Memory and Computation Coordinated Mapping of DNNs onto Complex Heterogeneous SoC

Size Zheng
Peking University
zhengsz@pku.edu.cn

Siyuan Chen
Peking University
chensiyuan@pku.edu.cn

Yun Liang*
Peking University & Beijing Advanced Innovation Center for Integrated Circuits
ericlyun@pku.edu.cn

***Abstract*—The DNN models are now pervasively used for various applications. Meanwhile, the computing hardware has shifted towards heterogeneous system composed of various accelerators. The intertwined complexity of DNN models and hardware makes it challenging for mapping DNN models. Existing mapping frameworks suffer from inefficiencies due to under utilization of computation and bandwidth in heterogeneous SoC. In this paper, we propose COMB, a mapping framework that coordinates the memory and computation and data transfer overhead of heterogeneous accelerators to achieve latency improvement and energy efficiency with two optimizations: dataflow grouping and accelerator mapping. Dataflow grouping maps multiple independent DNN layers to the same accelerator at the same time to spatially share the hardware resources; accelerator mapping finds the optimized placement of the layer groups to accelerators to reduce data transfer overhead. These two optimizations provide a huge design space for heterogeneous DNN mapping. To explore the space efficiently, we present a hybrid scheduling algorithm by combining greedy algorithm and genetic algorithm. In evaluation, COMB achieves $1.28\times$ and $1.37\times$ speedup for latency compared to MAGMA and H2H; COMB also reduces 22.7% and 29.2% energy consumption compared to MAGMA and H2H.**

***Index Terms*—Heterogeneous SoC, DNN, Mapping, Genetic Algorithm**

I. Introduction

The advancement in multi-modality and multi-task learning brings various heterogeneous DNN model architectures that are composed of multiple different types of computations (e.g., convolution layers, fully connected layers, and transformer layers) as shown in Figure 1 part a). The heterogeneous layers prefer different hardware accelerator designs for high performance [1]–[3]. To provide low latency and high energy efficiency, various heterogeneous SoC designs [1], [4] have been proposed recently. The heterogeneous SoC is normally composed of multiple different types of accelerators that are connected to each other spatially through network-on-chip (NoC) as shown in Figure 1 part b). The major features of such heterogeneous SoC can be summarized into two aspects. For computation, the accelerators in the heterogeneous system employ different dataflows [2] (e.g., weight-stationary, output-stationary) and different hardware resources to accelerate different workloads. For memory, each accelerator occupies a local scratchpad memory to store a tile of input/output data for computation. The scratchpad is limited in size and the accelerators have to communicate with each other to get the required data through NoC. The accelerators that are close to each other need less delay (NoC hops) for data transmission compared to those that are farther away from each other.

Mapping heterogeneous DNNs to heterogeneous SoCs is challenging. It requires the mapping to fully utilize the hardware computation resources and communication bandwidth to achieve high efficiency. First, for computation, the DNN accelerators are over-customized for special layer types and input shapes. Mapping each layer to its most suitable accelerator without consideration of the whole SoC utilization may cause congestion on certain accelerators and increase the overall latency. Second, for data transfer overhead, improper spatial placement of DNN layers may bring heavy data transfer overhead. In Figure 1 part c), we show examples for such inefficient mappings. On the left side, it maps too many workloads to accelerator 1 and leaves other accelerators (acc2) idle. On the right side, the mapping requires a large amount of data transfer along a long path (from left-top to right-bottom), causing a long data transfer delay.

*Corresponding author.

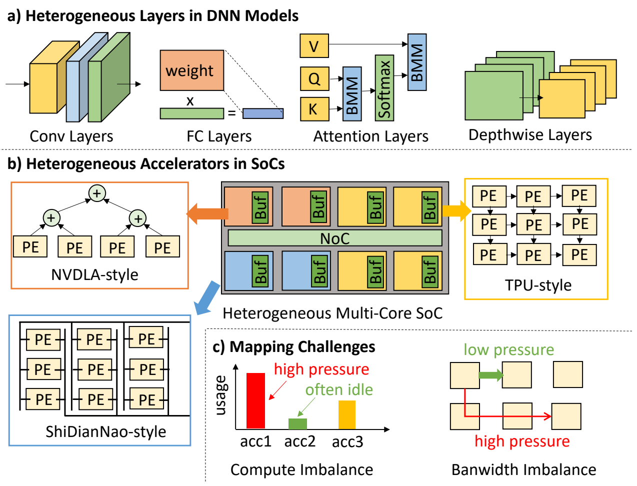


Fig. 1: Part a): Heterogeneous DNN layers. Part b): Heterogeneous DNN accelerators. Part c): Imbalance problem in mapping.

Existing works [1], [5], [6], [8], [9], [11] only partially address the heterogeneous DNN mapping problem. PREMA [8] proposes to use preemptible NPUs and predictive schedulers to map DNNs to a large systolic array. It improves task latency for only one particular dataflow and one single accelerator. AI-MT [9] packs compute-intensive and memory-intensive workloads together to fully utilize hardware compute and memory resources, but they assume all the accelerators are homogeneous without consideration of heterogeneous dataflow. Herald [1] proposes to use heterogeneous dataflows for heterogeneous DNNs. It prioritizes computation mapping without modeling the inter-accelerator communication overhead. Recent state-of-the-art works MAGMA [6] and H2H [5] take data transfer overhead into consideration and optimize the mapping problem for both homogeneous and heterogeneous accelerators. However, their mappings assume that a single DNN layer is enough to fully utilize the computation resources of one accelerator and that all the accelerators can communicate with each other uniformly, which is not accurate for complex heterogeneous SoCs. As a result, their mappings may result in sub-optimal performance.

In this paper, we propose COMB, a mapping framework that coordinates the computation utilization and data transfer overhead between accelerators to achieve better latency and energy efficiency for complex heterogeneous SoCs. COMB is equipped with two optimizations: dataflow grouping and accelerator mapping. Dataflow grouping can pack two or more independent DNN layers together into one task and execute them concurrently on one accelerator to improve computation utilization. The grouping process doesn't aim to choose the best dataflow for each DNN layer but aims to achieve better utilization for the whole SoC. Accelerator mapping is to choose the optimized placement of layers onto accelerators so that data transfer overhead between accelerators can be reduced. Intuitively, layers that have a large amount of data transfer should be placed in adjacent accelerators to exploit the high bandwidth between neighbor accelerators. These two optimizations are tightly coupled with each other and should be applied together in mapping. The design space is huge and can't be explored efficiently through brute-force enumeration. As a result, we combine greedy algorithms with genetic algorithms to explore the design space. In summary, we make the following contributions:

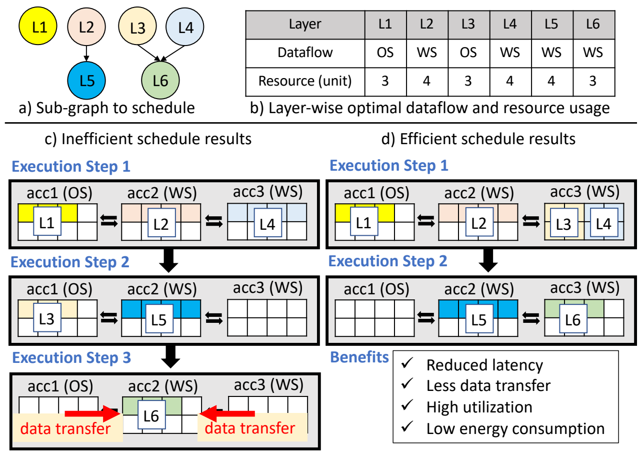
1) We formulate the mapping problem for complex heterogeneous SoCs with consideration of resource utilization and spatial non-uniform data transfer overhead.
2) We propose two optimizations: dataflow grouping and accelerator mapping to optimize the mapping for heterogeneous SoCs. We also combine greedy and genetic algorithms to explore the design space of these two optimizations.
3) We implement a mapping framework COMB for complex heterogeneous SoCs and achieve better latency and energy efficiency compared to state-of-the-art works.

In evaluation, COMB achieves 1.28× and 1.37× speedup for latency compared to MAGMA and H2H; COMB also reduces 22.7% and 29.2% energy consumption compared to MAGMA and H2H.

II. Motivational Example

In this section, we use a simple example in Figure 2 to motivate our work. For hardware part, we consider a heterogeneous SoC with three accelerators (acc1, acc2, acc3). They are designed to support different dataflows. Acc1 supports output stationary dataflow (OS); acc2 and acc3 support weight stationary dataflow (WS). In the SoC, acc1 is connected with acc2, and acc2 is connected with acc3. For the software part, we consider a sub-graph as shown in Figure 2 part a). The layers (L1-L6) in this sub-graph are heterogeneous and prefer different dataflows. We show the characteristics of these layers in the Table in Figure 2 part b). These layers require different units of hardware resources, which are also shown in the Table.

To map the heterogeneous sub-graph to the heterogeneous SoC, we show the mapping comparison of existing works [5], [6] (part c) and COMB (part d) in Figure 2. The mapping in part c) requires three execution steps and two cross-accelerator data transfers. In the first step, L1, L2, and L4 are mapped to their preferred accelerators. At the second step, L3 is mapped to acc1, L5 is mapped to acc2. The outputs of L2 reside in acc2, so L5 can get the required data without cross accelerator data transfer. L6 is not mapped because it relies on the outputs of L3. At the third step, L6 is mapped to acc2. The outputs of L4 reside in acc3 and the outputs of L3 are stored in acc1. So we need two data transfer operations for L6. The computation utilization is low because this mapping fail to share the same accelerator for independent layers (e.g., L3 and L4).



Layer	L1	L2	L3	L4	L5	L6
Dataflow	OS	WS	OS	WS	WS	WS
Resource (unit)	3	4	3	4	4	3

Fig. 2: Motivational example.

In Figure 2 part d), we show our mapping for the same example. In the first step, it maps L3 to a sub-optimal dataflow accelerator (acc3). Although mapping L3 to acc3 can cause latency increase and resource usage increase (e.g., resource usage increases from 3 units to 4 units), this mapping enables the concurrent execution of two layers (L3 and L4) on the same accelerator (acc3) and force them to use the same dataflow (**dataflow grouping**). In the second step, it maps L6 to acc3 so that L6 can use the outputs of L3 and L4 within the same accelerator, eliminating cross-accelerator data transfer (**accelerator mapping**). As a result, both latency and energy efficiency are improved by our mapping.

In the following sections, we explain how to systematically apply these two optimizations and find better mappings for heterogeneous SoCs.

III. Problem Formulation in COMB

In this section, we formulate the scheduling problem for complex heterogeneous SoCs. Compared to the formulations in previous works [5], [6], our formulation focuses on the spatial resource sharing and non-uniform data transfer overhead of heterogeneous SoCs.

A. Graph Representations for DNNs and SoCs

First of all, we introduce the graph representations for heterogeneous DNNs and SoCs. The computation of one DNN can be represented as a directed graph $G = (V, E)$, where V is composed of all the layers in the DNN, and E is composed of all the tensors in the DNN (one tensor can be treated as an edge between producer layer and consumer layer). For heterogeneous DNNs, we have multiple directed graphs $\mathcal{G} = (G_1, G_2, ..., G_M)$, where M is the number of DNNs. Similarly, a heterogeneous SoC can be represented as a directed graph $H = (A, Net)$, where A is the set of accelerators $A = \{\mathrm{acc}_1, \mathrm{acc}_2, ..., \mathrm{acc}_N\}$, N is the number of accelerators; $Net = \{(\mathrm{acc}_i, \mathrm{acc}_j, cost)|1 \le i, j \le N\}$ represents the NoC for accelerators. Each entry in Net records the data transfer cost between two accelerators. We also define several methods in Table I to calculate different properties of the multi-DNN graph and SoC graph.

B. Dataflow Grouping

Based on the graph representations of heterogeneous DNNs and SoCs, we present the formulation of our first optimization: dataflow grouping. Each layer in DNN graph G can be deployed to different

TABLE I: Methods to access DNN and SoC properties.

Name	Explanation
DNN Related Methods	
Pred(L)	Get the predecessor layers of layer $L \in V$
DV(L)	Data transfer volume for outputs of layer L
GroupOf(L)	Get the dataflow group of layer L
SoC Related Methods	
NumPE(acc)	Get the number of PEs of accelerator acc
MemCap(acc)	Get the scratchpad capacity of accelerator acc
Dataflow(acc)	Get the dataflow of acceleraotr acc
Comm(acc$_1$, acc$_2$, V)	The cost of transferring data of volume V from acc$_1$ to acc$_2$ according to Net

accelerators acc $\in A$ with different dataflows. However, as shown in the example in Figure 2, mapping some layers to sub-optimal dataflows may improve the global utilization, indicating that there exists a larger design space for dataflow mapping for the whole DNN graph. Formally, for multi-DNN graph $\mathcal{G} = (G_1, ..., G_M)$, a dataflow grouping choice can be represented as an ordered list of layer groups:

$$\begin{aligned} &D_1 \preceq D_2 \preceq \cdots \preceq D_K \quad \text{ where } \; D_i = \{L^i_1, L^i_2, \dots L^i_{P_i}\} \\ &D_i \cap D_{i'} = \emptyset \;\; \forall i \neq i', (D_1 \cup ... \cup D_K) = (V_1 \cup ... \cup V_M) \\ &L^i_j \in (V_1 \cup ... \cup V_M) \;\; 1 \leq i \leq K, \; 1 \leq j \leq P_i \end{aligned} \tag{1}$$

D_i represents a group of layers. The notation $D_i \preceq D_j$ indicates that all the layers in D_i don't depend on the results of any layer in D_j. But D_j may depend on some layers in D_i. So the group of layers D_i can be executed no later than D_j. P_i is the number of layers in D_i. All the layers in one group D_i are to be mapped to the same accelerator with the same dataflow and they are expected to spatially share the hardware resources of the same accelerator.

For a multi-DNN graph $\mathcal{G}$, there exist different dataflow grouping choices and the design space is huge (in the worst case, the space size is of $O(X!)$, where X is the total number of layers in $\mathcal{G}$). In Figure 3 we show an example of mapping a multi-graph to heterogeneous SoC and also show two different choices of dataflow grouping in part b). Each box in part b) represents a layer group. Even for the small multi-graph in part a), there exist various different grouping choices. Selecting the optimal one is hard and can't be solved through brute-force enumeration.

C. Accelerator Mapping

For a given dataflow grouping choice, we need to map the different groups to different accelerators. The formulation of accelerator mapping is as follows

$$\begin{aligned} &Map : \{D_1, D_2, ..., D_K\} \rightarrow A \\ &Time : \{D_1, D_2, ..., D_K\} \rightarrow \mathcal{R} \end{aligned} \tag{2}$$

where D_i is one layer group after dataflow grouping, A is the set of accelerators, $\mathcal{R}$ is real number domain. The function Map determines the mapping from one group to one accelerator. $Map(D_i) = acc$ means that group D_i is executed on accelerator acc. The second function $Time$ determines the start time of the execution of one group. $Time(D_i) = t$ means group D_i starts to execute at time t.

Not all the mappings are valid because of two constraints. First, **resource constraint**: all the groups should make sure not to exceed hardware resource limits. Second, **order constraint**: the groups have execution order as shown in Equation 1, which should be followed in mapping. For resource constraint, we can get the resource requirements (PE usage and memory usage) of a mapping through off-the-shelf evaluation model such as Maestro [2] and TENET [10]. As a result, we can formulate the resource constraint for D_i as follows

$$\begin{aligned} &\sum_j \text{PEUsage}(L^i_j, \text{Dataflow}(Map(D_i))) \leq \text{NumPE}(Map(D_i)) \\ &\sum_j \text{MemUsage}(L^i_j, \text{Dataflow}(Map(D_i))) \leq \text{MemCap}(Map(D_i)) \end{aligned} \tag{3}$$

where the PEUsage and MemUsage functions are provided by hardware models (Maestro). For order constraint, we need to make sure each group is not executed before other groups that it depends on, which can be formulated as follows

$$Time(D_j) \geq Time(D_i) + Cost(D_i), \;\; \forall D_i \preceq D_j \text{ and } D_j \npreceq D_i \tag{4}$$

where $Cost(D_i)$ is the total runtime cost of communication and computation. The cost can be calculated as follows

$$\begin{aligned} &Cost(D_i) = \text{CompCost}(D_i) + \text{CommCost}(D_i) \\ &\text{CommCost}(D_i) = \sum_{L^i_j \in D_i} \sum_{L' \in Pred(L^i_j)} \text{Func}_i(L') \\ &\text{Func}_i(L') = \text{Comm}(Map(\text{GroupOf}(L')), Map(D_i), \text{DV}(L')) \end{aligned} \tag{5}$$

The CompCost function is provided by hardware models. The explanations of functions Comm, GroupOf, and DV are shown in Table I. In Figure 3 part c) we show two possible accelerator mapping choices for a given grouping choice in part b). Different accelerator mapping choices can result in different data transfer overhead. For this example, the left one needs 5 NoC hops for data transfer, while the right one only needs 3 hops. To choose the optimal mapping is hard and the design space is also huge ($O(K^N)$, K is number of groups, N is number of accelerators).

D. Mapping Problem Formulation

Based on our formulation of dataflow grouping and accelerator mapping, we formulate the problem of mapping heterogeneous DNNs to heterogeneous SoC as follows

$$\begin{aligned} &\min_{D_1 \preceq ... \preceq D_K, Map, Time} \; \max_i \{Time(D_i) + Cost(D_i)\} \\ &\text{s.t. Constraints in Equation 3 and Equation 5 hold} \end{aligned} \tag{6}$$

The maximum end time each group D_i, which is calculated by $\max_i Time(D_i) + Cost(D_i)$, represents the end-to-end execution time of the multi-graph $\mathcal{G}$. So minimizing the maximum end time is equivalent to minimizing the end-to-end latency. Other metrics such as throughput and energy consumption can also be considered based on our formulation of dataflow grouping and accelerator mapping.

IV. SCHEDULING ALGORITHM OF COMB

In this section, we present our scheduling algorithm. As explained in Section III, the design space for dataflow grouping and accelerator mapping is extremely huge and can't be explored through brute-force enumeration. We find that the optimization process of dataflow grouping can be formulated as a genetic algorithm and the accelerator mapping can be effectively solved by a greedy algorithm. Moreover, these two optimizations should be done jointly to get an optimized mapping. As a result, our scheduling algorithm is a hybrid of genetic algorithm and greedy algorithm.

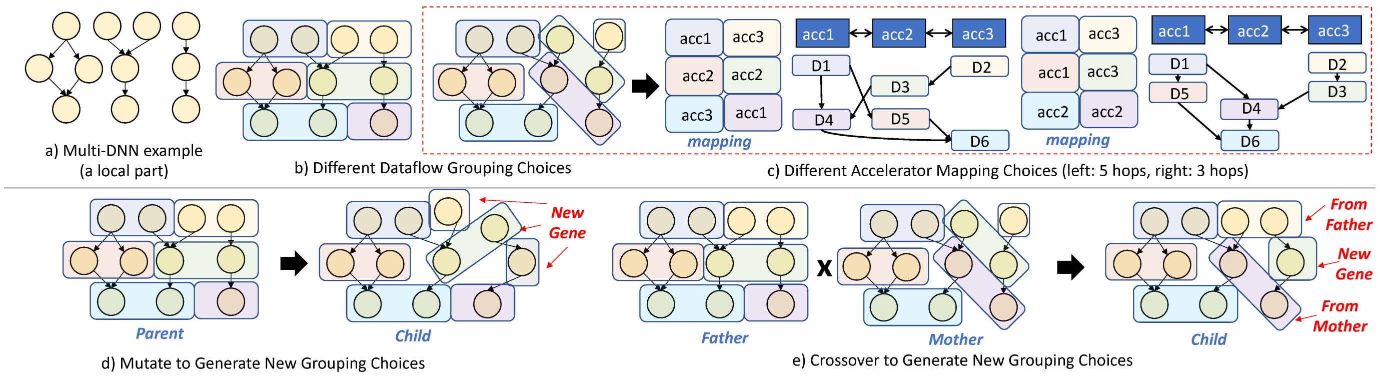

a) Multi-DNN example (a local part)
b) Different Dataflow Grouping Choices
acc1
acc3
acc2
acc2
acc3
acc1
mapping
acc1
acc2
acc3
D1
D2
D3
D4
D5
D6
acc1
acc3
acc1
acc3
acc2
acc2
mapping
c) Different Accelerator Mapping Choices (left: 5 hops, right: 3 hops)
Parent
Child
New Gene
d) Mutate to Generate New Grouping Choices
Father
Mother
Child
From Father
New Gene
From Mother
e) Crossover to Generate New Grouping Choices

Fig. 3: Examples to explain dataflow grouping and accelerator mapping in detail. Part d) and e) show how our genetic algorithm generates new grouping choices.

Algorithm 1: Skeleton of COMB Scheduling Algorithm

```
input : Multi-DNN graph $\mathcal{G}$, SoC graph $H = (A, B)$
input : hyperparam $K$, mutation ratio $r$, iteration steps NumIter
output: Optimized mapping $map$
1  opt_lat = inf; $map$ = None;
2  population = RandGenGrouping($\mathcal{G}$);
3  score_board = EmptyDict();
4  for iter in range(NumIter) do
5      for group_choice in population do
6          mapping = GreedyAccMapping(group_choice, $H$);
7          latency = Fitness(mapping);
8          score_board[group_choice] = (mapping, latency);
9          if latency $<$ opt_lat then
10             opt_lat = latency; $map$ = mapping;
11     topk = score_board.topk($K$); population.clear();
12     for father in topk do
13         if rand() $<$ $r$ then
14             population.add(Mutate(father, $\mathcal{G}$));
15         for mother in topk do
16             population.add(Crossover(father, mother, $\mathcal{G}$));
17 return $map$;
```

A. The Skeleton of the Algorithm

In this part, we show the skeleton of our scheduling algorithm and the detailed parts of the algorithm will be explained in later sections. The skeleton is shown in Algorithm 1. In the genetic algorithm, the genome is a dataflow grouping choice $D_1 \preceq D_2 \preceq ... \preceq D_K$. The group D_i is the gene in this genome. The algorithm randomly generates the initial population at line 2, which is a set of randomly generated grouping choices. Then, from line 4, the algorithm begins to iterate on the population by continuously mutating (use **Mutate** function at line 14) or crossovering (use **Crossover** function at line 16) the genomes in the population. The good genomes are selected by using the **Fitness** function (at line 7), which evaluates the latency of the mapping that is produced according to a genome (grouping choice). To get the mapping, we need to decide the accelerator mapping using **GreedyAccMapping** function at line 6. After several steps (usually 10 steps are enough) of iterations, the final optimized mapping will be returned by this algorithm. In the following sections, we will explain the functions **GreedyAccMapping**, **Fitness**, **Mutate**, and **Crossover**.

B. Greedy Accelerator Mapping

The greedy algorithm that selects accelerator mapping choices is shown in Algorithm 2. Given a group choice $D_1 \preceq ... \preceq D_K$, the algorithm iterates over each group D_i and greedily selects the best accelerator for it by minimizing the end time of all the layer groups before D_i. To do this, the algorithm evaluates end time of the execution of all the layers in D_i by querying the end time of the layer's predecessor layers (line 10-14) and the execution time of the target accelerator (at line 9). Then, the algorithm selects the largest end time of the layers as the end time of D_i (line 15) because D_i has to wait all its previous layers. From line 16 to 19, the algorithm records the minimal possible ending time for D_i. This greedy algorithm follows all the constraints in Equation 5. So the generated mapping is always valid.

C. Explanation about Other Functions

Fitness Function is used by the genetic algorithm to judge the quality of a genome. Our fitness function first checks if a mapping conforms to the resource constraints in Equation 3. If the mapping violates the constraints, the quality is *inf* (the lowest quality). Otherwise, the quality is the end-to-end latency. The latency is measured on the SoC with hardware evaluators.

Mutate Function is used to generate a new genome (grouping choice) from an existing genome. In Figure 3 part d) we show an example of Mutate Function. In detail, a parent grouping choice is analyzed and part of its groups (genes) are fixed, which will be passed to its child grouping choice. Other groups that are not fixed will be re-grouped randomly (produce new genes). In mutation, we choose the groups that have high PE utilization (fully utilize hardware resources) to be fixed.

Crossover Function is used to generate a new genome by selecting good genes from two existing genomes. In Figure 3 part e) we show an example of crossover. Part of the groups from father grouping choice and mother grouping choice are passed to the child grouping choice without modification. But the groups from father and mother may have conflicts with each other because they overlap with each other and thus they can't coexist in the child grouping choice. For such cases, we randomly reserve the group from either father or mother, and the remaining ungrouped layers will be re-grouped randomly to produce new genes in the child grouping choice.

Algorithm 2: Greedy Accelerator Mapping Algorithm

```
input : Dataflow grouping choice $group = [D_1, D_2, ..., D_K]$, SoC
        graph $H = (A, B)$
output: Datapath selection results (Map, Time)
1  num_groups = $group$.size(); num_acc = $A$.size();
2  Map = Dict(); Time = Dict(); AccTable = Dict();
3  for $D_i$ in $group$ do
4      Map[$D_i$] = None; Time[$D_i$] = inf;
5      for $acc$ in $A$ do
6          end_$time_{D_i}$ = 0;
7          for Layer $L_j^i$ in $D_i$ do
8              start_time = AccTable[$acc$];
9              comp_time = HWModel($L_j^i$);
10             for Layer $L'$ in Pred($L_j^i$) do
11                 $D'$ = GroupOf($L'$);
12                 end_$time_{D'}$ = Time[$D'$]; $acc_{D'}$ = Map[$D'$];
13                 start_time = max(start_time, end_$time_{D'}$ +
                     Comm($acc_{D'}$, $acc$, DV($L'$)));
14             end_time = start_time + comp_time;
15             end_$time_{D_i}$ = max(end_$time_{D_i}$, end_time);
16         if end_$time_{D_i}$ < Time[$D_i$] then
17             Time[$D_i$] = end_$time_{D_i}$;
18             Map[$D_i$] = $acc$;
19     AccTable[Map[$D_i$]] = Time[$D_i$];
20 return Map, Time;
```

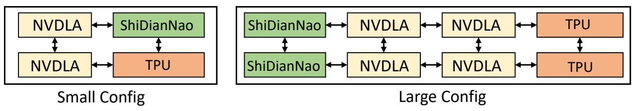


Fig. 4: Connection of the SoCs in evaluation.

V. Evaluation

A. Evaluation Setup

Workloads: we follow the workloads selection in previous work [1], [6], [8]. We use three types of heterogeneous DNN workloads: Vision (ResNet-50, MobileNet-V2, GoogleNet, Language (Albert, Bert-Base, LSTM), Mixed (Yolo-V5 , GoogleNet, Bert-Base, LSTM, MobileNet-V2). By changing the number of networks in each type, we get totally of six different workloads as shown in Table II.

SoC Configuration: we use the simulator from H2H [5] to simulate full SoC performance and use the hardware model Maestro [2] to evaluate the latency and energy consumption for each accelerator. We consider three different accelerators in the SoC: NVDLA, ShiDianNao [13], and TPU [14] (each accelerator has 128 × 128 PEs). We prepare two SoC configurations as shown in Table II. The small SoC contains two NVDLA instances, one ShiDianNao instance, and one TPU instance; the large SoC contains four NVDLA instances, two ShiDianNao instances, and two TPU instances. The NoC connections of the two configurations are shown in Figure 4. The frequency of all the accelerators is 200 MHz. We use two network bandwidth configurations for NoC: Low-BW (0.8GB/s) and High-BW (3.2GB/s).

Baseline: we compare to H2H [5] and MAGMA [6] in evaluation. H2H uses a dynamic programming algorithm to optimize communication in mapping. MAGMA uses a genetic algorithm for design space exploration. They both only consider mapping one layer/job to one accelerator at a time without consideration for spatial resource sharing. They don't consider the non-uniform data transfer overhead of NoC and assume all the accelerators can communicate with each other uniformly.

TABLE II: Setup of Workloads and SoC.

Heterogeneous DNN Workloads	
Vision-Light	2 ResNet-50, 2 MobileNet-V2, 2 GoogleNet
Vision-Heavy	8 ResNet-50, 8 MobileNet-V2, 8 GoogleNet
Language	2 Albert, 2 Bert-Base, 4 LSTM
Mixed-Light	2 Yolo-V5, 2 GoogleNet, 2 Bert-Base 2 LSTM, 2 MobileNet-V2
Mixed-Mid	4 Yolo-V5, 4 GoogleNet, 4 Bert-Base 2 LSTM, 4 MobileNet-V2
Mixed-Heavy	8 Yolo-V5, 8 GoogleNet, 8 Bert-Base 2 LSTM, 8 MobileNet-V2
Heterogeneous SoC Configs	
Small	2 NVDLA, 1 ShiDianNao, 1 TPU
Large	4 NVDLA, 2 ShiDianNao, 2 TPU

B. Overall Latency and Energy Results

For all the workloads and SoC configurations, we evaluate the latency and energy consumption of H2H, MAGMA, and COMB and show the results in Figure 5. As the results shown, COMB achieves the best latency for all the cases, reducing from 7.6% to 47.9% latency compared to H2H (0.4% to 45.7% compared to MAGMA) for all the cases. For small SoC and Low-BW, the speedup to H2H ranges from 1.23× to 1.91×; the speedup to MAGMA ranges from 1.21× to 1.84×. For Large-SoC and Low-BW, the geometric speedup to H2H is 1.38× and the geometric speedup to MAGMA is 1.28×. The results of High-BW are similar, which shows the scalability and generality of COMB. Overall geometric speedup to H2H is 1.37× and speedup to MAGMA is 1.28×. Our scheduling algorithm in COMB can automatically adapt to different accelerator configurations and topology without manual interference.

As for energy consumption, COMB uniformly achieves the minimal energy consumption for all the cases. Overall, COMB can reduce from 12.3% to 46.5% energy consumption compared H2H (1.9% to 46.3% compared to MAGMA). Overall, COMB can reduce 29.2% energy compared to H2H (22.7% compared to MAGMA). The reasons are two folds. First, COMB can group more layers together to share the spatial resources to reduce the number of idle PEs (idle PEs also consume energy). Second, COMB reduces the delay of data transfer and thus reduces the energy consumption of communication.

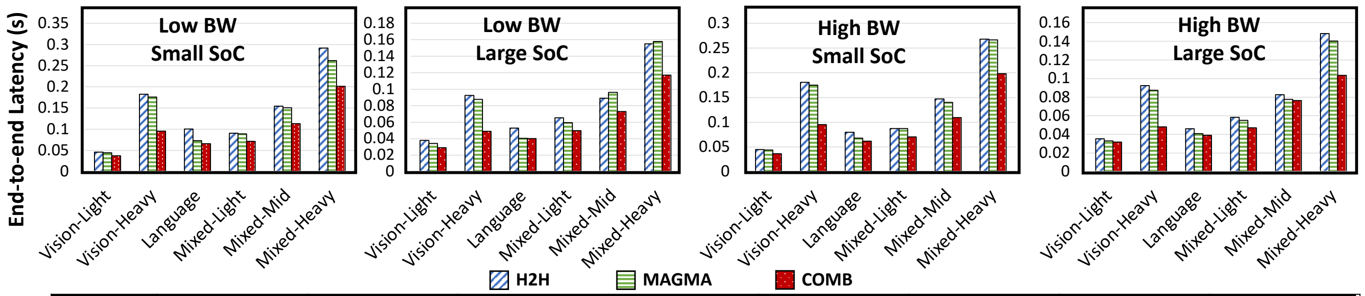
C. Efficiency of Dataflow Grouping

To show the effectiveness of COMB's dataflow grouping optimization, we show the detailed accelerator utilization (calculated by dividing the PE busy time by total execution time) in Figure 6a. We use Large SoC and High-BW and the workload is Mixed-Heavy. Overall, COMB achieves better utilization for all the accelerators (the number after accelerator name is the position in the mesh SoC). The utilization of COMB is 1.28× to that of H2H and MAGMA on average. ShiDianNao's utilization is low because only depthwise convolutions are mapped to ShiDianNao in our SoC.

To show the efficiency of COMB's genetic algorithm, we show the performance of the steps of genetic searching in Figure 6b. As the results show, COMB can quickly optimize the latency by finding better dataflow grouping choices in several steps (usually 10 steps), demonstrating the efficiency of the genetic algorithm.

D. Efficiency of Accelerator Mapping

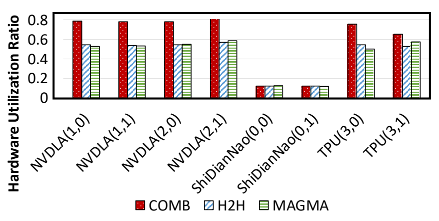
To show the efficiency of COMB's accelerator mapping algorithm, we prepare another version of COMB without our greedy accelerator mapping algorithm and use a round-robin mapping algorithm instead (called COMB-RR). We compare the performance of the two versions of COMB and show the results in Figure 6c. The results show that our



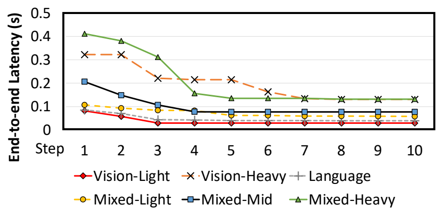
Energy Consumption (nJ)

SoC	BW	Scheduler	Vision-Light	Vision-Heavy	Language	Mixed-Light	Mixed-Mid	Mixed-Heavy
Small	Low	H2H	1.43E+09	5.56E+09	3.27E+09	2.58E+09	4.60E+09	8.75E+09
		MAGMA	1.4E+09	5.64E+09	2.83E+09	2.52E+09	4.47E+09	8.65E+09
		COMB	**1.08E+09**	**3.03E+09**	**2.78E+09**	**1.79E+09**	**3.14E+09**	**6.24E+09**
	High	H2H	1.40E+09	5.65E+09	2.99E+09	2.25E+09	4.07E+09	7.64E+09
		MAGMA	1.38E+09	5.52E+09	2.65E+09	2.24E+09	4.03E+09	7.64E+09
		COMB	**8.42E+08**	**3.01E+09**	**2.51E+09**	**1.53E+09**	**2.76E+09**	**5.36E+09**
Large	Low	H2H	1.53E+09	5.80E+09	3.85E+09	2.99E+09	5.17E+09	9.41E+09
		MAGMA	1.39E+09	5.56E+09	3.26E+09	2.65E+09	4.80E+09	9.27E+09
		COMB	**9.18E+08**	**3.15E+09**	**2.90E+09**	**1.98E+09**	**3.51E+09**	**6.49E+09**
	High	H2H	1.43E+09	5.76E+09	2.91E+09	2.46E+09	4.36E+09	8.26E+09
		MAGMA	1.41E+09	5.52E+09	2.75E+09	2.35E+09	4.13E+09	7.93E+09
		COMB	**1.25E+09**	**3.08E+09**	**2.54E+09**	**1.59E+09**	**3.24E+09**	**5.44E+09**

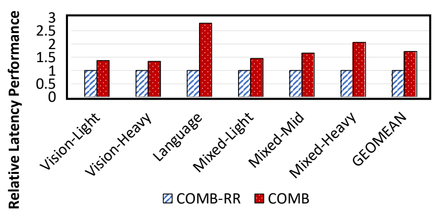
Fig. 5: End-to-end latency (the unit is second) and energy consumption (the unit is nJ) evaluation results.



(a) Hardware utilization comparison of COMB, H2H, and MAGMA. The workload is Mixed-Heavy.



(b) The achieved latency performance of the search steps of COMB's genetic algorithm.



(c) Latency comparison of COMB-RR (round-robin accelerator mapping) and COMB.

Fig. 6: Detailed performance analysis of COMB. We use Large-SoC and High-BW.

greedy accelerator mapping algorithm can provide 1.72× speedup in latency on average, demonstrating the efficiency of our algorithm.

VI. Conclusion

Deploying heterogeneous DNNs to heterogeneous SoCs is challenging because of the intractable design space and complex mapping decisions. Existing mapping frameworks suffer from inefficiencies for lack of awareness of the computation and data transfer imbalance problem. In this paper, we propose COMB to use a hybrid algorithm based on greedy algorithm and genetic algorithm to efficiently explore the space with dataflow grouping and accelerator mapping to improve the mapping performance. In evaluation, COMB achieves 1.28× and 1.37× speedup for latency compared to MAGMA and H2H.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (NSFC) under grant No. U21B2017 and National Key R&D Program of China No. 2022YFB4500401.

References

[1] H. Kwon et al. "Heterogeneous Dataflow Accelerators for Multi-DNN Workloads," in HPCA 2021
[2] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer and A. Parashar, "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," in MICRO 2019
[3] Michael Ditty et al. Nvidia's xavier soc. In Hotchips 2018.
[4] P. Mantovani et al., "Agile SoC Development with Open ESP: Invited Paper," in ICCAD 2020
[5] X. Zhang, C. Hao, P. Zhou, A. K. Jones, and J. Hu, "H2H: heterogeneous model to heterogeneous system mapping with computation and communication awareness," in DAC 2022
[6] S.-C. Kao and T. Krishna, "MAGMA: An Optimization Framework for Mapping Multiple DNNs on Multiple Accelerator Cores," in HPCA 2022
[7] Size Zheng et al. "Chimera: An Analytical Optimizing Framework for Effective Compute-intensive Operators Fusion." in HPCA 2023
[8] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-Task Scheduling Algorithm For Preemptible Neural Processing Units," in HPCA 2020
[9] E. Baek, D. Kwon, and J. Kim, "A Multi-Neural Network Acceleration Architecture," in ISCA 2020
[10] Liqiang Lu et al. "TENET: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation," in ISCA 2021
[11] Qingcheng Xiao et al. "HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation," in ISCA 2021
[12] Size Zheng et al. "FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System." in ASPLOS 2020
[13] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in ISCA 2015
[14] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit", in ISCA 2017
[15] Size Zheng et al. "AMOS: enabling automatic mapping for tensor computations on spatial accelerators with hardware abstraction," in ISCA 2022