Inter-layer Scheduling Space Definition and Exploration for Tiled Accelerators

Jingwei Cai*†
caijw21@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Yuchen Wei*†
weiyc22@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Zuotong Wu
eliminatespace@gmail.com
Xi'an Jiaotong University
IIISCT
Xi'an, Shaanxi, China

Sen Peng
nice_day1330@163.com
Xi'an Jiaotong University
IIISCT
Xi'an, Shaanxi, China

Kaisheng Ma‡
kaisheng@mail.tsinghua.edu.cn
Tsinghua University
Beijing, China

ABSTRACT

With the continuous expansion of the DNN accelerator scale, inter-layer scheduling, which studies the allocation of computing resources to each layer and the computing order of all layers in a DNN, plays an increasingly important role in maintaining a high utilization rate and energy efficiency of DNN inference accelerators. However, current inter-layer scheduling is mainly conducted based on some heuristic patterns. The space of inter-layer scheduling has not been clearly defined, resulting in significantly limited optimization opportunities and a lack of understanding on different inter-layer scheduling choices and their consequences.

To bridge the gaps, we first propose a uniform and systematic notation, the Resource Allocation Tree (RA Tree), to represent different inter-layer scheduling schemes and depict the overall space of inter-layer scheduling. Based on the notation, we then thoroughly analyze how different inter-layer scheduling choices influence the performance and energy efficiency of an accelerator step by step. Moreover, we show how to represent existing patterns in our notation and analyze their features. To thoroughly explore the space of the inter-layer scheduling for diverse tiled accelerators and workloads, we develop an end-to-end and highly-portable scheduling framework, SET. Compared with the state-of-the-art (SOTA) open-source Tangram framework, SET can, on average, achieves 1.78× performance improvement and 13.2% energy cost reduction simultaneously. Moreover, *the SET framework will be open-sourced.*

CCS CONCEPTS

• **Software and its engineering** → **Compilers**; • **Computer systems organization** → **Parallel architectures**.

KEYWORDS

scheduling, inter-layer scheduling, neural networks, tiled accelerators

ACM Reference Format:
Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. 2023. Inter-layer Scheduling Space Definition and Exploration for Tiled Accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3579371.3589048

1 INTRODUCTION

A wide range of deep neural networks (DNNs) has been created to solve real-world problems in many fields, such as image recognition [19, 22, 31], object detection [18, 44], and natural language processing [13, 18, 21]. With the pursuit of higher accuracy and better adaptability to complex scenarios, DNNs are becoming deeper, and their structures are becoming more complex [10, 13, 18, 19, 22].

Under this trend, many large-scale accelerators, whose scales range from dozens of square millimeters to even a whole wafer, have been developed to accelerate the inference period of such increasingly complex DNNs [4, 16, 17, 26, 46, 52, 56]. Considering that a single large hardware-tile (HW-tile) suffers low utilization and energy efficiency, these large-scale accelerators mainly employ the tiled architecture [4, 16, 17, 26, 46, 52, 56], where each HW-tile includes a Processing Element (PE) array and a global buffer and is interconnected by a Network on Chip (NoC).

However, the tiled architecture itself cannot easily guarantee high utilization and energy efficiency. How to efficiently translate massive computing and storage resources into actual performance is still an open challenge [16, 17, 46, 51, 61]. The critical point of solving this challenge depends on *scheduling*, which can be categorized into *intra-layer scheduling* and *inter-layer scheduling*.

Intra-layer scheduling studies how to map a single layer onto one or several HW-tiles. Its space has been depicted by many notations [32, 37, 42, 60] and explored by various methods [20, 23, 28].

*Both authors contributed equally to this research.
†Work partially done during the internship at IIISCT.
‡Corresponding author.

ISCA '23, June 17–21, 2023, Orlando, FL, USA.

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589048

Inter-layer scheduling studies how to schedule the computing order and resource allocation of all layers in a DNN on an accelerator, which also significantly influences the energy efficiency and performance of the accelerators. For example, Fused-layer [3] allocates different computing resources for different layers and orchestrates the layers in a layer-pipeline (LP) manner, achieving a 95% reduction in DRAM access compared with a basic-version layer-sequential (LS) pattern which computes layers one after another. Moreover, the larger the accelerator scale, the greater the impact of inter-layer scheduling. For example, Tangram, which proposes optimized LP and LS techniques [17], achieves 67% energy savings on a 32×32-tile accelerator, compared to less than 20% savings on a 4×4-tile accelerator.

Although inter-layer scheduling plays an increasingly important role in keeping tiled accelerators highly utilized and energy-efficient, there is a significant deficiency in its research: Most works [3, 6, 17, 38, 39, 55, 62] continue to optimize existing heuristic patterns, LP and LS, but do not propose new patterns, let alone define the space of inter-layer scheduling on tiled accelerators clearly and systematically. The lack of a clear definition of the inter-layer scheduling space significantly limits the opportunities for optimizing the performance and energy efficiency of tiled accelerators. Moreover, the lack of a systematic definition hinders people from understanding how different inter-layer scheduling choices influence different hardware behaviors and how these behaviors further influence the energy efficiency and performance of an accelerator. These challenges motivate us to make the following contributions:

- We introduce a uniform and systematic *Resource Allocation Tree* notation to depict the space of inter-layer scheduling for inferencing DNNs with various structures on tiled accelerators with various architectures. The notation includes *Temporal Cut*, which allocates the same HW-tile group and different computing time intervals to each of its children, and *Spatial Cut*, which allocates different HW-tile groups to each child of it. Each RA Tree is a hierarchical organization of Cuts and Leaf Nodes (layers of the DNN). Then, we elaborate on how to parse the tree structure into the corresponding resource allocation scheme and the flow of data among the HW-tiles. To the best of our knowledge, we are **the first** to define the space of inter-layer scheduling clearly and systematically.
- Based on the RA Tree notation, we thoroughly analyze how different inter-layer scheduling choices affect hardware behaviors and how these behaviors affect accelerator performance and energy efficiency. Moreover, we represent existing LS and LP patterns in our notation and analyze their features.
- Combining the above, we develop an end-to-end and highly-portable scheduling framework, SET, to automatically explore the whole DNN scheduling space for tiled accelerators. To explore the vast newly-defined inter-layer scheduling space efficiently, we equip SET with a Simulated-Annealing-based algorithm with 6 specifically designed operators. SET can be ported to various tiled accelerators with minor modifications, featuring good portability. We have developed an end-to-end SET deploying flow for our test chip. The framework is available at https://github.com/SET-Scheduling-Project/SET-ISCA2023.
- Compared with the SOTA open-source Tangram framework, SET can, on average, achieves 1.78× performance improvement and 13.2% energy cost reduction simultaneously. In addition, we conduct a number of experiments on different DNNs, batch sizes, and hardware platforms to demonstrate the universality of SET and the effect of exploring the newly-defined space over existing LS and LP scheduling patterns. Moreover, we leverage SET to analyze the characteristics of LS and LP and reveal the features of the inter-layer scheduling space.

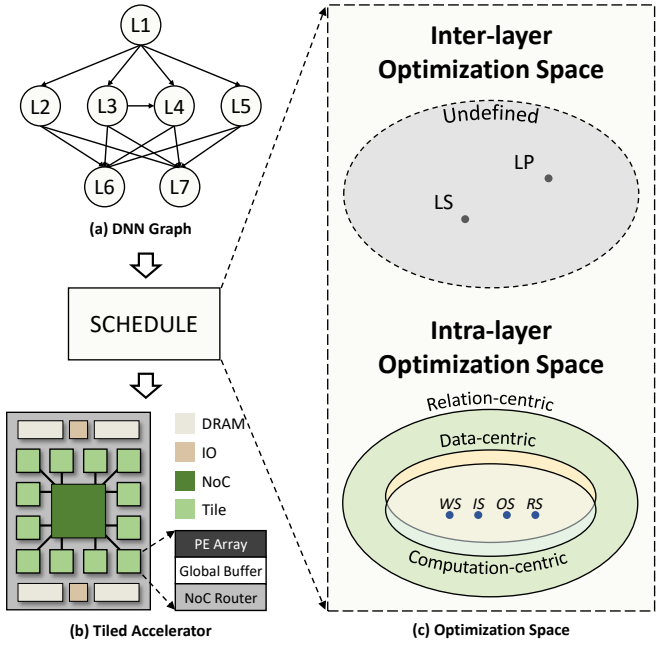

Figure 1: *Left*: The deploying flow from the DNN graph to the accelerator. *Right*: The two spaces for scheduling DNNs on the tiled accelerators, including undefined inter-layer scheduling space and well-defined intra-layer scheduling space (the inclusion relationship between the intra-layer scheduling spaces defined by different notations is drawn according to Tenet [37]).

2 BACKGROUND AND MOTIVATION

2.1 Tiled Accelerators

In recent years, "Tiled Accelerators" (also known as "Spatial Accelerators") have become very popular in DNN acceleration. Many tiled DNN accelerators have been developed by industry [7, 25, 30, 35, 50, 52, 56, 59] and academia [11, 16, 17, 46, 55, 63]. Based on these existing tiled accelerators, we aim to abstract out a basic hardware requirement (or hardware template) to support our inter-layer scheduling research, as has been well done in intra-layer scheduling [20, 23, 28, 32, 42, 57, 60].

Before delving into the hardware requirements of inter-layer scheduling, we can first abstract out the distinct hardware behaviors under inter-layer scheduling compared to intra-layer scheduling. These behaviors include (1) the parallel execution of different layers on different HW-tiles, which necessitates independent control of each HW-tile and requires efficient support for concurrent access to DRAMs; and (2) the producer-consumer data

communication between HW-tiles or HW-tile and DRAM, which requires a flexible NoC and efficient support for concurrent access to buffers. While most existing tiled accelerators have supported the inter-layer scheduling behaviors mentioned above efficiently [7, 25, 50, 52, 56, 64], their architecture details, such as control logic, NoC architecture, etc., vary due to different design goals and considerations. Nevertheless, from the inter-layer scheduling perspective, we can clearly identify the common features of these accelerators. Based on these common features, we give the definition of the tiled accelerator mentioned in this paper, which is also the primary hardware requirement of our notation, as follows:

- The basic components of a tiled accelerator include an NoC, some HW-tiles, DRAM PHY and controllers, and other IOs (Fig. 1 (b)). The NoC should be able to connect the HW-tiles, DRAMs, and other components. Moreover, each HW-tile can access arbitrary global buffers of other HW-tiles or the DRAMs of the accelerator through NoC.
- The basic components of a HW-tile include a unified buffer, an NoC router, and PEs. A HW-tile can be assigned to different layers at different times but cannot compute workloads for multiple layers simultaneously. What is worth mentioning is that a HW-tile is not equivalent to a core in our definition. Any computing unit that meets the above definition, such as a Simba chiplet [46] in a package, can be viewed as a HW-tile.

In summary, our RA Tree notation is applicable to all tiled accelerators satisfying the above basic requirement without specifying intra-tile or NoC architectures, which is where SET portability comes from.

2.2 Scheduling for Tiled Accelerators

Scheduling plays a crucial role in deploying DNNs on accelerators with high utilization and energy efficiency. We categorize DNN scheduling into *inter-layer* and *intra-layer scheduling*, which share a top-down relationship. When we schedule a DNN onto a target tiled accelerator, the inter-layer scheduling first decides the computing order and resource allocation of each layer, and then the intra-layer scheduling decides how to map the layer on the allocated HW-tile group.

Specifically, intra-layer scheduling studies how to map a layer on parallel computing units spatially, tile workloads into small pieces to fit the capacity of each-level memory, and control the computing order of each small workload. A lot of research has been done in this field [16, 20, 23, 29, 32, 33, 37, 42, 54, 58]. If we study the development of this field carefully, we can find that, in the early days, many heuristic intra-layer scheduling patterns are proposed, such as output stationary (OS) pattern [8, 11, 14], weight stationary (WS) pattern [15, 26, 27, 40, 41], input stationary (IS) pattern [43], and row stationary (RS) pattern [9, 12]. We call this stage the "Heuristic Period". With the deepening of researchers' understanding, intra-layer scheduling moves into the "Automation Period". In this period, various notations, such as Computation-centric notation [42, 60], Data-centric notation [32], and relation-centric notation [37], are proposed to specify the overall space of intra-layer scheduling and explore it thoroughly (Fig. 1(c)).

In *inter-layer scheduling*, there mainly exist two heuristic patterns, *layer-sequential* and *layer-pipeline*. LS employs all computing resources to process each layer one after another, while LP allocates different groups of HW-tiles to different layers and orchestrates these layers in a pipeline manner. Existing works either optimize LS or LP themselves [17, 61, 62] or optimize them for various applications [6, 34, 38, 55] or hardware [3, 45, 47, 63]. However, if we compare the development of inter-layer scheduling with intra-layer scheduling, we argue that inter-layer scheduling is still in the "Heuristic Period". The space of inter-layer scheduling has not been clearly defined, explored thoroughly, or fully understood. This situation motivates us to propose a notation to specify the space of inter-layer scheduling and then leverage the notation to explore the space thoroughly and analyze the influences of different inter-layer scheduling choices.

3 RA TREE DEFINITION AND ANALYSIS

3.1 RA Tree Notation

As shown in Fig. 2(a), we develop a uniform notation, Resource Allocation Tree (**RA Tree**) to describe inter-layer scheduling schemes. In this section, we provide the definition and properties of the components in the RA Tree to demonstrate how to generate its corresponding inter-layer scheduling scheme from an arbitrary RA Tree. The generation methods of different RA Trees by our SET framework are introduced in Sec. 4.2.

3.1.1 Overall Introduction to RA Tree Notation. The central theme of RA Tree notation revolves around resource allocation. We identify two resource types: *spatial resources*, which are the total computing resources, and *temporal resource*, which is usage duration of the computing resources. Our notation is a structured and recursive notation to describe the allocation of the spatial and temporal resources for each sample within each layer, i.e., determine the number of HW-tiles and the duration of their usage for each sample within every layer.

It is worth noting that in this paper, the granularity of spatial computing resource allocation is defined at the level of HW-tiles (depicted in the bottom-left corner of Fig. 1 and defined in Sec. 2.1),

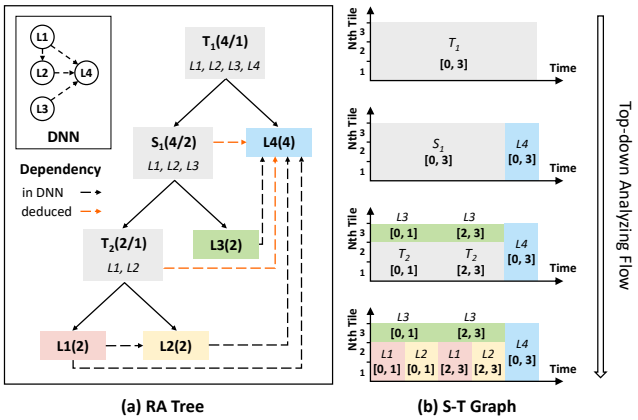

Figure 2: (a) An example RA Tree. (b) The top-down hierarchical analyzing flow of the left RA Tree. The bottom one is the S-T Graph corresponding to the left RA Tree. For simplicity, the dependencies of the RA Tree are only drawn in this figure.

including all buffers and the PE array within it. However, in general, the granularity can be coarser or finer, depending on whether the computing resources can be independently controlled to compute different layers. For instance, in this paper, the PE array cannot be independently controlled and must work collectively to compute the partitioned workload of the same layer assigned to each HW-tile, which is also the control granularity adopted by numerous tiled accelerators today [7, 16, 17, 35, 46, 50, 52]. However, if each PE within a HW-tile can be independently controlled to compute different layers, the allocation granularity of our RA Tree can be adjusted to the PE level.

For a better understanding of the scheme represented by an RA Tree, we also introduce the Spatial-Temporal Graph (**S-T Graph**) to visually display the allocation of the two resources. The S-T Graph is used to explain the RA Tree examples in this work and is not necessary for our SET framework. The bottom graph of Fig. 2(b) illustrates the S-T Graph corresponding to Fig. 2(a). The two axes correspond to the resource types: the X-axis is the timeline, and the Y-axis is the HW-tiles on the accelerator. Each colored rectangle corresponds to the processing of a layer with a specific batch of samples. The height of the rectangle indicates which HW-tiles process the layer, and the width is the time period of the processing. Each rectangle is marked with its corresponding layer and batch samples. For example, "$L1[0,1]$" means the processing of layer $L1$ with samples 0 to 1, a batch of size 2.

3.1.2 Basic Attributes and Encoded Information of Node. In this part, we will firstly give the symbolic definition for the RA Tree, then explain the terms and rules in the definition.

Denote L as the set of layers in the DNN model, HWT as the set of HW-tiles on the accelerator, b_{tot} as the number of a sample in a batch of the workload (also denoted as the *total batch size*). An RA Tree R is a set of Nodes:

$$R = \{N_1, N_2, \ldots, N_m\}$$

Where each Node N_i is in the form of

$$N_i = (type_i, C_i, L_i, TG_i, b_i, sb_i, FROM_i, TO_i)$$

$$\text{Here } type_i \in \{Leaf, SCut, TCut\}, \tag{1}$$

$$C_i \subset R \text{ is the children of } N_i \text{ in the tree structure}, \tag{2}$$

$$L_i \subset L, \quad TG_i \subset HWT, \tag{3}$$

$$b_i, sb_i \in \mathbb{Z}^+, \ b_i \% sb_i = 0, \tag{4}$$

$$FROM_i = \{N_j \mid j \neq i, \exists l \in L_j, l' \in L_i, l \rightarrow l'\}, \tag{5}$$

$$TO_i = \{N_j \mid j \neq i, \exists l \in L_j, l' \in L_i, l' \rightarrow l\} \tag{6}$$

($l \rightarrow l'$ means l' needs the ofmaps of l)

$$\text{When } type_i = Leaf: \quad C_i = \emptyset, \quad |L_i| = 1, \quad sb_i = 1 \tag{7}$$

$$\text{When } type_i \neq Leaf: \quad C_i \neq \emptyset, \quad L_i = \bigcup_{N_j \in C_i} L_j, \quad \forall N_j \in C_i, \ b_j = \frac{b_i}{sb_i}, \tag{8}$$

$$TG_i = \begin{cases} TG_j \ (\forall N_j \in C_i) & , type_i = TCut \\ \bigcup_{N_j \in C_i} TG_j & , type_i = SCut \end{cases} \tag{9}$$

$$\text{When } type_i = SCut, \forall N_j, N_k \in C_i, TG_j \cap TG_k = \emptyset \tag{10}$$

$$\text{If } N_r \text{ is the root of the tree}, L_r = L, TG_r = HWT, b_r = b_{tot}$$

As encoded in $type_i$, an RA Tree has three types of Nodes: Leaf, S Cut and T Cut, the latter two are collectively referred as Cuts (Formula (1)). A Leaf (non-gray rectangle in Fig. 2(a)) represents the processing of a single layer (the layer in L_i in Formula (7)), and the layer is labeled on the Leaf (*e.g.* $L1$ on pink rectangle in Fig. 2(a)). The number of Leaves is equal to the number of layers in the DNN. A Cut is represented by a gray rectangle in an RA Tree (e.g., T_1, S_1, T_2 in Fig. 2(a)), has two levels of representation. From the perspective of its parent node, it represents the processing of all layers beneath it (L_i in Formula (8)), which are labeled on the second line of the Cut Node. For example, from the perspective of T_1, S_1 represents the processing of $L1$,$L2$, and $L3$. From the perspective of its children, it illustrates the allocation of spatial and temporal resources for them. For example, from the perspective of T_2 and $L3$, S_1 represents the spatial HW-tile allocation scheme to them.

Each Node possesses a group of HW-tiles, referred to as a **HW-tile group** (TG_i in Formula (3)), which are responsible for computing the workloads of the Node. We refer to the number of HW-tiles in the HW-tile group as **HW-tile group size**. For example, in Fig. 2, the HW-tile group of $L1$ contains HW-tile $\{1, 2\}$ and has a size of 2, while the HW-tile group of S_1 contains HW-tile $\{1, 2, 3\}$ and has a size of 3. The strategies we use to determine the size and physical position of the HW-tile group are detailed in Sec. 3.3.2.

Each Node has its own *batch size* (b_i in Formula (4)), which is the number of samples the Node process each time (represented by one colored rectangle in Fig. 2(b)). For each Leaf in the figures, its batch size is marked in parenthesis (e.g., the batch size of $L1$ and $L4$ is 2 and 4, in Fig. 2). For each Cut, one batch is divided into one or several subbatches, whose size is the batch size for each of its children. (see b_j in Formula (8)) The batch size b_i and the number of subbatches sb_i are marked in the form (b_i/sb_i) in each Cut in all figures. A Leaf Node does not need to divide subbatches, so its number of subbatches $sb_i = 1$. (Formula (7))

Each Node also contains dependency information (Formula (5), (6)). If there is a dependency $l \rightarrow l'$ in the DNN, and $l \in L_i, l' \in L_j$, then we construct a dependency $N_i \rightarrow N_j$ in the RA Tree. Node N_i will be recorded in $FROM_j$ and Node N_j will be recorded in TO_i. The dependencies can be categorized into original dependencies in the DNN and deduced dependencies (both marked in Fig. 2(a)). For example, in Fig. 2(a), there is no dependency from either $L1$ or $L2$ to $L3$, so there is no dependency from T_2 to $L3$. However, since dependency $L1 \rightarrow L4$ exists, the deduced dependency $T_2 \rightarrow L4$ also exists. For a valid RA Tree, we require all dependencies between Leaves to be from left to right (i.e., Leaves form a topological order in the DNN) to ensure the consumer of a data processes after the producer. Notice that this requirement guarantees that all dependencies, whether original or deduced, are from left to right.

3.1.3 Attributes of S Cut and T Cut. In this section, we will introduce how S and T Cut allocates spatial and temporal resources for each child.

An **S Cut** allocates different HW-tile groups to each child (Formula (9), (10)). In an S Cut, all children process in parallel or pipeline, determined by the dependencies among them. In detail, each child

processes its subbatches sequentially, and if child B has a dependency on child A, child B must start at least one subbatch later to obtain the output feature maps (ofmaps) of child A.

For example, in Fig. 3 LP, the S Cut above $L1$ and $L2$ has two subbatches with a batch size of 2, so $L1$ and $L2$ have to sequentially process two batches with size 2 on different HW-tile groups, and since there is a dependency from $L1$ to $L2$ (shown in Fig. 2(a)), $L2$ starts at the second batch of $L1$; while in Fig. 2 there is no dependency from T_2 to $L3$, so the two Nodes can start in parallel (see the third graph in Fig. 2(b)).

A **T Cut** allocates different usage duration of the same HW-tile for each child (Formula (9)). Each child of a T Cut is processed sequentially at the unit of a subbatch. In concrete, the first subbatch is processed by the Cut's children in order from left to right sequentially, then the second subbatch is processed, and then the third, ..., until the last one is processed.

For example, the bottom graph in Fig. 2(b) illustrates the processing of the children of Cut T_2: the first subbatch of $L1$ is processed, followed by the first subbatch of $L2$, and then the second subbatch of $L1$ and $L2$ is processed.

In a nutshell, S/T Cut allocates the spatial/temporal resources for its children.

Currently, the RA Tree notation and the following SET framework focus on DNN inference scenarios. Given that the training process can also be viewed as a graph with some specific operators and limitations, RA Tree notation also has the potential to be used in training scenarios. In the future, we will extend our notation and framework to DNN training scenarios.

3.2 Space Size Analysis

Our inter-layer scheduling space consists of all RA Trees satisfying the above definitions. The size of the space is enormous: For a n-layer DNN, if we only consider the tree structure and the two Cut types, there are already $O((5+2\sqrt{6})^n) \approx O(9.899^n)$ candidates (calculation procedure is provided at this link [1]); To calculate the total size of the space, we need to multiply the number of topological orders of the DNN network, and consider different subbatch sizes of each Cut, further increasing the whole space size.

3.3 RA Tree Analysis

In this section, we elaborate on how to parse an RA Tree into each layer's resource allocation scheme and the flow of data among the components of an accelerator. It is worth mentioning that the RA Tree notation is not tied to the strategies introduced in this section. RA Tree is a generic notation, allowing for the replacement of buffer management, dataflow management, and core allocation strategies to adapt to various scenarios.

3.3.1 Data Flow. We define *data flow* here as the flow of input feature maps (ifmaps), weights, and ofmaps of each layer among the components of an accelerator.

For feature maps (fmaps), if a dependency is between two layers under different children of a root T Cut, the corresponding fmaps will be sent to DRAM. Otherwise, there are two cases: (1) If the ofmaps of the former layer are consumed by the latter layer immediately, they will be sent directly to the consuming HW-tiles through NoC. (2) If the ofmaps of the former layer are not consumed by the latter layer immediately, it will be sent to DRAM for temporal buffering. For the example in Fig. 2, the ofmaps of $L1$ will be immediately consumed by $L2$. Thus $L1$'s ofmaps can be directly sent to the HW-tile group of $L2$. Since S_1 Cut and $L4$ are different children of the root T Cut, the ofmaps of the layers under S_1 Cut will be sent to DRAM first. Then $L4$ will load these ofmaps from DRAM.

For weights, if the root Node is a T Cut, the weights of all layers under each child will be loaded from DRAM and pinned on-chip for each batch of the child. If the root Node is an S Cut, all weights will always be pinned on-chip.

3.3.2 HW-tiles Allocation. As introduced above, HW-tiles allocation only occurs at each S Cut, since each child of a T Cut uses all HW-tiles of the T Cut. Specifically, the resource allocation for an S Cut is a process of deciding how many and which computing resources should each child of the S Cut own, which will be introduced below. After mastering the resource allocation strategies for a single S Cut, the resource allocation of each RA Tree can be analyzed recursively.

To equalize the processing time of each child and reduce bubble overheads, the most straightforward method is to keep the number of HW-tiles allocated for each child proportional to the child's total number of operations (ops), which is the sum of the number of ops in all layers belonging to it. However, we observe that using the total op number to represent processing time is inaccurate because of two issues: (1) Due to the inherent features of HW-tile microarchitecture, layers of different types or shapes may have different utilization on the same HW-tile even if they have the same number of ops. For example, if we schedule the first layer in ResNet-50 [19] on the Simba tile [46], we can only utilize 3 of the 8 MACs in one Vector, and the utilization will be upper-bounded by 37.5%. (2) When layers have dependencies in an S Cut, their corresponding batches of samples cannot start at the same time, creating filling and draining overheads (see $L1$ and $L2$ in Fig. 3 LP). Ideally, the reduction of utilization is $B/(B+S)$, where B is the number of subbatches, and S is the difference in starting time between the first and last children. For example, the $B/(B+S)$ of the S Cut in Fig. 3 LP is equal to $2/(2+1)$.

To take care of such two issues, we introduce an attribute, "Normalized Processing Time"(NPT), for each Node. The NPT of a layer (Leaf Node) is calculated by simulating the layer with one sample on a HW-tile. For a T Cut, we only need to sum up the NPT of its children to get its NPT. For an S Cut, the sum is divided by $B/(B+S)$ defined in issue (2) of the previous paragraph to get the NPT of the Cut. Then we can recursively define the NPT of each Node in a bottom-up manner. The remaining question is: How to divide the HW-tile group of the S Cut into subgroups for its children, so that the sizes of these subgroups are proportional to the NTPs of the children? Since the size of each subgroup must be an integer, in most cases, the division cannot be strictly proportional, resulting in an imbalance between different subgroups, which creates bubbles and reduces utilization. We will refer to this deficiency as "bubble overheads" below. The severity of this problem increases with the number and diversity of layers, and can even cause utilization to

drop by tens of percentage points. Therefore, we propose an optimal HW-tile allocation algorithm to alleviate this problem as much as possible, and formally prove the optimality of the algorithm. Due to space constraints, we place the algorithm and its proof at this link [2].

After determining the number of HW-tiles in each HW-tile group, we also need to determine which physical HW-tiles the HW-tile group corresponds to. Firstly, each HW-tile should be attached with an id for allocation. The id-attaching policy can be substituted arbitrarily based on the features of the accelerator, such as the NoC topology. For the default mesh NoC used in the experiments, we employ a modified stripe-based id-attaching strategy like Tangram [17] to ensure a fair comparison with it. For example, the HW-tiles in blue will be assigned ids from 1 to 8 in a left-right and bottom-up order (first row in Fig. 4). Then, the children of each S Cut will take physical HW-tiles owned by the Cut in a left-right order. For the example in Fig. 2, the S_1 Cut owns the HW-tiles with ids from 1 to 3. The T_1 Cut and $L3$ will take the HW-tiles with ids 1&2 and id 3, respectively.

3.4 Representation of LP and LS

In this section, we first show how to use our RA Tree notation to represent existing LP and LS patterns (Fig. 3), and then analyze their features.

Due to the limitations in the capacity of the on-chip buffer and the number of HW-tiles, current LP and LS, which have been optimized by existing works [6, 17, 38, 55, 61, 62], tend to cut a DNN into multiple segments and process them one by one. Thus, in RA Tree notation, the root Cut of their schemes is T Cut, and each child corresponds to a segment. For each segment, LP allocates different HW-tile groups for different layers, so the depth-one Cuts in an LP scheme, if any, should be S Cuts. In contrast, LS processes each layer in a segment with all the HW-tiles and switches the fmaps of

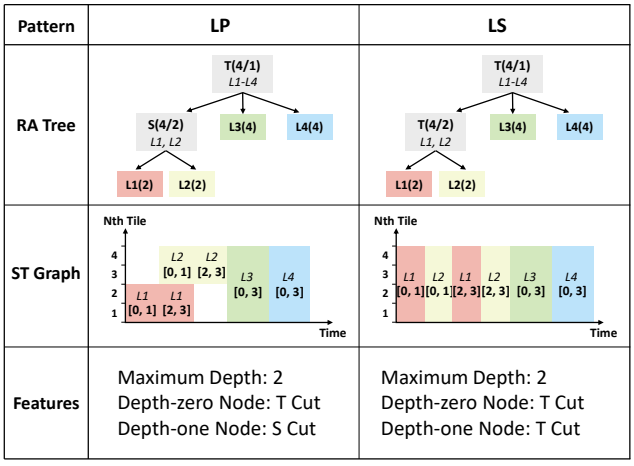

Figure 3: Representing LP and LS in RA Tree notation. The DNN topology is the same as the one in Fig. 2. A Node is called "depth-n Node" if the depth of the Node (distance to the root Node) is n. "Maximum Depth" refers to the maximal depth of the Nodes in the RA Tree.

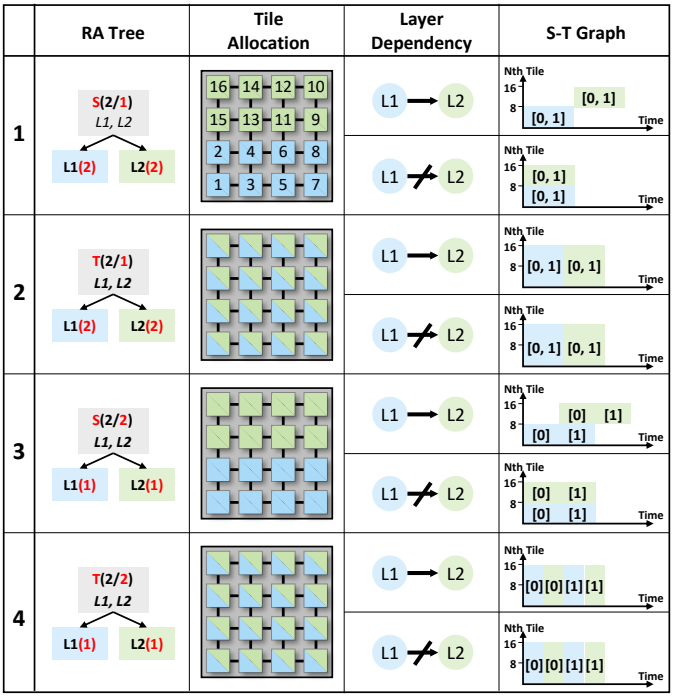

Figure 4: Trade-offs behind different Cuts and DNN topology. The first row shows an id-attaching example.

the layers with dependencies through the on-chip buffer without accessing the DRAM. Thus, the depth-one Cuts in an LS scheme, if any, should be T Cuts. From the above analysis, the scheduling space of LP and LS patterns can be calculated as $O(2^n)$[1], which is only about $\frac{1}{(2^n)^{2.22}}$ of the space defined by us. However, most of the existing works touch only a small part of the LS or LP scheduling space, which further shows the significant potential of exploring the broad scheduling space defined by us.

3.5 Trade-offs Analysis

In this section, we reveal the complex trade-offs behind different inter-layer scheduling choices with the help of our RA Tree notation. The fundamental trade-offs can be studied by analyzing a simple example, where two identical convolution layers with a workload of batch size two are scheduled to be mapped on a 16-tile accelerator with mesh NoC (most following analysis is not sensitive to NoC topology). The following analysis is applicable to most types of layers, such as fully connected layers, general matrix-to-matrix multiplication, and so on. Other types of layers can be theoretically analyzed similarly. Trade-offs behind more complex tree structures can be seen as a hierarchical combination of these fundamental trade-offs.

As shown in Fig. 4, when comparing 1 with 2 and 3 with 4, the trade-offs brought by S and T Cuts can be analyzed. As shown in the second column of Fig 4, the HW-tile group size of each layer

[1]Since each Depth-one Node corresponds to one or multiple layers, a scheme in LS/LP can be seen as dividing a list of n items (layers) into one or multiple segments, thus the total number is $2^{n-1} = O(2^n)$.

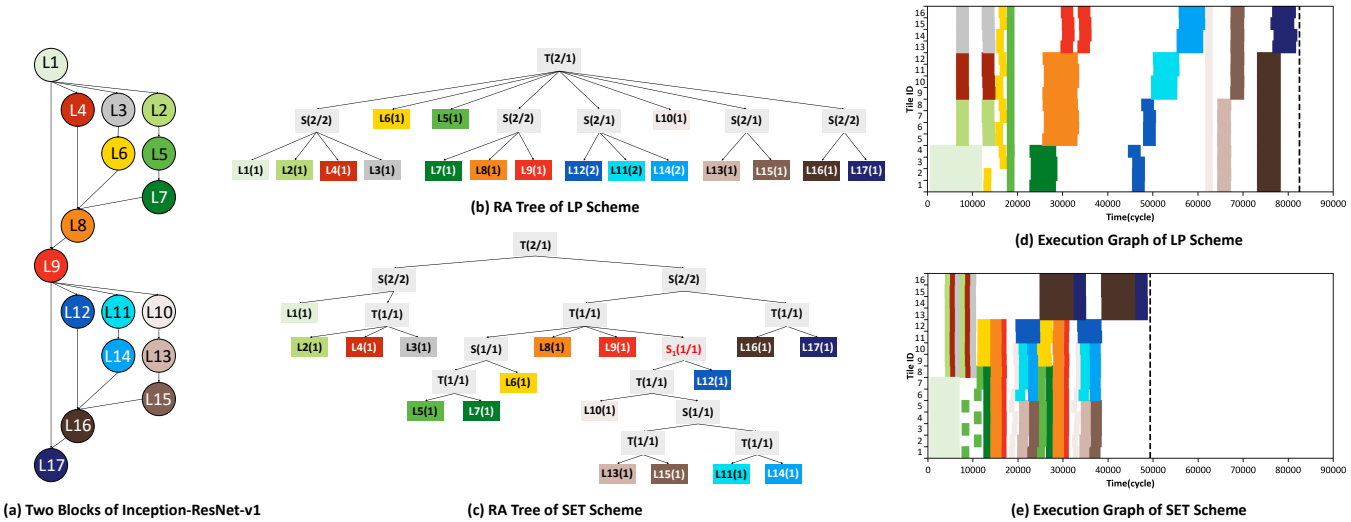

Figure 5: A practical comparison of two schemes: one explored within the space defined by our RA Trees (abbreviated as SET scheme), and the other explored by Tangram-like strategy [17], which mainly falls in LP pattern (abbreviated as LP scheme). The practical execution graph of both RA Trees are drawn by the cycle-accurate simulator of a 16-tile accelerator.

under S Cut is less than the counterpart in T Cut, which is a core contributing factor. A smaller HW-tile group for a layer brings the following benefits: (1) **less data duplication**: Partitioning a layer into multiple HW-tiles in any dimension requires copying some part of the data [16]. Thus, a smaller HW-tile group means less data duplication. For example, if we partition the output channel dimension for 8 HW-tiles, each HW-tile needs $\frac{1}{8}$ weights and entire ifmaps. When the HW-tile group size scale to 16, the system buffers one copy of the weights and 16 copies of the ifmaps for this layer; (2) **larger intra-tile exploration space**: Each layer needs to be partitioned first and then optimized by the intra-tile scheduling. Thus, a smaller HW-tile group means a larger workload for each HW-tile, facilitating intra-tile scheduling to take advantage of parallel computing resources and data locality. From the above analysis, a smaller HW-tile group reduces NoC communication costs due to (1), reduces DRAM access times due to (1), and improves intra-tile utilization and data reuse due to (2).

The S Cut also has some drawbacks: (1) **filling and draining overheads**: When the layers share a sequential dependency, some resources will be idle in the filling and draining period, which can be observed by comparing the sub-graphs of the first and third row of Fig. 4; (2) **bubble overheads**: This deficiency has been introduced when introducing NPT in Sec. 3.3.2. These deficiencies may worsen performance. The advantages and disadvantages of T Cut and S Cut are largely complementary. (3) **possibly data-fetching overheads**: If a smaller HW-tile group can no longer buffer all data of a layer, it has to fetch data from DRAM additionally, resulting in increased DRAM accesses and NoC communication costs.

By comparing 1 with 3, and 2 with 4, the trade-offs brought by subbatch size can be analyzed. A smaller subbatch size brings the following benefits: (1) **less filling and draining overheads**: a smaller subbatch size can reduce the filling and draining overheads when the layers have a sequential dependency and are cut by S Cut; (2) **less on-chip buffer usage**: A smaller subbatch size means that each HW-tile group needs to buffer fewer ifmaps, which will also bring the same benefits as the second benefits of a smaller HW-tile group. A smaller subbatch size will also influence intra-layer scheduling effects: a smaller subbatch size means less exploration space for intra-layer scheduling, which may reduce the utilization and energy efficiency of intra-layer scheduling schemes.

3.6 Case Study: Learn From a Practical Example

In this case study, we employ our cycle-accurate simulator (introduced in Sec. 4.3) to show a practical comparison between the scheme explored within the space defined by our RA Trees (abbreviated as SET scheme) and the scheme explored by SOTA Tangram-like strategy [17], which mainly falls in LP pattern (abbreviated as LP scheme) in Fig. 5. The workloads, batch size, and hardware platform are two Inception-ResNet-v1 blocks, 2, and a 16-tile accelerator, respectively. We utilize this case study to vividly show how an RA Tree scheme incorporating artful combination of S and T cuts can outperform the LP scheme.

In Fig. 5, we present the RA Trees of the schemes along with their corresponding execution graphs for all HW-tiles on the accelerator. Each HW-tile can initiate computation as soon as all the required data is available, without the need to synchronize with other HW-tiles processing the same layer. Consequently, different HW-tiles computing the same layer may begin or end their computation at distinct times. The blank regions in Fig. 5 represent idle states for the respective HW-tiles.

Overall, the SET scheme saves 41.1% latency and 31.5% energy than the LP scheme. Especially, the DRAM accesses and NoC communication costs are reduced by 61.1% and 35.3%, which contributes to most energy cost reductions. By comparing Fig. 5(d) and Fig. 5(e), it can be intuitively observed that the workload execution of the SET scheme is more compact and closely aligned. In the following,

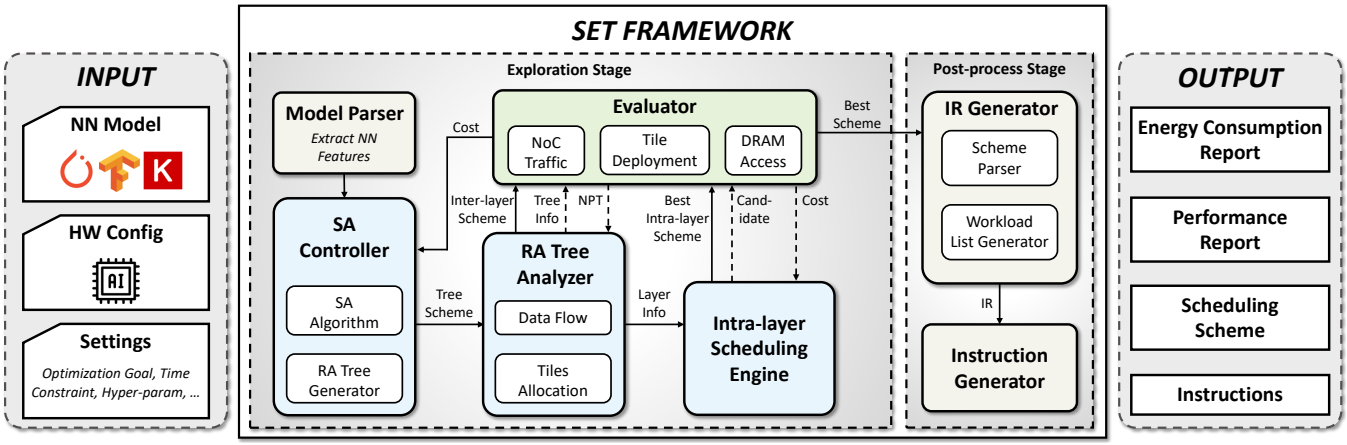

Figure 6: An overview of SET scheduling framework. Solid arrows represent the SA iteration loop, and dotted arrows represent local calls to the evaluator from each engine.

we will first analyze the reasons behind the relatively sparse workload execution of the LP scheme and then proceed to discuss how the SET scheme effectively mitigates these issues.

For LP scheme: As introduced in Sec. 3.3.2, ensuring that HW-tiles allocated to multiple layers are proportionally distributed to maintain reasonable bubble overhead is a challenging task. Consequently, we observe that the LP scheme struggles to simultaneously put more than four layers under a S Cut, as shown in Fig. 5(b). Such limitation significantly limits possible producer-consumer reuse opportunities, which leads to more DRAM accesses. Moreover, the dependencies between layers in the pipeline introduces much filling and draining overheads. As a result, filling and draining overheads (e.g., blank regions during intervals of approximately 0-7000 cycles and 45000-65000 cycles) and increased DRAM and NoC bandwidth pressure (e.g., blank regions during intervals of approximately 35000-45000 cycles) contribute to the sparser workload execution in the LP scheme, as shown in Fig. 5(d).

For the SET scheme: As shown in Fig. 5(c) and (e), layers with shared dependencies tend to be under the same T Cut, consequently utilizing the same computing cluster for computation (e.g., L5→L7, L8→L9, L11→L14, L13→L15). As a result, the ofmaps from the producer layer can be rapidly and locally transmitted to the consumer layer, reducing NoC communication costs. Furthermore, the immediate consumption of these ofmaps enhances buffer usage efficiency. Concurrently, as shown in Fig. 5(e), layers exhibiting parallel relationships are inclined to engage distinct clusters (e.g., L5 and L6, L10 and L12, L11 and L13, and L14 and L15), capitalizing on the advantages of smaller clusters discussed in Sec. 3.5 without incurring filling and draining overheads. Moreover, the strategic combination of S Cut and T Cut makes it easier to balance the HW-tile allocation for the children of S Cut, thus significantly alleviates the bubble costs associated with S Cut. For instance in Fig. 5(c), if we place L10~L15 under S_1 Cut (marked in red), the bubble costs would be substantial. However, as shown in Fig. 5(c) and (e), by combining several S and T Cuts, we can strike a sound balance in the HW-tile allocation among the children of S_1 Cut without compromising parallel opportunities and producer-consumer reuse opportunities

In a nutshell, SET scheme can achieve both benefits of S Cut and T Cut by combining them in a artful manner.

4 SET FRAMEWORK

4.1 SET Overview

As shown in Fig. 6, SET is an end-to-end DNN scheduling framework. SET inputs: (1) an NN model description file produced by high-level frameworks like Pytorch; (2) hardware configuration (NoC, DRAM, HW-tile, and so on); (3) framework settings (optimization goal and constraints, hyperparameters, and so on). After the scheduling, SET outputs: (1) energy cost and performance reports; (2) detailed scheduling scheme (3) instructions (optional).

The optimization goal of SET can be set as $Energy^n \times Delay^m$, where n and m can be set as arbitrary numbers to represent different levels of concern for power and performance. The energy and latency here are the total energy cost and latency of a batch of samples. Thus, for latency-centric and throughput-centric scenarios, we can employ small and large batch sizes to test the effectiveness of SET, respectively.

The whole scheduling process can be divided into two stages, the exploration stage and the post-processing stage. The exploration stage explores the vast scheduling space for deploying the target DNN on the target tiled accelerator. The results are then sent to the post-processing stage for IR and instructions generation.

4.2 The Exploration Stage

In the exploration stage, the inter-layer scheduling space is explored by a simulated-annealing (SA) algorithm, and the intra-layer scheduling space is explored by an exhaustive-search-based method.

Specifically, the Model Parser engine first parses the DNN description file, abstracting out the DNN graph topology information and the information of each layer in the DNN for exploration.

The initial RA Tree in SA is generated by putting all Leafs under a root T Cut in a topological order. When scheduling a L-layer

network, the SA algorithm conducts $N = \beta L$ iterations, where β is a hyperparameter. And in iteration n, the temperature is $T_n = T_0 \frac{1-\frac{n}{N}}{1+\alpha\frac{n}{N}}$, where T_0 is the initial temperature and α is the cooling speed. The design guarantees that the final temperature T_N will be 0. In each iteration, the SA Controller will randomly choose an operation (Fig. 7) and use the operation to change the original RA Tree into a new RA Tree. Then the new RA Tree will be sent to the RA Tree Analyzer for analyzing the flow of data and each layer's HW-tile allocation scheme. The type and dimension information of each layer and its allocated HW-tile group will be sent to the Intra-layer Scheduling Engine. Then, the Intra-layer Scheduling Engine will explore the intra-layer scheduling space by searching all partition, tiling, and loop order candidates (dotted arrow in Fig. 6). Finally, the optimal solution explored by the Intra-layer Scheduling Engine together with the information analyzed by the RA Tree Analyzer will be sent to the Evaluator (see Sec. 4.3) for an overall evaluation, whose results will be sent to the SA Controller. If the new cost c' is higher than the cost c of the previous RA Tree, the scheme will be accepted with probability $p = e^{\frac{c-c'}{cT_n}}$, where T_n is the temperature of the current iteration; otherwise the modified RA Tree will always be accepted.

Given that the RA Tree Analyzer has been introduced in Sec. 3.3, we only introduce the SA operators and Intra-layer Scheduling Engine below.

4.2.1 SA Operators. We develop six operators to change the RA Tree in each iteration. With these operators, any two trees in this space can be transformed into each other in finite steps, which is an important property to ensure that simulated annealing can find a near-optimal scheme.

As shown in Fig. 7, the operators will be introduced as follows:
OP1: Randomly choose two adjacent Leaves and then exchange them. This operation requires the two layers to have no direct or transitive dependency. Adjacency here means adjacency in the order of all the layers in the in-order traversal.
OP2: Randomly choose a Leaf and move it to another Cut that shares the same parent or grandparent Node.
OP3: Randomly choose a Cut, then randomly choose a group of consecutive children to constitute a new Cut. The attributes of the Cut are generated randomly.
OP4: Randomly choose a non-root Cut and delete it, then place its children into its parent Cut.
OP5, 6: Randomly choose a Cut and increase/decrease its subbatch number by random times. As a result, its children's batch size will decrease/increase with the same ratio.

The six operators ensure that the newly generated RA trees will not violate the dependency restrictions introduced in Sec. 3.1.

4.2.2 Intra-layer Scheduling Engine. Since intra-layer scheduling has been well studied by existing works, as introduced in Sec. 2.2, we adopt the classical strategies in this part. Firstly, we employ a flexible partition strategy [16] to explore how to partition each layer into smaller workloads and distribute them to each HW-tile in the corresponding HW-tile group. Secondly, for each partition scheme, we employ a search-based intra-tile dataflow exploration strategy [17, 32, 42] to explore the loop order and tiling size.

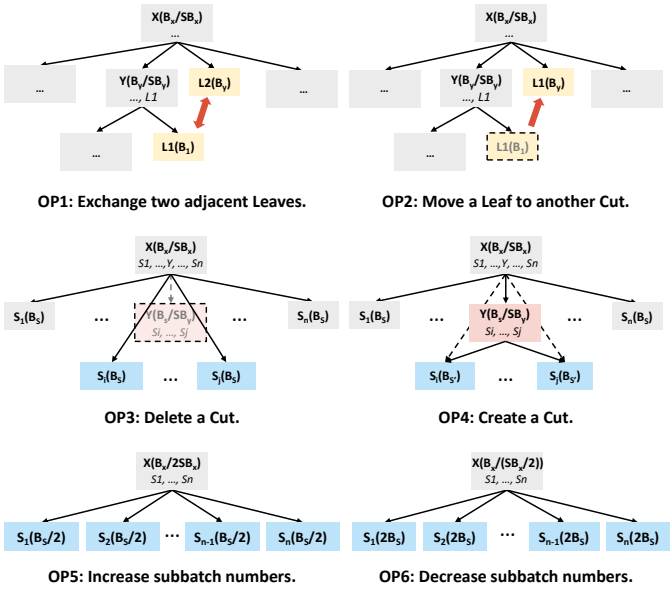

Figure 7: SA operators. The gray and colored Nodes represent unchanged and changed Nodes, respectively.

4.3 Evaluator

The default Evaluator is basically based on a scalable architecture template with a mesh NoC and NVDLA-style HW-tiles, which is also the architecture of our test chip mentioned in the following. The Evaluator can estimate the cost of deploying different DNNs on tiled accelerators in different configurations using different generated scheduling schemes.

Specifically, based on the inter-layer scheduling scheme, the evaluator can analyze the workload distribution of all HW-tiles and the data dependencies between HW-tiles or between HW-tile and DRAM. Then, it can figure out the amount of data transmission on each link of NoC and the access pattern of DRAM. Based on the intra-layer scheduling scheme, it can figure out the access times of buffers at all levels and the operation times of MACs with different precision in each HW-tile. Based on the analysis results, the latency is evaluated by a fast simulator; the total energy cost E_{tot} is calculated by

$$E_{tot} = \sum_{i=0}^{a}\sum_{j=0}^{1} E_{D_j} \times A_{D_{i,j}} + \sum_{k=0}^{c} E_{hop} \times N_k + \sum_{l=0}^{d} E_{T_l}$$

E_{D_j} means the energy cost per access of reading or writing the DRAM. $A_{D_{i,j}}$ means the reading or writing times of the ith DRAM. E_{hop} means the energy cost per hop of transmitting a flit of data from one router to its neighbor router. N_k means the number of flits that pass the kth link. E_{T_l} means the energy cost of the lth HW-tile, which includes the access energy cost of buffers at all levels, MACs with different precision, and intra-tile communication.

A 4-tile test chip is being designed to verify our scalable architecture, hardware system design, and compiling flow in preparation for designing a larger-scale accelerator. Based on this test chip, we also develop a scalable cycle-accurate simulator. Because the cycle-accurate simulator is much slower than the fast simulator

used in the exploration stage, it cannot be employed to explore the vast scheduling space. We employ this cycle-accurate simulator to show a practical scheduling example introduced in Sec. 3.6.

4.4 Post-processing Stage and SET Portability

The RA Tree notation can be applied to tiled accelerators with different architectures that satisfy the basic requirements introduced in Sec. 2.1. Therefore, SET is also designed to be highly portable to take full advantage of the notation's universality potential.

When porting SET to a new tiled accelerator, the Evaluator and Instruction Generator should be substituted. The Evaluator only needs to satisfy the scheduling engine's calling interface and give the scheme's evaluation results. Moreover, in order to reduce the difficulty of replacing the Instruction Generator, we develop an IR Generator Engine. This engine first parses the explored scheme and makes some analysis, then generate a workload list for each HW-tile. Each entry of the list records: (1) workload's attributes, such as data dimension, layer information, buffer requirement, and so on; (2) data sources and destinations; (3) intra-tile computing information. SET intra-tile optimization tool allows the users to explore classic architectures, such as NVDLA-style [41] HW-tile and Eyeriss-style [9] HW-tile. If the user wants to customize a new HW-tile architecture, the intra-tile scheduling engine, which is a part of the Intra-layer Scheduling Engine introduced in Sec. 4.2.2, will also need to be substituted. The whole deploying flow based on SET has been developed for our test chip and future larger-scale accelerator.

5 EVALUATION

5.1 Experiment Setup

5.1.1 Hardware Configuration. To thoroughly test the effects of SET, we consider two hardware platforms in the evaluation: a 16-tile (4×4) edge platform and a 144-tile (12×12) cloud or autonomous driving platform. The optimization goal of the edge and cloud platforms are E^2D and ED^2 to show their prior concern on power and performance, respectively. Both experimental platforms employ TSMC 12nm process and run at 1GHz. Both platforms share the same NVDLA-style HW-tile with 1024 int8 MACs and 1MB global buffer. The default total DRAM bandwidth is set as 0.5GB/s per 1 TOPs for each platform.

The unit energy costs of different operations used for energy cost evaluation introduced in Sec. 4.3 are as follows. The mesh NoC hop and 8-bit MAC are estimated at 0.7 pJ/bit and 0.018 pJ/op. The DRAM energy cost is extracted from the datasheet [24] at 7.5pJ/bit. For the register files and the SRAM buffers at different capacities, we employ Memory compiler [5] to generate their modules and directly acquire their energy cost per access.

5.1.2 Workloads. To thoroughly test SET effects and analyze the trade-offs behind different across-layer scheduling schemes, we scale batch size from 1 to 64, covering latency-sensitive scenarios to throughput-centric scenarios as introduced in Sec. 4.1. In our experiments, ResNet-50 [19], GoogLeNet [49], Inception-ResNet-v1 [48], PNASNet [36], and Transformer [53] as workloads.

5.1.3 Baselines. We choose the SOTA open-source Tangram scheduling framework [17] as our first baseline. For inter-layer scheduling, Tangram first employs a Dynamic Programming (DP) algorithm to cut DNNs into segments under LP patterns. Throughout the experiments, all optimization options in Tangram are turned on, and all hardware configurations of the two frameworks are kept the same (we implement an Eyeriss-style HW-tile in our evaluator).

5.1.4 SA Hyperparameters. For the SA hyperparameters mentioned in Sec. 4.2, we set $T_0 = 0.07, \alpha = 8, \beta = 100$ in the following experiments. Increasing T_0 and β for the SA algorithm will result in better RA Trees, but at a cost of longer searching time. Thus the hyperparameter can be determined according to the time budget of specific scenario.

To better study the pros and cons of each individual pattern, we use SET to explore LS and LP separately and set them as another baseline. Comparisons with such baselines can better prove the benefits of exploring the whole inter-layer scheduling space and provide vast insights into LS, LP, and the newly defined inter-layer scheduling space.

5.2 Comparisons with Tangram

Since Tangram does not support Inception-ResNet, PNASNet, and Transformer, the comparisons are conducted with the remaining workloads. Moreover, Tangram cannot support E^2D and ED^2 optimization goals. Thus we choose EDP as the optimization goal in the comparison on both platforms.

5.2.1 Validation. In order to ensure the fairness of the comparison of the inter-layer optimization effects between the two frameworks, we first test the intra-layer scheduling engines and the evaluators of the two frameworks by processing each layer of a DNN separately. Results show that, on average, the error of EDP is 3% for all workloads and batch sizes. Considering the improvement of SET over Tangram, we believe this error is acceptable.

5.2.2 Results. Fig. 9 shows the overall comparisons between Tangram and SET. On average, across all platforms, batch sizes, and workloads, SET simultaneously achieves 1.78× performance improvement and 13.2% energy cost reduction compared with Tangram, yielding a reduction of 51.2% in total EDP. The following compares the two frameworks step by step. In terms of intra-layer scheduling, the comparable effects of the intra-layer scheduling engines in these two frameworks have been demonstrated in Sec. 5.2.1. For inter-layer scheduling, SET employs an SA algorithm to explore a space with $O(9.899^n)$ schemes (calculated in Sec. 3.2), while Tangram employs a DP algorithm to exhaustively explore a space that is a subset of the space of LS and LP patterns, and these patterns contain much fewer schemes ($O(2^n)$, calculated in Sec. 3.4). The DP algorithm is more likely to yield a better solution than SA in the same space due to its capacity to explore the space exhaustively. The above analysis and comparisons demonstrate that: (1) The larger scheduling space depicted by our RA Tree notation, which contains more efficient scheduling schemes, is the decisive factor in achieving performance and energy efficiency gains; (2) Although this extremely large space no longer allows SET to use exhaustive search algorithms such as DP, our specifically-designed

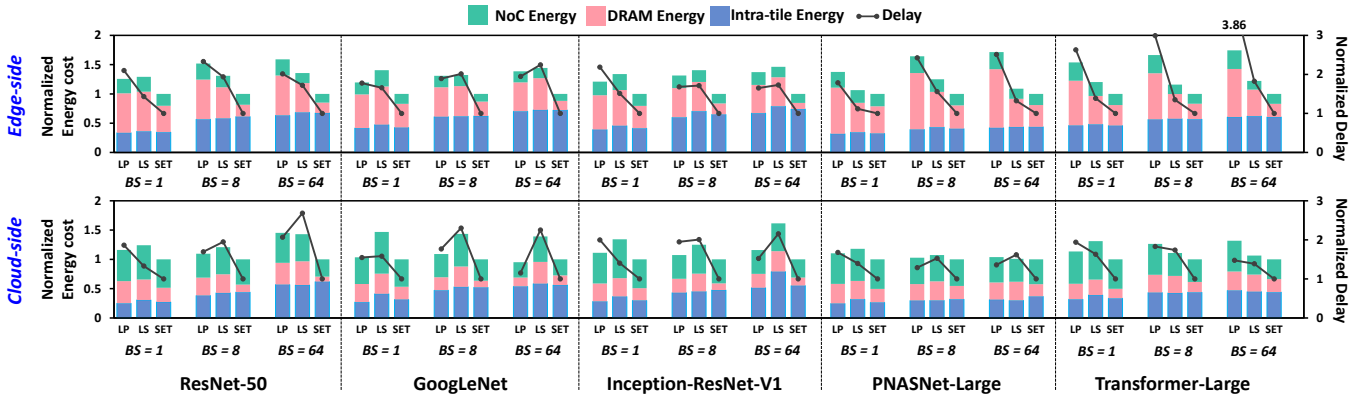

Figure 8: Comparisons among LP, LS, and SET with different batch sizes (BS), workloads, and hardware platforms. Each row takes the same platform, and each column takes the same workload.

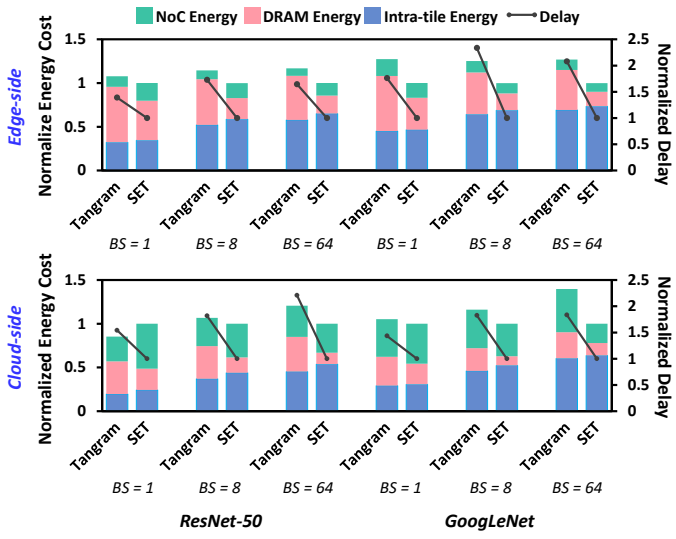

Figure 9: Comparison with Tangram

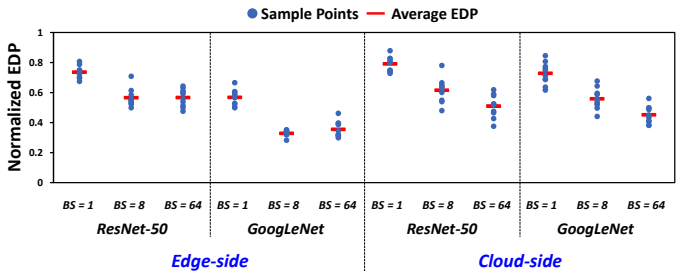

Figure 10: Variance of SET improvements. The SA algorithm is evaluated 10 times on each setting, and the EDP on each evaluation and their average are plotted above. The Y-axis is the normalized EDP, where the EDP of Tangram is normalized to 1.

SA algorithm can still ensure an efficient exploration of the space and yield good solutions.

Meanwhile, we also compared the end-to-end running time of Tangram and SET. The experiments of both frameworks are performed on an AMD Ryzen-7 5800X CPU with 4 threads for each case. Results show that the average running time of SET is 706.9 seconds over all cases, which is about 68x faster than Tangram. The main reasons for the faster speed are that: (1) we equip SET with a better framework structure and implement its coding more efficiently; (2) The well-designed SA operators enable the exploration to converge to an efficient scheduling scheme quickly.

5.2.3 Variance of SET Improvements. To study the variance of the improvements of SET across different mappings, we run SA 10 times with different seeds for each situation in Fig. 9, and plot the results in Fig. 10. On average across these ten situations, the best-case and worst-case reduces 53.9% and 35.2% EDP compared with Tangram, while the average reduces 45.4%. Also, every sample reduces some EDP compared with Tangram. This demonstrate the substantial and stable impact of SET, despite the inherent randomness of the SA algorithm.

5.3 Comparisons with LS and LP

5.3.1 Analysis of SET. Fig. 8 shows the overall comparisons among LS, LP, and SET. Across all platforms, batch sizes, and workloads, SET simultaneously achieves an average of 1.90× and 1.68× performance improvement and 21.7% and 21.5% energy reduction over LP and LS, respectively.

Essentially, exploring our space by SET is exploring how to adjust the hierarchical tree structure to leverage the strengths of each Cut, avoid its weaknesses, and try to achieve excellent balance. As a comparison, the maximum depth of LS and LP patterns is two, and the type of depth-one Cut is limited. Thus, though SET explores them fairly, they struggle to strike a good balance.

5.3.2 Analysis of LP. As introduced in Sec. 3.5, a major drawback of S Cut (also LP) is the overheads of filling and draining, so DNNs with more parallel layers, such as GoogLeNet and Inception-ResNet, allow LP to take advantage of the advantages of S Cut without being

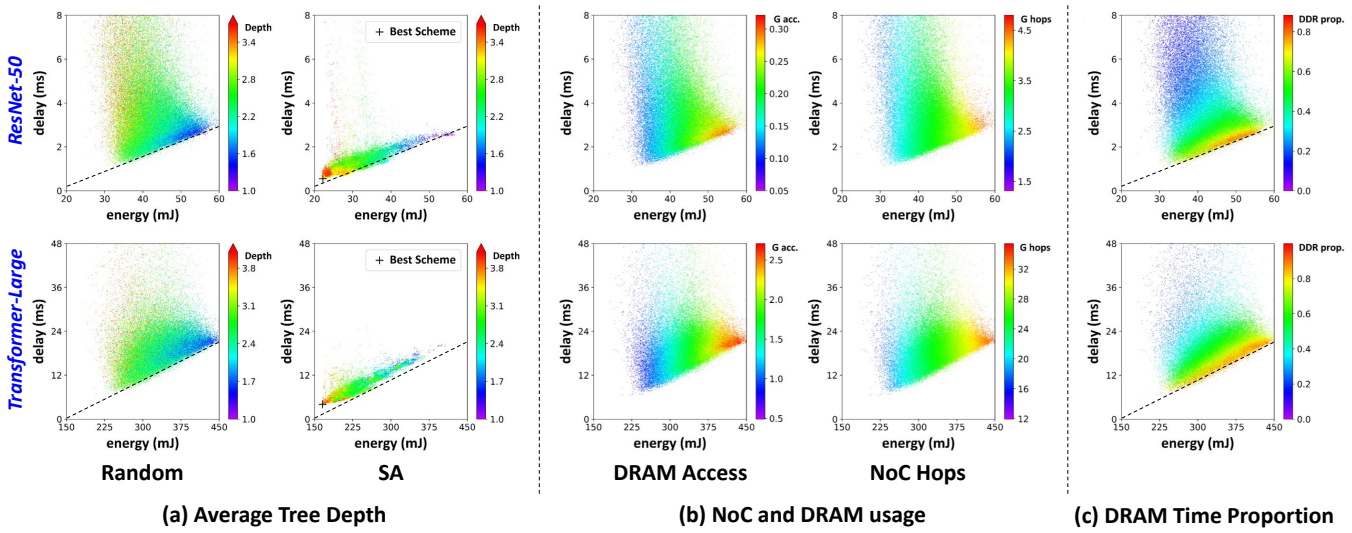

Figure 11: Scheduling space depiction. Each colored dot represents an RA Tree under the cloud platform and batch size 8. (a) The average tree depth of random RA Trees and RA Trees on the search path of SA. Depth is averaged among the depth of all Leaves. (b) The DRAM access and NoC hops of random RA Trees. (c) The proportion of DRAM time (defined in Sec. 6.1 (2)) over total delay of random RA Trees. All "RANDOM" figures show the same set of schemes, but exhibit different features.

affected by its disadvantages. This is also why LP performs better on GoogLeNet and Inception-ResNet than ResNet with fewer parallel layers. Although PNASNet and Transformer also have many parallel layers, LP performs poorly on them due to another main drawback of S Cut, bubble overheads. Their layers are much smaller and more diverse, so even with our optimal allocation algorithm 3.3.2, it is difficult for LP to find a well-balanced HW-tile allocation scheme for a large group of layers. Therefore, the pipeline length is significantly limited, and the on-chip buffer cannot be well utilized, worsening performance and energy efficiency. The severity of this problem can be greatly alleviated with the increase in the number of HW-tiles, which is reflected in the fact that LP performs better on the cloud platform that the edge one. The above analysis is also the reason why LP performs worse than LS in scheduling PNASNet and Transformer on edge-side platforms as batch sizes increase.

5.3.3 Analysis of LS. It is observed that LS performs much worse on the cloud platform than the edge platform because the disadvantages of LS, which are also the advantages of LP introduced in Sec. 3.5, are much more serious on the cloud platform. As shown in Fig. 8, using all HW-tiles in the cloud platform to compute each layer incurs significant NoC communication and DRAM access overheads, worsening performance and energy efficiency. Moreover, the parallel dimension of each layer is also limited, making it difficult for LS to use a large number of parallel resources on the cloud platform, further deteriorating performance.

6 DISCUSSION

6.1 Scheduling Space Analysis

In this case study, we employ SET framework to analyze the structure of the whole scheduling space. As shown in Fig. 11, we randomly select schemes from the scheduling space and use SET to evaluate several metrics, including tree depth, DRAM access, NoC hops, and DRAM time proportion. For comparison, we also plot the search path of our SA algorithm in the right part of Fig. 11 (a). From the figure, we make the following observations:

(1) As shown in Fig. 11 (a), schemes with deeper RA Trees tend to cost less energy, but have a bigger variance and smaller lower bound on its delay. The reason is as follows:

Since intra-tile energy are very close between different schemes (shown in Fig. 8), the variation of the total energy is determined by the cost of DRAM access and NoC hops. A deeper RA Tree can capture more on-chip reuse within its structure, significantly reducing DRAM access times and NoC hops (see Fig. 11 (a) and (b)), thus reducing much energy.

However, the latency of a scheme is not only bounded by DRAM and NoC bandwidth, but also affected by the filling&draining and bubble overheads in S Cuts as introduced in Sec. 4. Although a deeper tree can reduce DRAM and NoC bandwidth pressure, randomly using S Cuts increases the risk of introducing more fill&drain costs and bubbles into the scheme. Thus compared with schemes with shallower RA trees, only deep RA Trees which balance workloads well can have both less DRAM&NoC usage, less filling&draining overheads, and bubbles simultaneously, and enjoy much less delay. As shown in the right part of Fig. 11, our SA successfully finds such deeper RA Trees with much lower delays, proving the effectiveness of our SA algorithm.

(2) In both ResNet-50 and Transformer-Large, there exists a "DRAM bound" in the plot, where all dots are above this bound (see the dotted line in Fig. 11). This marks the bound of DRAM bandwidth. To demonstrate this, we introduce the "DRAM Time" of each scheme, which is calculated by dividing the total DRAM access by the DRAM bandwidth. This is the ideal total access time of DRAM. As shown in Fig. 11 (c), we study the proportion of "DRAM Time" in the total delay of each scheme. A proportion near 1 means

DRAM bandwidth is fully utilized most of the time, and the scheme is close to the DRAM bound. From Fig. 11 (c), we can see that the dots close to the dotted line has the biggest DRAM Time proportion, ranging from 0.8 to 0.98. This marks that schemes right above the dotted line fully utilize DRAM, and its delay cannot go down any further and pass the line, which is the meaning of this "DRAM bound".

7 RELATED WORK

Existing works on inter-layer scheduling for tiled accelerators mainly focus on optimizing LS and LP patterns, a small subset of the space defined by us. For LS, Efficient Scheduling [62] and NASA [38] study how to cut a DNN graph into segments to fully use on-chip buffer and reduce DRAM access times. Shortcut [6] proposes a buffer management architecture and strategy to optimize the shortcut structure of DNNs. For LP, Tangram [17] and Atomic [61] take different approaches to realizing a finer-grained layer pipeline, optimizing for the filling and draining overheads of LP. In fact, most of these specific optimizations on LS or LP are compatible with SET and can be employed to optimize the attributes of S Cut or T Cut to further improve SET effects. However, these works did not clearly define inter-layer scheduling space, significantly limiting the optimization opportunities and hindering the understanding of inter-layer scheduling choices and corresponding consequences.

8 CONCLUSION

This work proposes a universal notation to define the problem and exploration space of inter-layer scheduling on tiled accelerators. In concrete, this work first proposes a universal notation, RA Tree, and then dives into bridging the notation with hardware behaviors and analyzing the complex trade-offs behind different scheduling choices. Based on the space definition mentioned above, we develop an end-to-end and highly-portable framework, SET, to explore the whole scheduling space for tiled accelerators. We conduct vast experiments on different DNNs, batch sizes, and accelerator scales. Results show SET achieves significant energy-efficiency & performance improvements over the SOTA Tangram scheduling framework and existing patterns.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2022YFB2804103) and the Key Research and Development Program of Shaanxi (2021ZDLGY01-05).

REFERENCES

[1] [n. d.]. Calculation of Our Inter-layer Space Size. https://github.com/SET-ISCA2023/Tile-Alloc-Algorithm/blob/master/Space_Size_Calculation.pdf.
[2] [n. d.]. The Optimal Tile Allocation Algorithm. https://github.com/SET-ISCA2023/Tile-Alloc-Algorithm/blob/master/Optimal_Tile_Allocation_Algorithm.pdf.
[3] Manoj Alwani, Han Chen, Michael Ferdman, and Peter A. Milder. 2016. Fused-layer CNN accelerators. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2016, Taipei, Taiwan, October 15-19, 2016*. IEEE Computer Society, 22:1–22:12. https://doi.org/10.1109/MICRO.2016.7783725
[4] Apple. [n. d.]. Apple A15 Bionic. https://en.wikipedia.org/wiki/Apple_A15.
[5] ARM. [n. d.]. ARM Artisan. https://developer.arm.com/downloads-beta/search?term=artisan.
[6] Arash AziziMazreah and Lizhong Chen. 2019. Shortcut Mining: Exploiting Cross-Layer Shortcut Reuse in DCNN Accelerators. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 94–105. https://doi.org/10.1109/HPCA.2019.00030
[7] Cambricon. [n. d.]. MLU290-M5. https://www.cambricon.com/index.php?m=content&c=index&a=lists&catid=340.
[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967
[9] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 367–379. https://doi.org/10.1109/ISCA.2016.40
[10] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. 2017. Dual Path Networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 4467–4475. https://proceedings.neurips.cc/paper/2017/hash/f7e0b956540676a129760a3eae309294-Abstract.html
[11] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2014, Cambridge, United Kingdom, December 13-17, 2014*. IEEE Computer Society, 609–622. https://doi.org/10.1109/MICRO.2014.58
[12] Yu-Hsin Chen, Tien-Ju Yang, Joel S. Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232
[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423
[14] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, Deborah T. Marr and David H. Albonesi (Eds.). ACM, 92–104. https://doi.org/10.1145/2749469.2750389
[15] Clément Farabet, Berin Martini, B. Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2011, Colorado Springs, CO, USA, 20-25 June, 2011*. IEEE Computer Society, 109–116. https://doi.org/10.1109/CVPRW.2011.5981829
[16] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Yunji Chen, Olivier Temam, and John Carter (Eds.). ACM, 751–764. https://doi.org/10.1145/3037697.3037702
[17] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 807–820. https://doi.org/10.1145/3297858.3304014
[18] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 580–587. https://doi.org/10.1109/CVPR.2014.81
[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778. https://doi.org/10.1109/CVPR.2016.90
[20] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. 2021. Mind mappings: enabling efficient algorithm-accelerator mapping space search. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, Tim Sherwood, Emery D. Berger, and Christos Kozyrakis (Eds.). ACM, 943–958. https://doi.org/10.1145/3445814.3446762

[21] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.* 29, 6 (2012), 82–97. https://doi.org/10.1109/MSP.2012.2205597

[22] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[23] Qijing Huang, Aravind Kalaiah, Minwoo Kang, James Demmel, Grace Dinh, John Wawrzynek, Thomas Norell, and Yakun Sophia Shao. 2021. CoSA: Scheduling by Constrained Optimization for Spatial Accelerators. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 554–566. https://doi.org/10.1109/ISCA52012.2021.00050

[24] Innosilicon. [n. d.]. Innosilicon GDDR6. https://www.innosilicon.com/html/ip-solution/14.html.

[25] Michael James, Marvin Tom, Patrick Groeneveld, and Vladimir Kibardin. 2020. ISPD 2020 Physical Mapping of Neural Networks on a Wafer-Scale Deep Learning Accelerator. In *ISPD 2020: International Symposium on Physical Design, Taipei, Taiwan, March 29 - April 1, 2020, delayed to September 20-23, 2020*, William Swartz and Jens Lienig (Eds.). ACM, 145–149. https://doi.org/10.1145/3372780.3380846

[26] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter C. Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David A. Patterson. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[27] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 1–12. https://doi.org/10.1145/3079856.3080246

[28] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*. IEEE, 622–636. https://doi.org/10.1109/MICRO50266.2020.00058

[29] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*. IEEE, 44:1–44:9. https://doi.org/10.1145/3400302.3415639

[30] Simon Knowles. 2021. Graphcore. In *IEEE Hot Chips 33 Symposium, HCS 2021, Palo Alto, CA, USA, August 22-24, 2021*. IEEE, 1–25. https://doi.org/10.1109/HCS52781.2021.9567075

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[32] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 754–768. https://doi.org/10.1145/3352460.3358252

[33] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar (Eds.). ACM, 461–475. https://doi.org/10.1145/3173162.3173176

[34] Juhyoung Lee, Dongjoo Shin, Jinsu Lee, Jinmook Lee, Sanghoon Kang, and Hoi-Jun Yoo. 2019. A Full HD 60 fps CNN Super Resolution Processor with Selective Caching based Layer Fusion for Mobile Devices. In *2019 Symposium on VLSI Circuits, Kyoto, Japan, June 9-14, 2019*. IEEE, 302. https://doi.org/10.23919/VLSIC.2019.8778104

[35] Sean Lie. 2021. Multi-Million Core, Multi-Wafer AI Cluster. In *IEEE Hot Chips 33 Symposium, HCS 2021, Palo Alto, CA, USA, August 22-24, 2021*. IEEE, 1–41. https://doi.org/10.1109/HCS52781.2021.9567153

[36] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive Neural Architecture Search. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11205)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 19–35. https://doi.org/10.1007/978-3-030-01246-5_2

[37] Liqiang Lu, Naiqing Guan, Yuyue Wang, Liancheng Jia, Zizhang Luo, Jieming Yin, Jason Cong, and Yun Liang. 2021. TENET: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 720–733. https://doi.org/10.1109/ISCA52012.2021.00062

[38] Xiaohan Ma, Chang Si, Ying Wang, Cheng Liu, and Lei Zhang. 2021. NASA: Accelerating Neural Network Design with a NAS Processor. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 790–803. https://doi.org/10.1109/ISCA52012.2021.00067

[39] Huiyu Mo, Wenping Zhu, Wenjing Hu, Guangbin Wang, Qiang Li, Ang Li, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. 9.2A 28nm 12.1TOPS/W Dual-Mode CNN Processor Using Effective-Weight-Based Convolution and Error-Compensation-Based Prediction. In *IEEE International Solid-State Circuits Conference, ISSCC 2021, San Francisco, CA, USA, February 13-22, 2021*. IEEE, 146–148. https://doi.org/10.1109/ISSCC42613.2021.9365943

[40] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman P. Jouppi, and David A. Patterson. 2020. Google's Training Chips Revealed: TPUv2 and TPUv3. In *IEEE Hot Chips 32 Symposium, HCS 2020, Palo Alto, CA, USA, August 16-18, 2020*. IEEE, 1–70. https://doi.org/10.1109/HCS49909.2020.9220735

[41] Nvidia. 2017. NVDLA Deep Learning Accelerator. http://nvdla.org..

[42] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel S. Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2019, Madison, WI, USA, March 24-26, 2019*. IEEE, 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[43] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel S. Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 27–40. https://doi.org/10.1145/3079856.3080254

[44] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 779–788. https://doi.org/10.1109/CVPR.2016.91

[45] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 14–26. https://doi.org/10.1109/ISCA.2016.12

[46] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3352460.3358302

[47] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*. IEEE Computer Society, 541–552. https://doi.org/10.1109/HPCA.2017.55

[48] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial*

Intelligence, February 4-9, 2017, San Francisco, California, USA, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 4278–4284. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[50] Emil Talpes, Douglas Williams, and Debjit Das Sarma. 2022. DOJO: The Microarchitecture of Tesla's Exa-Scale Computer. In *2022 IEEE Hot Chips 34 Symposium, HCS 2022, Cupertino, CA, USA, August 21-23, 2022*. IEEE, 1–28. https://doi.org/10.1109/HCS55958.2022.9895534

[51] Zhanhong Tan, Hongyu Cai, Runpei Dong, and Kaisheng Ma. 2021. NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 1013–1026. https://doi.org/10.1109/ISCA52012.2021.00083

[52] Jasmina Vasiljevic, Ljubisa Bajic, Davor Capalija, Stanislav Sokorac, Dragoljub Ignjatovic, Lejla Bajic, Milos Trajkovic, Ivan Hamer, Ivan Matosevic, Aleksandar Cejkov, Utku Aydonat, Tony Zhou, Syed Zohaib Gilani, Armond Paiva, Joseph Chu, Djordje Maksimovic, Stephen Alexander Chin, Zahi Moudallal, Akhmed Rakhmati, Sean Nijjar, Almeet Bhullar, Boris Drazic, Charles Lee, James Sun, Kei-Ming Kwong, James Connolly, Miles Dooley, Hassan Farooq, Joy Yu Ting Chen, Matthew Walker, Keivan Dabiri, Kyle Mabee, Rakesh Shaji Lal, Namal Rajatheva, Renjith Retnamma, Shripad Karodi, Daniel Rosen, Emilio Munoz, Andrew Lewycky, Aleksandar Knezevic, Raymond Kim, Allan Rui, Alexander Drouillard, and David Thompson. 2021. Compute Substrate for Software 2.0. *IEEE Micro* 41, 2 (2021), 50–55. https://doi.org/10.1109/MM.2021.3061912

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[54] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Ross Pinckney, Priyanka Raina, Yanqing Zhang, Brian Zimmer, William J. Dally, Joel S. Emer, Stephen W. Keckler, and Brucek Khailany. 2019. MAGNet: A Modular Accelerator Generator for Neural Networks. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4-7, 2019*, David Z. Pan (Ed.). ACM, 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942127

[55] Xingbin Wang, Boyan Zhao, Rui Hou, Amro Awad, Zhihong Tian, and Dan Meng. 2021. NASGuard: A Novel Accelerator Architecture for Robust Neural Architecture Search (NAS) Networks. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 776–789. https://doi.org/10.1109/ISCA52012.2021.00066

[56] Ofri Wechsler, Michael Behar, and Bharat Daga. 2019. Spring Hill (NNP-I 1000) Intel's Data Center Inference Chip. In *2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, August 18-20, 2019*. IEEE, 1–12. https://doi.org/10.1109/HOTCHIPS.2019.8875671

[57] Yannan Nellie Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. 2022. Sparseloop: An Analytical Approach To Sparse Tensor Accelerator Modeling. In *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022*. IEEE, 1377–1395. https://doi.org/10.1109/MICRO56248.2022.00096

[58] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. 2021. HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation. In *48th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, June 14-18, 2021*. IEEE, 1055–1068. https://doi.org/10.1109/ISCA52012.2021.00086

[59] Andrew Yang. 2019. Deep Learning Training At Scale Spring Crest Deep Learning Accelerator (Intel® Nervana™ NNP-T). In *2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, August 18-20, 2019*. IEEE, 1–20. https://doi.org/10.1109/HOTCHIPS.2019.8875643

[60] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, James R. Larus, Luis Ceze, and Karin Strauss (Eds.). ACM, 369–383. https://doi.org/10.1145/3373376.3378514

[61] Shixuan Zheng, Xianjue Zhang, Leibo Liu, Shaojun Wei, and Shouyi Yin. 2022. Atomic Dataflow based Graph-Level Workload Orchestration for Scalable DNN Accelerators. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022, Seoul, South Korea, April 2-6, 2022*. IEEE, 475–489. https://doi.org/10.1109/HPCA53966.2022.00042

[62] Shixuan Zheng, Xianjue Zhang, Daoli Ou, Shibin Tang, Leibo Liu, Shaojun Wei, and Shouyi Yin. 2020. Efficient Scheduling of Irregular Network Structures on CNN Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3408–3419. https://doi.org/10.1109/TCAD.2020.3012215

[63] Haozhe Zhu, Bo Jiao, Jinshan Zhang, Xinru Jia, Yunzhengmao Wang, Tianchan Guan, Shengcheng Wang, Dimin Niu, Hongzhong Zheng, Chixiao Chen, Mingyu Wang, Lihua Zhang, Xiaoyang Zeng, Qi Liu, Yuan Xie, and Ming Liu. 2022. COMB-MCM: Computing-on-Memory-Boundary NN Processor with Bipolar Bitwise Sparsity Optimization for Scalable Multi-Chiplet-Module Edge Machine Learning. In *IEEE International Solid-State Circuits Conference, ISSCC 2022, San Francisco, CA, USA, February 20-26, 2022*. IEEE, 1–3. https://doi.org/10.1109/ISSCC42614.2022.9731657

[64] Brian Zimmer, Rangharajan Venkatesan, Yakun Sophia Shao, Jason Clemons, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Ross Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel S. Emer, C. Thomas Gray, Stephen W. Keckler, and Brucek Khailany. 2020. 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* 55, 4 (2020), 920–932. https://doi.org/10.1109/JSSC.2019.2960488

A ARTIFACT APPENDIX

A.1 Abstract

This artifact appendix section describes how to access the artifacts of the SET framework introduced in Sec. 4. Experiments in Sec. 5 and Sec. 6 are based on this framework.

A.2 Artifact check-list (meta-information)

- **Algorithm: Simulated Annealing (SA)**
- **Program: C++, Python (only for data collection)**
- **Compilation: by Makefile**
- **Metrics: Cost function (ED, ED^2, E^2D, etc.)**
- **Output: The optimized RA Tree produced by SET framework, including its performance and energy cost report (we choose Fig. 9 as our main results to reproduce in the AEs). (optional) An IR for the optimized RA Tree, which records the workload list for each core.**
- **How much disk space required (approximately)?: about 3MB**
- **How much time is needed to prepare workflow (approximately)?: at most 5 min (C++ compilation)**
- **How much time is needed to complete experiments (approximately)?: 10 hours (sequential running time, can be faster if running in parallel)**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: Several, see download**
- **Archived (provide DOI)?: Yes, 10.5281/zenodo.7751328**

A.3 Description

A.3.1 How to access. The artifact is uploaded to Zenodo (https://doi.org/10.5281/zenodo.7751328)

A.3.2 Software dependencies. A C++ compilation environment with support for the C++ 14 standard is required. It is recommended to use "GNU make" to build the program.

A.4 Installation

For artifact evaluation, start by downloading the artifact from Zenodo:

```
  $ wget -O SET_artifact.zip https://zenodo.org/
record/7751328/files/SET_artifact.zip
  $ unzip SET_artifact.zip
  $ cd SET_artifact
  $ ls -p -1

  SET_Fig8_data.xlsx
  SET_framework/
  Tangram_baseline/
```

Our SET scheduling framework is in "SET_framework".

For the convenience of evaluation, we have also provided our baseline, the open-sourced Tangram framework, in the folder "Tangram_baseline". The installation, usage, and modifications for the Tangram framework can be found in "Tangram_baseline/README_SET-Baseline.md".

The original data of Fig. 9 is included in the Excel file "SET_Fig8_data.xlsx" and can be verified by rerunning the two frameworks.

To install the SET framework, just use the makefile provided with the framework.

```
  $ cd SET_framework
  $ make
```

Then the executable will be generated at "./build/stschedule"

A.5 Experiment workflow

Now that our environment is set up, we will run the SET artifact and evaluate the result.

The first step is to write a config file. The config files for the experiments are provided under the folder "config". For config file customization, see the Experiment customization section.

Then, we can run the SET framework using the following command:

```
  $ ./build/stschedule conf.txt > res.txt
```

Where "conf.txt" is the name of the config file, and "res.txt" is the name of the output file.

To evaluate the experiment more conveniently, we provide a shell script, "run.sh", which will be introduced in the next part.

A.6 Evaluation and expected results

To evaluate the results, simply run

```
  $ ./run.sh
```

The script runs all experiments sequentially. Changing the script to execute in parallel can significantly reduce the running time, but it requires more available threads in the running environment. (The program executes multiple SA algorithms in different threads in parallel and chooses the best result among them. By default, 4 threads are needed for each execution).

The results are generated in the folder "results". After all executions finish, run the Python script "get_res.py" to generate a ".csv" file that contains all results:

```
  $ python3 ./get_res.py
  $ ls *.csv

  results.csv
```

We have provided the expected result files in folder "example_results". To verify the correctness of our framework, one can compare these files with the results output by the program under folder "results", or compare the data in the csv file with the data in the Excel file.

A.7 Experiment customization

To run customized experiments, users can write their own config file and feed it to the program. In a config file, the full list of parameters include:

```
  seed: the random seed used in SA

  net: the DNN workload
  batch: the number of samples in one batch

  mapper: the core type of the tile
  xLen, yLen: the size of the tile mesh
  stride: the stride used in placement

  costFunc: controls the cost function
  roundFactor: controls the number of rounds in the
       SA algorithm
  nTries: the number of SA that runs in parallel

  runLS, runLP: whether run LS/LP experiments or not
        (in addition to SET)

  IR_name: the name of the generated IR
```

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html