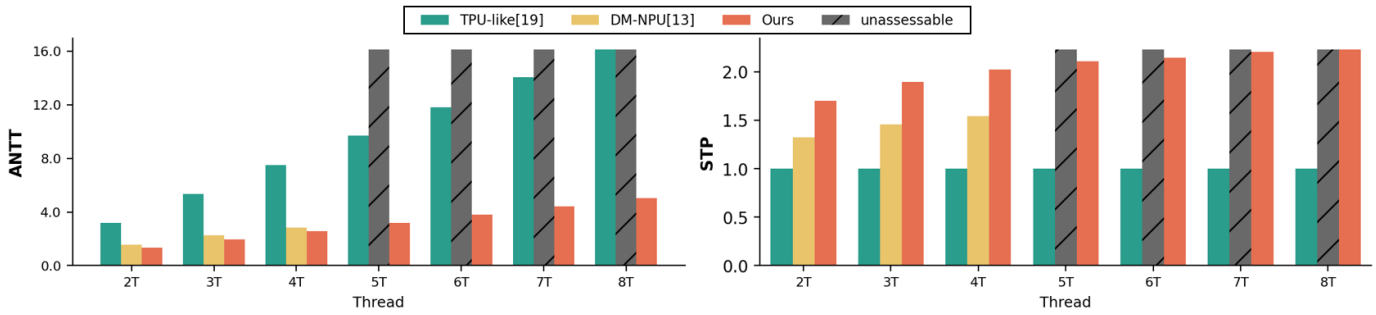

Fig. 9. ANTT and STP of proposed architecture compared to others with the multitasking benchmarks. Lower ANTT and higher STP are better.

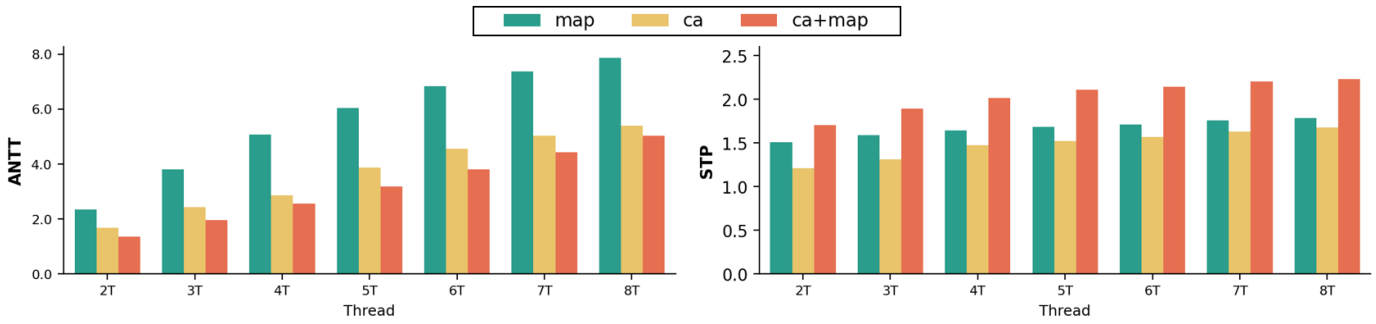

Fig. 10. Ablation study on the proposed DNN layer execution compiler and core allocation compiler.

more than four DNN tasks, and therefore relies on time-sharing multiplexing. This limitation prevents us from assessing DM-NPU's performance in terms of ANTT and STP under 5T to 8T, but we can confidently assert that our design offers superior performance compared to DM-NPU.

On the other hand, based on our designed flexible multi-systolic array architecture, we further propose two compilers, one for maximizing the hardware utilization of DNN layers during execution, and another for achieving optimal core allocation in terms of ANTT or STP. These compilers further enable our architecture to achieve lower ANTT and higher STP.

C. Ablation Study

To evaluate the impact of the two proposed compilers on performance, we conducted ablation experiments. In these experiments, "map" represents the use of the DNN layer execution compiler, where the number of cores allocated to each DNN task is random. Meanwhile, "ca" represents the use of the core allocation compiler, which assigns an optimal number of cores to each DNN task but employs a "square" tiling approach, namely, a regular systolic array. The experimental results demonstrate that both compilers further improve performance, as shown in Figure 10.

In the DNN layer execution compiler, it is capable of assembling systolic arrays of different shapes based on the computational characteristics of various DNN layers given the number of cores. Even during the execution of a single DNN layer, the tiling configuration of multiple systolic array blocks can vary to maximize hardware utilization. In the core allocation compiler, we employ an exhaustive search approach to obtain the optimal solution for the number of cores allocated to each DNN task. The number of cores determines the amount of hardware resources available for the DNN task, significantly influencing its runtime. Therefore, we can reasonably allocate cores based on different requirements, such as minimizing ANTT or maximizing STP.

VI. Conclusion

In this paper, we design an efficient multi-DNN accelerator based on multiple systolic arrays. Each systolic array is interconnected through NoC, capable of assembling into systolic arrays of various sizes and shapes. To fully leverage the flexibility of the multi-systolic array architecture, we design a DNN layer execution compiler that generates specific systolic array tiling forms for different DNN layer executions, thus maximizing hardware utilization. Furthermore, we design a core allocation compiler to optimize core allocation for each DNN task, minimizing ANTT or maximizing STP. Experimental results show that our design achieves significant improvements in both ANTT and STP performance compared to TPU-like accelerators and DM-NPU. Additionally, ablation experiments demonstrate that the two compilers further enhance the performance of the multi-systolic array architecture.

References

[1] Girshick, R.B., et al., Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014: p. 580-587.

[2] Collobert, R., et al., Natural Language Processing (Almost) from Scratch. J. Mach. Learn. Res., 2011. 12: p. 2493-2537.

[3] Li, H., et al., A convolutional neural network cascade for face detection. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015: p. 5325-5334.

[4] Krizhevsky, A., I. Sutskever, and G.E. Hinton, ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2012. 60: p. 84 - 90.

[5] Hinton, G.E., et al., Deep Neural Networks for Acoustic Modeling in Speech Recognition. IEEE Signal Processing Magazine, 2012. 29: p. 82.

[6] Deng, L., et al., Recent advances in deep learning for speech research at Microsoft. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013: p. 8604-8608.

[7] Chen, C., et al., DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. 2015 IEEE International Conference on Computer Vision (ICCV), 2015: p. 2722-2730.

[8] Kung, "Why systolic architectures?," in Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

[9] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 2017, pp. 1-12, doi: 10.1145/3079856.3080246.

[10] S. I. Venieris, C. -S. Bouganis and N. D. Lane, "Multiple-Deep Neural Network Accelerators for Next-Generation Artificial Intelligence Systems," in Computer, vol. 56, no. 3, pp. 70-79, March 2023, doi: 10.1109/MC.2022.3176845.

[11] S. Ghodrati et al., "Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, Greece, 2020, pp. 681-697, doi: 10.1109/MICRO50266.2020.00062.

[12] E. Baek, E. Lee, T. Kang and J. Kim, "STfusion: Fast and Flexible Multi-NN Execution Using Spatio-Temporal Block Fusion and Memory Management," in IEEE Transactions on Computers, vol. 72, no. 4, pp. 1194-1207, 1 April 2023, doi: 10.1109/TC.2022.3218428.

[13] J. Choi et al., "Enabling Fine-Grained Spatial Multitasking on Systolic-Array NPUs Using Dataflow Mirroring," in IEEE Transactions on Computers, vol. 72, no. 12, pp. 3383-3398, Dec. 2023, doi: 10.1109/TC.2023.3299030.

[14] V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[15] E. Baek, D. Kwon and J. Kim, "A Multi-Neural Network Acceleration Architecture," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020, pp. 940-953, doi: 10.1109/ISCA45697.2020.00081.

[16] S. Zeng et al., "Serving Multi-DNN Workloads on FPGAs: A Coordinated Architecture, Scheduling, and Mapping Perspective," in IEEE Transactions on Computers, vol. 72, no. 5, pp. 1314-1328, 1 May 2023, doi: 10.1109/TC.2022.3214113.

[17] S. I. Venieris and C. -S. Bouganis, "f-CNNx: A Toolflow for Mapping Multiple Convolutional Neural Networks on FPGAs," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 2018, pp. 381-3817, doi: 10.1109/FPL.2018.00072.

[18] S. -C. Kao and T. Krishna, "MAGMA: An Optimization Framework for Mapping Multiple DNNs on Multiple Accelerator Cores," 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Korea, Republic of, 2022, pp. 814-830, doi: 10.1109/HPCA53966.2022.00065.

[19] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-Task Scheduling Algorithm For Preemptible Neural Processing Units," 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), San Diego, CA, USA, 2020, pp. 220-233, doi: 10.1109/HPCA47549.2020.00027.

[20] S. Selvam, V. Ganesan and P. Kumar, "FuSeConv: Fully Separable Convolutions for Fast Inference on Systolic Arrays," 2021 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, 2021, pp. 651-656, doi: 10.23919/DATE51398.2021.9473985.

[21] V. J. Reddi et al., "MLPerf Inference Benchmark," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020, pp. 446-459, doi: 10.1109/ISCA45697.2020.00045.

[22] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[23] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.

[24] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[25] Simonyan K , Zisserman A .Very Deep Convolutional Networks for Large-Scale Image Recognition[J].Computer Science, 2014.DOI:10.48550/arXiv.1409.1556.

[26] Redmon J , Farhadi A .YOLOv3: An Incremental Improvement[J].arXiv e-prints, 2018.DOI:10.48550/arXiv.1804.02767.

[27] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu and S. Wei, "Deep Convolutional Neural Network Architecture With Reconfigurable Computation Patterns," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 8, pp. 2220-2233, Aug. 2017, doi: 10.1109/TVLSI.2017.2688340.

[28] Howard A G , Zhu M , Chen B ,et al.MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications[J]. 2017.DOI:10.48550/arXiv.1704.04861.

[29] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multiprogram Workloads," in IEEE Micro, vol. 28, no. 3, pp. 42-53, May-June 2008, doi: 10.1109/MM.2008.44.