Klotski: DNN Model Orchestration Framework for Dataflow Architecture Accelerators

Chen Bai[1,3], Xuechao Wei[3], Youwei Zhuo[3], Yi Cai[3], Hongzhong Zheng[3], Bei Yu[1], Yuan Xie[2,3]
[1]The Chinese University of Hong Kong [2]Hong Kong University of Science and Technology
[3]DAMO Academy, Alibaba Group

***Abstract*—Dataflow architecture accelerators are a new kind of scalable DNN accelerators. The availability of input operands of the instructions solely determines the execution of instructions. This paper proposes the Klotski framework to solve DNN model orchestration for dataflow architecture accelerators. First, a Bayesian optimization-based entropy-directed partition algorithm is proposed to transform a DNN model into μops. Second, a unified formal formulation for μops scheduling and mapping is presented. Third, a two-stage methodology is proposed to decouple the scheduling and mapping, making the solution feasible. Extensive results show that Klotski outperforms baselines in runtime by an average of 9.55% and 48.48%.**

I. Introduction

The DNN models continue to evolve, hit breakthroughs, and gain astonishing outcomes with emergent abilities [1]. The success is achieved via computational scaling and algorithm innovation, which incurs deeply-stacked layers and complicated model structures and layer connections [2]–[4].

The ultimate pursuit of extremely efficient DNN model inference propelled the appearance of various DNN accelerators [5]–[8]. And the DNN accelerators are also scaled to keep pace with the increasing DNN models' size and structural complexity. The accelerator scales following two aspects. First, a DNN accelerator becomes more "brawny" [6], [9]. The hardware resources, like on-chip memory, computation units, *etc.*, are assigned more. Second, when "brawny" scaling cannot offer more profits from performance improvements [10], independent DNN accelerators are connected via dedicatedly-designed network-on-chip (NoC) [11]–[16]. As a result, they formulate scalable DNN accelerators (we term scalable scaling). The scalable scaling is effective. The philosophy behind this is that collaborative multi-processing opens opportunities for exploiting higher execution parallelism.

Dataflow architecture accelerators are a new kind of scalable DNN accelerators [17]–[19]. A fundamental distinction from previous scalable DNN accelerators is the execution model. That is, in dataflow architecture, the executability and execution of instructions is solely determined based on the availability of input operands to the instructions [20], [21]. In other words, dataflow architecture facilitate the asynchronous mechanism where multiple instructions operate on multiple data streams simultaneously. Fine-grained parallelism is allowed, and most synchronization steps are removed compared to traditional scalable DNN accelerators [13].

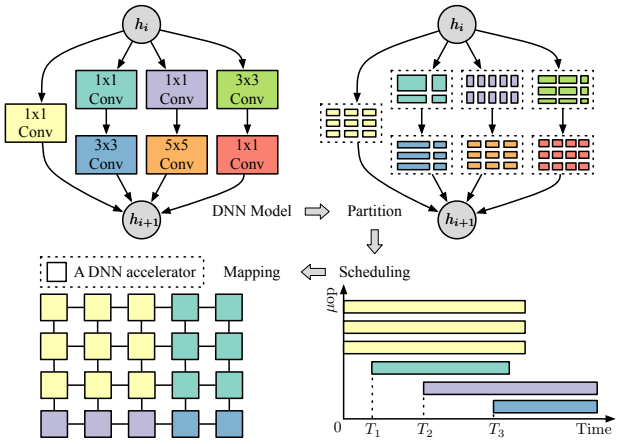

Fig. 1 A pipeline overview of DNN model orchestration for scalable DNN accelerators connected using the mesh topology. The example uses the GoogLeNet inception module. h_i and h_{i+1} are for hidden inputs and outputs. Partition generates different shapes of μ*tensors* for μ*ops*. Scheduling allocates μops' *time slots*. Mapping assigns accelerators for μops. Multiple μops can be executed simultaneously.

The orchestration of DNN models determine how to partition, schedule and map the model to scalable DNN accelerators, as shown in Fig. 1. Previous works propose some solutions to the problem [13], [22], [23]. CNN-Partition [22] proposed an automated hardware resource partition method to maximize the efficiency of accelerators. Tangram [13] applied alternate layer loop ordering (ALLO) dataflow to improve the inter-layer parallelism and leveraged the zig-zag mapping strategy to the underlying accelerators. Atomic dataflow [23] partitioned DNN models with simulated annealing and scheduled the DNN computation graph in finer granularity based on dynamic programming with pre-determined heuristics.

Nevertheless, these methods are proposed for traditional scalable DNN accelerators. In this paper, we propose Klotski, a DNN model orchestration framework for dataflow architecture accelerators. We give a formal mathematical formulation in Klotski. Firstly, we propose a Bayesian optimization-based entropy-directed DNN partition algorithm. In dataflow architecture accelerators, we flatten a DNN model to many

This work is done during Chen Bai's internship at Alibaba DAMO Academy, and is partially supported by AI Chip Center for Emerging Smart Systems (ACCESS) and Research Grants Council of Hong Kong SAR (No. CUHK14210723).

μops (shortened from "micro-operations") and break the layer connections to embrace higher parallelism. In consequence, the concept of inter-layer and intra-layer (or inter-operator and intra-operator in other literature) parallelism is giving way to the inter-μops and intra-μops parallelism. Secondly, we give a unified formal formulation for the scheduling and mapping. Thirdly, based on the unified formulation, we propose a two-stage methodology to decouple it, making the solution feasible. The scheduling allocates time slots for each μop to attain the promising *makespan* with an integer linear programming (ILP) model. And the mapping decides the allocation of an accelerator for a μop based on a mixed-integer programming (MIP) model. Our mapping algorithm can minimize the NoC transfer overheads during dataflow executions. It is worth noting that our partition algorithm is tightly coupled with the two-stage methodology. The partition can generate μops that maximize hardware utilization and achieve *load balancing* for dataflow architecture accelerators.

Our contributions are summarized as follows:

- A Bayesian optimization-based entropy-directed partition algorithm is proposed for μops generation.
- A unified formal formulation for the scheduling and mapping is proposed for the newly-emerged dataflow architecture accelerators.
- A two-stage methodology decoupling the unified formulation is proposed to make the solution feasible. With the methodology, we can minimize the makespan and NoC communication overheads.
- Extensive results show that Klotski can achieve 9.55% and 48.48% higher execution performance improvement.

The remainder of this paper is organized as follows. Section II introduces the preliminaries and the problem formulation. Section III provides the Klotski framework. Section IV is for experiments. Finally, Section V concludes this paper.

II. Preliminaries

A. Dataflow Architecture Accelerators

The traditional scalable DNN accelerators [10], [13] tile individual accelerators [5], [7]–[9] in a 2D manner. NoC connects all individual accelerators. And memory controllers, shared SRAMs, *etc.*, surround the 2D accelerator arrays. A microprocessor is leveraged to trigger the execution of scalable DNN accelerators. It sends instructions, controls data movement, and synchronizes all DNN accelerators.

Similarly, individual DNN accelerators are also connected via NoC in dataflow architecture accelerators. On the contrary, they are decentralized. An individual accelerator has the attribute of "egoism" in dataflow execution. The accelerators work asynchronously, and no central governor is incorporated. An instruction execution is solely triggered by the status of its required operands, *i.e.*, whether the operands are ready. The overall coordination of all accelerators are implemented by a proprietary NoC [19], [24]. The primal difference between the dataflow architecture accelerators and the traditional scalable DNN accelerators leads to the existence of accelerator synchronization. The traditional scalable

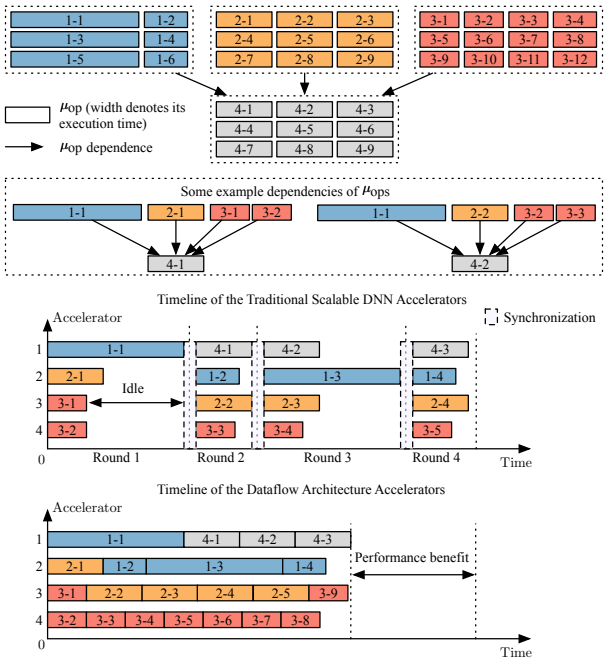

Fig. 2 Comparison between traditional scalable DNN accelerators and dataflow architecture accelerators.

DNN accelerators schedule a tensor computation per *round* after a synchronization. Instead, the dataflow architecture accelerators can schedule a tensor computation at any time slot, and no such synchronization is required.

Fig. 2 manifests the difference in detail. The illustration shown in Fig. 2 is from the three convolution layers in Fig. 1 with blue, orange, and red colors. The input/output tensors of the three convolution layers are partitioned into μtensors with different shapes, and example computation dependencies of μops are also visualized. For example, due to irregular partitions, the μop 4-1 relies on the outputs from μops 1-1, 2-1, 3-1, and 3-2. We use the scalable DNN accelerators, including four individual accelerators, as an example to illustrate the execution, and corresponding execution timelines are also shown. In traditional scalable DNN accelerators, μops are scheduled per round. Synchronization latencies are produced between adjacent rounds. The time interval of a round is decided by the μop, whose computing latency is the highest. Oppositely, dataflow architecture accelerators eliminate all synchronizations. So, the μop 4-1 can be fired immediately without delay. In this way, dataflow architecture accelerators permit exceptionally higher performance efficiency. The performance benefits come from two aspects. First, the synchronization overheads are eliminated, and a μop can be scheduled for any time slot. The performance improvement is demonstrated in Fig. 2 when two designs accomplish the computation of μop 4-3. Second, the μop processing throughputs are highly increased. Such as, the accelerator 4 in dataflow architecture can aggressively be filled with more μops from the red convolution layer.

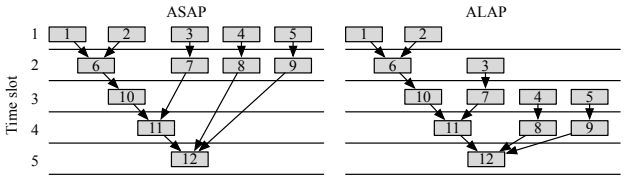

Fig. 3 An overview of ASAP and ALAP scheduling.

B. ASAP & ALAP Scheduling

Scheduling is a significant part of DNN model orchestration for dataflow architecture accelerators. Two basic scheduling techniques are as soon as possible (ASAP) and as late as possible (ALAP) algorithms. Considering a DNN model which is partitioned into several μops, ASAP and ALAP techniques can give the scheduling flexibility of each μop. Scheduling flexibility refers to the duration between the earliest possible and the latest possible time slot to issue a μop to an individual accelerator for execution. Fig. 3 gives the overview of ASAP and ALAP scheduling results of 12 μops (suppose the computation latency of each μop equals one). The μop 3 has scheduling flexibility between the time slot 1 and 2, and the μop 8 is more flexible, *i.e.*, from time slots 2 to 4.

Formally, the scheduling flexibility of a μop u_i given by ASAP and ALAP is formulated as follows:

$$\text{ASAP}(u_i) = \begin{cases} \max \text{ASAP}(u_j) + l(u_j), & \exists\ u_j \prec u_i,\ \forall u_j \\ 0, & \nexists\ u_j \prec u_i,\ \forall u_j \end{cases} \tag{1}$$

$$\text{ALAP}(u_i) = \begin{cases} \min \text{ALAP}(u_j) - l(u_j), & \exists\ u_i \prec u_j,\ \forall u_j \\ \max\limits_{u_k \in V} \text{ALAP}(u_k), & \nexists\ u_i \nprec u_j,\ \forall u_j \end{cases} \tag{2}$$

In Equation (1) and Equation (2), $l(u_j)$ is the execution latency of μop u_j. $u_j \prec u_i$ denotes producer-consumer relations. u_j is the producer, and u_i is the consumer. V is the set of μops.

C. Problem Formulation

The computation of a DNN model is usually represented as a directed acyclic graph (DAG) $G(V, E)$, where V denotes the set of all layers, and E represents the producer and consumer relations between these layers. We first introduce definitions used in dataflow architecture accelerators. Then we formally give three problem formulations for DNN model orchestration.

Definition 1 (μOp)**.** *μOp is defined as the execution granularity of an individual accelerator in dataflow architecture accelerators.*

Definition 1 is the formal definition of μop. A μop's operands are termed μtensors. We use u to denote a μop. The execution granularity is also the minimal scheduling unit. For example, the execution granularity can be per layer, per-atomic dataflow [23], *etc.*, for a DNN model.

Property 1 (μOp precedence constraints)**.** *The consumer μop should not begin to execute before the producer μops are completed.*

Following Section II-B, we use $u_i \prec u_j$ to denote that u_i is an immediate producer of u_j. The earliest possible time slot for u_j to start execution is when all its dependent μops are finished execution according to Property 1.

Definition 2 (Accelerator)**.** *An accelerator executes one μop at a time until its completion. Other μops cannot preempt the execution.*

We denote the dataflow architecture accelerators as $M(\boldsymbol{a}, \boldsymbol{b})$, where $\boldsymbol{a} = \{a_1, a_2, ..., a_m\}$ refers to accelerators defined in Definition 2. Each element b_i of $\boldsymbol{b}$ is the on-chip memory capacity for the accelerator a_i (*i.e.*, $\|\boldsymbol{a}\| = \|\boldsymbol{b}\|$). NoC connects all accelerators with a specific topology and routing algorithm.

The orchestration of a DNN model for dataflow architecture accelerators is formulated into three problems.

Problem 1 (Partition)**.** *The partition problem is to partition the computation of a DNN model into μops, aiming to maximize utilization of each accelerator, and achieve load balancing, given a set of constraints.*

Partitioning the computation of the DNN model into μops can introduce different parallelism granularity, as indicated by Definition 1. If we partition a DNN model by layer, then inter-layer and intra-layer parallelism is considered [13]. Finer-grained parallelism is achieved by partitioning the computation with smaller μops.

Problem 2 (Scheduling)**.** *The scheduling problem is to solve the allocation of time slots for μops (the solution from Problem 1), aiming to minimize the makespan.*

The scheduling problem decides the time slot allocations for each μop *w.r.t.* Property 1. Mapping is also conducted for μops. The mapping decides which individual accelerator is assigned to a given μop.

Problem 3 (Mapping)**.** *The mapping problem is to solve the allocation of accelerators for μops (the solution from Problem 1), aiming to minimize the NoC communication costs during dataflow executions.*

III. Klotski

A. Overview of Klotski

Fig. 4 gives an overview of Klotski. Given a DNN model and the dataflow architecture accelerators specifications, Klotski generates the model's orchestration solutions. The specifications refer to hardware configurations, like the type of an individual accelerator, memory capacity, processing elements (PE) arrays, NoC topology *etc.* First, we propose an entropy-guided partition algorithm based on Bayesian optimization (BO) (Section III-B). Various shapes of μtensors are generated. Then, we decouple the unified formal formulation (Section III-C) for μops scheduling and mapping via the

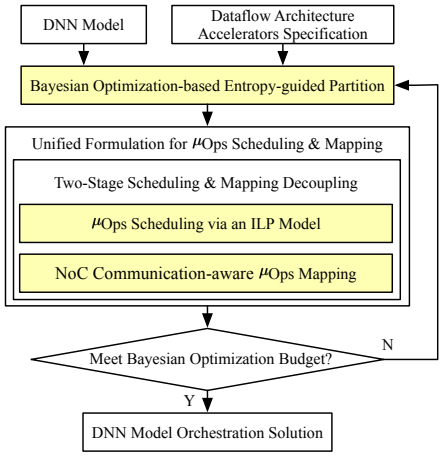

Fig. 4 An overview of Klotski framework.

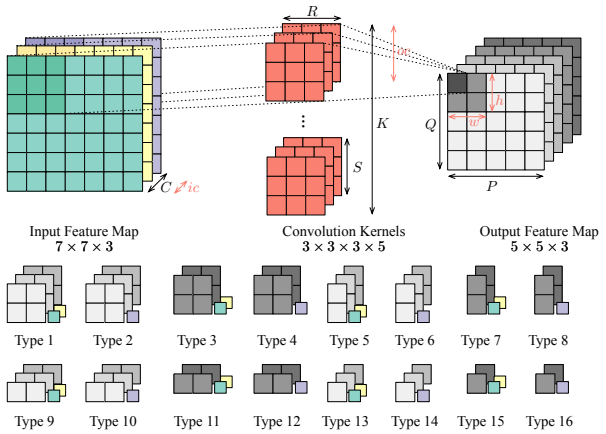

Fig. 5 An example shows a partition with $\boldsymbol{s}(2,2,2,3)$. We partition the output feature map computation by $\boldsymbol{s}$, generating at most 16 types of μtensors. Although the shape of type 1 and type 2 are the same (*i.e.*, $2 \times 2 \times 3$), they come from different input channels, *i.e.*, type 1 comes from the green and yellow channels, while type 2 is only from the purple channel. The input channels for each μtensor are also indicated.

two-stage methodology (Section III-D). μOps scheduling is solved via an ILP model. And the mapping is conducted via a NoC communication-aware MIP model. The makespan or a mean value of all μops' latencies are returned to the partition algorithm, expecting to generate promising μops in the next round. Finally, the DNN model orchestration solution is produced once the BO budget is met.

B. Bayesian Optimization-based Entropy-guided Partition

The partition algorithm produces μops for each layer of a DNN model. Two requirements should be satisfied in the process. *First, the computation of each μop should fully utilize an accelerator's resources.* Each accelerator involves PE arrays and applies specific unrolling and reusing strategies. Such as, ShiDianNao [5] adopts YX-partition, Eyeriss [7] leverages YR-partition, and NVDLA [9] implements KC-partition. The KC-partition favors μtensors with large input and output channels since NVDLA unrolls and parallelizes the two dimensions. Therefore, the shape of μtensors should adapt to various accelerator architectures, *i.e.*, maximize the utilization of each PE. *Second, the computation latency of all μtensors should be as close as possible to achieve load balancing.* Large gaps in computation latencies can delay consumer μops' executions since a producer μop can cost a high runtime to finish. As a result, the thundering herd problem [25] emerges when a DNN model consists of complicated structures. Namely, many consumer μops cannot utilize idle accelerators due to waiting for a "bad" producer μop with a long latency. Resource contention occurs immediately once the producer finishes execution.

Our partition utilizes a representation similar to the atomic dataflow [23]. We demonstrate the idea using a single convolution layer, as shown in Fig. 5. The convolution layer is with the shape represented by a tuple (R, S, P, Q, C, K), where the elements are for kernel width and height, output width and height, input channel, and output channel, respectively. We partition output tensors with $\boldsymbol{s}(h, w, ic, oc)$, where h and w refer to the μtensors' height and width, ic is for the input channels, and oc refers to the μtensors' output channels. If a dimension cannot be divisible by an element of $\boldsymbol{s}$, we treat the remainder as extra μtensors. Through $\boldsymbol{s}$, a convolution layer can be divided into up to 16 different kinds of μtensors, detailed in Fig. 5. We handle computation dependencies of μops from different layers via automated analysis.

As for the first requirement, we partition by remainder-free operations [23]. For example, an accelerator implements KC-partition, and its PEs are tiled into an $m \times n$ 2D array. Then, ic and oc should be multiple of m and n, which makes μtensors completely unrolled to utilize PEs fully. Regarding the second requirement, we formulate it as a design space exploration since the relation between a partition strategy and the corresponding execution makespan do not have a clear form. Hence, we leverage BO to search for promising $\boldsymbol{s}$. The search is guided by maximizing the proposed entropy function normalized by the execution makespan, as shown in Equation (3),

$$E(\boldsymbol{s}) = -\Big(\sum_{u_i \in V} \frac{l(u_i)}{l(V)} \ln \frac{l(u_i)}{l(V)}\Big) / (\alpha \cdot \text{makespan}), \quad (3)$$

where we reuse V and l defined in Section II-B, and $l(V) = \sum_{u_i \in V} l(u_i)$. The makespan refers to the execution runtime after we schedule and map μops to dataflow architecture accelerators. α is a pre-determined coefficient to normalize the entropy score.

The rationale behind Equation (3) is based on the *principle of maximum entropy* [26]. Entropy maximization tries to attain a uniform distribution. Analogously, the second requirement exactly aligns with the characteristic. So, the entropy of μops' latencies reaching its maximum corresponds to the optimal load balancing. Furthermore, we divide the entropy score with the makespan, expecting to find a partition strategy to improve the execution runtime. We also find it is effective

Algorithm 1 BO-based Entropy-guided Partition

Require: G: a DNN model. $\mathbb{D}$: the design space for $\boldsymbol{s}$. T: optimization budget.

1: $S = \emptyset$; Sample $\boldsymbol{s} \in \mathbb{D}$;
2: **for** $i = 1 \rightarrow T$ **do**
3: Partition G with $\boldsymbol{s}$;
4: Schedule, map, execute μops;
5: Evaluate $E(\boldsymbol{s})$; ▷ Equation (3)
6: $S = S \cup \{(\boldsymbol{s}, E(\boldsymbol{s}))\}$;
7: Construct a Gaussian process model with S;
8: $\boldsymbol{s}^* = \text{argmax}_{\boldsymbol{s}\in\mathbb{D}}\,\text{UCB}(\boldsymbol{s})$; $\boldsymbol{s} = \boldsymbol{s}^*$
9: **end for**
10: **return** Optimal $\boldsymbol{s}^*$ from S.

by setting the makespan as the mean latency of all μops.

We adopt BO due to its promising results for general applications [27]. BO consists of a *surrogate model* and an *acquisition function*. The surrogate model constructs a mapping from a partition strategy $\boldsymbol{s}$ to the function value $E(\boldsymbol{s})$. The acquisition function is leveraged to evaluate the utility of $\boldsymbol{s}$ and decide whether a particular $\boldsymbol{s}$ could improve $E(\boldsymbol{s})$ more. Within each BO iteration, the surrogate model is established on an augmented exploration set. The model is improved to predict more accurately on relative rankings between different $\boldsymbol{s}$. Algorithm 1 lists the partition algorithm, where S in line 1 is the exploration set. In line 7, we leverage the automatic relevance determination (ARD) Matérn $5/2$ kernel, and the upper confidence bound (UCB) in line 8 is our acquisition function. We augment the exploration set in line 6. Additionally, we restrict a partition solution if a generated μtensor size is larger than the on-chip memory b_i.

C. Unified Formulation for μOps Scheduling & Mapping

We give a formal formulation for the scheduling and mapping with generated μops (Section III-B). Our formulation targets to minimize the makespan and the NoC communication cost. First, we illustrate our motivations to optimize the NoC communication cost. Then we demonstrate scheduling constraints construction. Next, we give the formulation for the NoC communication cost. Finally, we present the unified formulation for μops scheduling and mapping.

Unlike traditional scalable DNN accelerators, dataflow architecture accelerators suffer from high NoC communication overhead. The reason lies in two folds. First, the asynchronization mechanism transfers the collaboration control of individual accelerators from an external microprocessor to NoC. NoC is not only responsible for data communication but also for coordination. Second, a DNN model is partitioned into μops, leading to many communication demands. Therefore, the numbers of NoC communications, like data movement and buffering, are growing.

We apply the list scheduling [28], [29] to acquire the upper bound of the makespan. The main idea of list scheduling is to schedule a ready μop, whose dependent producers are finished, from a pre-defined order for idle accelerators without delay. Theorem 1 gives the upper bound formally.

Theorem 1 (Upper Bound of the Makespan for μOps Scheduling)**.** *List scheduling achieves* $2 - 1/\|\boldsymbol{a}\|$ *times the optimal makespan for dataflow architecture accelerators.*

With the upper bound (denote it as T), we can calculate the scheduling flexibility of each μop with ASAP and ALAP (Section II-B) accordingly. Suppose the earliest possible and the latest possible time slot for one μop u_i to issue are S_i and L_i. We define a binary tensor $\boldsymbol{\mathcal{X}}$ with the size of $|V| \times T \times \|\boldsymbol{a}\|$ as the scheduling and mapping solution, shown in Equation (4)

$$\boldsymbol{\mathcal{X}}_{ijk} = \begin{cases} 1, \text{ } \mu\text{op } u_i \text{ is scheduled to the } k\text{-th accelerator} \\ \quad \text{at the } j\text{-th time slot.} \\ 0, \text{ otherwise.} \end{cases} \tag{4}$$

A legal solution should satisfy following constraints.

μOp constraint: A μop can only be scheduled within the scheduling flexibility (Equation (5)). And it can only be issued to one individual accelerator.

$$\sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{j=S_i}^{L_i} \boldsymbol{\mathcal{X}}_{ijk} = 1, \ \forall u_i \in V. \tag{5}$$

Each μop must be finished before T (Equation (6)).

$$\sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{j=S_i}^{L_i} (j + l(u_i) - 1)\boldsymbol{\mathcal{X}}_{ijk} \leq T, \ \forall u_i \in V. \tag{6}$$

Constraint by Property 1: Property 1 determines the scheduling priority for μops to guarantee correct computations (Equation (7)).

$$\sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{j=S_p}^{L_p} j \cdot \boldsymbol{\mathcal{X}}_{pjk} - \sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{j=S_q}^{L_q} j \cdot \boldsymbol{\mathcal{X}}_{qjk} \leq -l(u_p), \quad \forall u_p, \ \forall u_q \in V \text{ and } u_p \prec u_q. \tag{7}$$

Computing resource constraint: In each time slot, the number of μops fired or held by accelerators should be fewer than the number of accelerators (Equation (8)).

$$\sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{p=0}^{l(u_i)-1} \sum_{i=1}^{|V|} \boldsymbol{\mathcal{X}}_{i(j-p)k} \leq \|\boldsymbol{a}\|, \quad \forall j = \{1, 2, ..., T\} \tag{8}$$

Our targets are the makespan T and the NoC communication cost. We elucidate the NoC communication cost formulation by walking through a concrete example.

Consider an NoC with mesh topology and XY-YX routing algorithm [30], shown in Fig. 6. The producer μop u_p is mapped to the accelerator highlighted in yellow. The consumer μop u_q resides on the accelerator shaded with green and requests a NoC communication. x and y are the numbers of individual accelerators per row and column. The route path is not unique and is determined by runtime information like packet congestion. Three possible route paths are visualized in Fig. 6. Regardless of the route path used, communication costs can be uniformly described. The communication cost is relative to the number of hops from the source to the

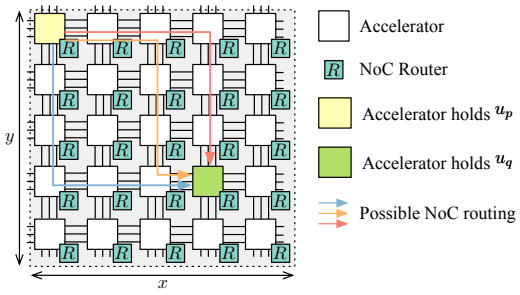

Fig. 6 An overview of an NoC communication. We can efficiently reduce NoC communication costs by placing dependent μops on the same or adjacent accelerators.

target accelerator, which is correlated with the locations of accelerators. Hence, the NoC communication cost from μop u_p to μop u_q is

$$c_{u_p \prec u_q} = |x_1 - x_2| + |y_1 - y_2|, \tag{9}$$

where u_p's assigned accelerator is at (x_1, y_1), and the acclerator for u_q is at (x_2, y_2). We have following equations to compute the locations of accelerators.

$$x_1 = \left\lfloor \frac{a}{x} \right\rfloor, \; y_1 = a \bmod x \tag{10}$$

$$x_2 = \left\lfloor \frac{b}{x} \right\rfloor, \; y_2 = b \bmod x \tag{11}$$

$$a - b \neq 0 \text{ if } \sum_{k=1}^{\|a\|} \boldsymbol{\mathcal{X}}_{pjk} + \sum_{k=1}^{\|a\|} \boldsymbol{\mathcal{X}}_{qjk} > 1, \quad j \in \{1, 2, ..., T\} \tag{12}$$

$$a = \sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{j=S_p}^{L_p} k\boldsymbol{\mathcal{X}}_{pjk}, \; b = \sum_{k=1}^{\|\boldsymbol{a}\|} \sum_{j=S_q}^{L_q} k\boldsymbol{\mathcal{X}}_{qjk}. \tag{13}$$

Equation (10) and Equation (11) are formulations of source and target accelerators' locations, respectively. Equation (12) prevents duplicated mappings if u_p and u_q are fired at the same time slot. Equation (13) decides which accelerator to map a μop. The entire NoC communication costs are the summation of the point-to-point μtensors transmissions (Equation (9)) between a producer and a consumer, as shown in Equation (14).

$$C = \sum_{e=1}^{|E|} |x_{e1} - x_{e2}| + |y_{e1} - y_{e2}|, \tag{14}$$

where E is the set of all producer-consumer relations among μops. x_{ei} and y_{ei} are (x_i, y_i) of the e-th relation.

The complete unified formulation for the scheduling and mapping is shown below.

$$\begin{aligned} &\underset{\boldsymbol{\mathcal{X}}}{\text{argmin}} \quad T + \beta C \\ &\text{s.t.} \quad \text{Equations (5)} - \text{(13)}, \end{aligned} \tag{15}$$

where β is a coefficient to trade-off T and C. And the length of a single time slot is determined by $\min \; l(u_i), \; \forall u_i \in V$. Nevertheless, two difficulties restrict us from solving Equation (15). First, when the problem size becomes large, it costs high runtime to construct constraints like Equation (8). Second, the problem cannot be solved with mathematical programming due to non-linearity in Equation (9), Equation (10), and Equation (11). We introduce a two-stage methodology to decouple the scheduling and mapping, making the solution feasible contrapuntally.

D. Two-Stage Scheduling & Mapping Decoupling

We decouple the scheduling and mapping by defining two new varibles. Define a $|V| \times T$ binary matrix $\boldsymbol{X}$ shown in Equation (16) as a scheduling solution.

$$\boldsymbol{X}_{ij} = \begin{cases} 1, \; \mu\text{op } u_i \text{ is scheduled to the } j\text{-th time slot.} \\ 0, \; \text{otherwise.} \end{cases} \tag{16}$$

And we define a binary matrix $\boldsymbol{Y}$ with the size of $|V| \times \|\boldsymbol{a}\|$ as the mapping solution, shown in Equation (17).

$$\boldsymbol{Y}_{ij} = \begin{cases} 1, \; \mu\text{op } u_i \text{ is mapped to the } j\text{-th accelerator;} \\ 0, \; \text{otherwise.} \end{cases} \tag{17}$$

As a result, Equation (15) is transformed into two sub-problems. The first sub-problem is listed in Equation (18).

$$\begin{aligned} &\underset{\boldsymbol{X}}{\text{argmin}} \quad T \\ &\text{s.t.} \sum_{j=S_i}^{L_i} \boldsymbol{X}_{ij} = 1, \quad \sum_{j=S_i}^{L_i} (j + l(u_i) - 1)\boldsymbol{X}_{ij} \leq T \\ &\sum_{j=S_i}^{L_i} j \cdot \boldsymbol{X}_{ij} - \sum_{j=S_k}^{L_k} j \cdot \boldsymbol{X}_{kj} \leq -l(u_i), \quad u_i \prec u_j \\ &\sum_{p=0}^{l(u_i)-1} \sum_{i=1}^{|V|} \boldsymbol{X}_{i(j-p)} \leq \|\boldsymbol{a}\|, \quad \forall j \in \{1, 2, ..., T\}, \end{aligned} \tag{18}$$

The computing resource constraint in Equation (18) (the last constraint) still requires a high runtime cost to construct. So, we relax it and allow the existence of contentions for accelerators in the formulation with Equation (19).

$$\sum_{i \in I = \{i | j \in [S_i, L_i]\}} \boldsymbol{X}_{ij} \leq \|\boldsymbol{a}\|, \quad \forall j \in \{1, 2, ..., T\} \tag{19}$$

The relaxation is valid for dataflow architecture accelerators since NoC can buffer the resource contentions. In this way, we can only require the number of issued μops to be fewer than the number of individual accelerators, given any time slot. Consequently, we accommodate Equation (12) for pair of μops that can be parallelized during executions. The parallelism comes from inter-μops and intra-μops.

Concerning the non-linearity in Equation (15), we introduce new variables to transform the mapping problem as mixed-integer linear programming. Take an example from Equation (10) to Equation (13). With newly-incorporated six rational variables $(k_1, k_2, n_1, n_2, r_1, r_2)$, four integer variables (x_1, x_2, p, q), and a binary variable z, we can transform the formulation to a solvable MIP model, as shown below[1].

[1]We omit the details of the cumbersome transformation process.

$$\underset{\boldsymbol{Y}_i,\boldsymbol{Y}_j}{\operatorname{argmin}} \quad k_1+k_2+n_1+n_2 \tag{20}$$

$$\text{s.t.} \quad x_1-x_2=k_1-k_2,\ p-q=n_1-n_2 \tag{21}$$

$$\frac{a}{x}-\epsilon\le x_1\le\frac{a}{x},\ \frac{b}{x}-\epsilon\le x_2\le\frac{b}{x} \tag{22}$$

$$a=p\cdot x+r_1,\ b=q\cdot x+r_2 \tag{23}$$

$$0\le r_1\le x-1,\ 0\le r_2\le x-1 \tag{24}$$

$$\delta-(1-z)\cdot M\le a-b\le-\delta+Mz \tag{25}$$

$$M=\|\boldsymbol{a}\|+1 \tag{26}$$

$$a=\sum_{k=1}^{\|\boldsymbol{a}\|}k\boldsymbol{Y}_{ik},\ b=\sum_{k=1}^{\|\boldsymbol{a}\|}k\boldsymbol{Y}_{jk}, \tag{27}$$

$$k_1,k_2,n_1,n_2\ge 0,\ p,q,x_1,x_2\in\mathbb{Z},\ z\in\{0,1\}. \tag{28}$$

In Equation (22) and Equation (25), ϵ and δ are two small constants. Specifically, we choose $\epsilon=0.999$ and $\delta=0.001$.

With the two-stage decoupling, we make the unified formulation solvable.

IV. Experiments

A. Implementation Details

We build an in-house simulator for the dataflow architecture accelerators. The simulator leverages a proprietary NoC with the mesh topology to connect individual DNN accelerators. The NoC transfers computation results between individual DNN accelerators and sends and receives requests and replies to achieving asynchronous executions. It applies the XY-YX routing algorithm [30] with a dedicated-designed congestion control mechanism. And its bandwidth is 512 bytes/cycle, with each packet size of 8 bytes. Each individual DNN accelerator adopts KC-partition and equips with a 12×14 PE array. PEs are shared with an on-chip SRAM memory of 5.86 MB. The size of registers in each PE is 5 KB. The precision of computations are 32 bits. The flit size is 192 bits and frequency is 1GHz. MAESTRO [31] is embedded to model the execution performance of one individual DNN accelerator for simplicity.

In Klotski, we adopt an open-source tool *nn_dataflow* [32] as the front end of DNN models. The DNN model partition is implemented based on the framework. In the partition algorithm, the maximal budget of the BO-based entropy-guided partition is set as 50. We use Gurobi v10.0 [33] as the ILP and MIP solver in the two-stage methodology.

B. Baselines & Wokrloads

Our baselines are based on Tangram [13] and the atomic dataflow [23]. Tangram [13] schedules a DNN model per layer and proposes a zig-zag mapping strategy. The atomic dataflow [23] is the latest methodology, combined with heuristics and dynamic programming. Both are representative solutions to the problem in traditional scalable DNN accelerators. However, since our problem originates from a new architecture, we cannot directly compare Klotski with them. We made appropriate modifications to Tangram and the atomic dataflow so that they can orchestrate a DNN workload with our architecture model. For Tangam, we schedule a DNN model layer-wise and allocate accelerators following the zig-zag manner, and we term it as "baseline 1". The atomic dataflow schedules a workload per round with heuristics due to the synchronization in traditional scalable DNN accelerators. We remove the synchronization and consider the heuristics proposed by the atomic dataflow for all ready μops rather than a candidate set in each round. We term the modified method as "baseline 2". The modifications adhere to both methods' original ideas but extend their applicability to dataflow architecture accelerators.

Our workloads are VGG16, VGG19, ResNet50, ResNet152, and Inception v3. The workloads include cascaded layers structures, branching cells, and residual layers with different scales. The cascaded layers can be parallelized if layers are partitioned into μops.

TABLE I The experimental results for the 3×3 topology

Workload	Method	Cycles	Ratio	Overall Runtime	Ratio	HUR [1]
VGG16	Baseline 1	1.2283E + 08	1.0000	−− [2]	−−	1.0000
	Baseline 2	5.5633E + 07	0.4529	477.6634	1.0000	2.5617
	Klotski	4.0659E + 07	0.3310	878.8832	1.8399	3.0602
VGG19	Baseline 1	1.5523E + 08	1.0000	−−	−−	1.0000
	Baseline 2	7.4207E + 07	0.4781	576.3081	1.0000	2.5229
	Klotski	5.5381E + 07	0.3568	887.5790	1.5401	2.9857
ResNet50	Baseline 1	7.7422E + 07	1.0000	−−	−−	1.0000
	Baseline 2	5.7060E + 07	0.7370	583.6488	1.0000	0.9762
	Klotski	4.8174E + 07	0.8443	1779.0426	3.0481	1.3050
ResNet152	Baseline 1	1.8984E + 08	1.0000	−−	−−	1.0000
	Baseline 2	1.7102E + 08	0.9009	867.0853	1.0000	1.2523
	Klotski	1.5947E + 08	0.8400	2800.9154	3.2302	1.3605
Inception	Baseline 1	2.5122E + 07	1.0000	−−	−−	1.0000
	Baseline 2	1.6345E + 07	0.6506	470.3763	1.0000	2.5103
	Klotski	1.3348E + 07	0.5313	1397.9008	2.9719	3.2996

[1] Hardware utilization ratio
[2] Not applicable

TABLE II The experimental results for the 4×4 topology

Workload	Method	Cycles	Ratio	Overall Runtime	Ratio	HUR
VGG16	Baseline 1	1.2283E + 08	1.0000	−−	−−	1.0000
	Baseline 2	4.5869E + 07	0.3734	317.5903	1.0000	2.1196
	Klotski	3.0670E + 07	0.2497	881.6310	2.7760	2.4547
VGG19	Baseline 1	1.5523E + 08	1.0000	−−	−−	1.0000
	Baseline 2	5.8049E + 07	0.3740	388.8627	1.0000	1.9895
	Klotski	3.9934E + 07	0.2573	1130.6444	2.9076	2.2964
ResNet50	Baseline 1	7.7422E + 07	1.0000	−−	−−	1.0000
	Baseline 2	5.3365E + 07	0.6893	541.8091	1.0000	2.8954
	Klotski	4.6260E + 07	0.5975	1019.2198	1.8811	3.1953
ResNet152	Baseline 1	1.8984E + 08	1.0000	−−	−−	1.0000
	Baseline 2	1.6578E + 08	0.8733	793.7304	1.0000	1.2264
	Klotski	1.5754E + 08	0.8299	2327.4657	2.9323	1.3438
Inception	Baseline 1	2.5188E + 07	1.0000	−−	−−	1.0000
	Baseline 2	1.5183E + 07	0.6028	419.3479	1.0000	2.2822
	Klotski	1.0781E + 07	0.4280	1432.0112	3.4148	2.8579

C. Comparison to Previous Methodologies

In the evaluation, we leverage three kinds of scales of dataflow architecture accelerators, in which the individual accelerators are organized in 3×3, 4×4, and 5×5 arrays. Such hardware scaling is used to denote different problem sizes, allowing us to fully evaluate Klotski and baselines. TABLE I, TABLE II, and TABLE III list related results, including running cycles, the overall runtime (include Gurobi

TABLE III The experimental results for the 5×5 topology

Workload	Method	Cycles	Ratio	Overall Runtime	Ratio	HUR
VGG16	Baseline 1	1.2283E + 08	1.0000	−−	−−	1.0000
	Baseline 2	4.2621E + 07	0.3470	466.9748	1.0000	2.7157
	Klotski	2.4240E + 07	0.1973	1640.0338	3.5120	3.4766
VGG19	Baseline 1	1.5523E + 08	1.0000	−−	−−	1.0000
	Baseline 2	5.0412E + 07	0.3248	569.8779	1.0000	2.8346
	Klotski	3.9046E + 07	0.2515	2755.4077	4.8351	3.1257
ResNet50	Baseline 1	7.7422E + 07	1.0000	−−	−−	1.0000
	Baseline 2	5.0868E + 07	0.6570	628.1705	1.0000	1.8228
	Klotski	4.4029E + 07	0.5687	1672.0000	2.6617	1.9678
ResNet152	Baseline 1	1.8984E + 08	1.0000	−−	−−	1.0000
	Baseline 2	1.6460E + 08	0.8671	858.0045	1.0000	1.2575
	Klotski	1.5240E + 08	0.8028	4505.7838	5.2515	1.3352
Inception	Baseline 1	2.5180E + 07	1.0000	−−	−−	1.0000
	Baseline 2	1.2733E + 07	0.5057	514.9384	1.0000	2.8642
	Klotski	8.3088E + 06	0.3300	2787.1383	5.4126	3.3710

solution runtime for Klotski) for different algorithms, and the hardware utilization ratio.

Baseline 1 does not partition a DNN model and straightly schedule and map the model layer-wise. The entire procedure is deterministic, and no search or tuning is incorporated. Hence, the runtime results for baseline 1 are not applicable. Baseline 2 is also deterministic except for the employed simulated annealing for partition. Thus, the runtime of baseline 2 largely depends on partition[2].

In the 3×3 topology, compared to baseline 1 and baseline 2, the solution given by Klotski outperforms by an average of 44.42% and 10.03% for all DNN workloads. In the 4×4 topology, the numbers are 49.01% and 9.29%. And in the 5×5 topology, they are 52.02% and 9.33%.

Klotski also utilizes the underlying hardware better. The hardware utilization is the average utilization of all individual DNN accelerators. In the 3×3 topology, Klotski achieves an average of 140.22% and a 45.35% higher than baselines. Compared to baseline 1, the improvement numbers are 142.96% and 165.52% for the 4×4 and 5×5 topologies, respectively. Klotski also improves the hardware utilization ratio compared to baseline 2. In the 4×4 topology, Klotski surpasses baseline 2 by 90.52% and 35.63%.

However, Klotski costs more runtime. It is due to that Klotski leverages much time to solve the scheduling and mapping in the two-stage methodology. When the topology size enlarges, the runtime cost continues to increase. We expect approximate algorithms to improve Klotski's efficiency in future work.

D. Ablation Study

We investigate the effectiveness of scheduling and mapping by Klotski with an ablation study. The partition strategy explored by Klotski is leveraged for baseline 2. Baseline 1's results are also compared, listed in Fig. 7. In the 3×3 topology, Klotski outperforms baseline 1 and baseline 2 by 46.34% and 4.10%. For the 4×4 and 5×5 topology, the improvments for baseline 1 and baseline 2 are 50.44%, 3.13%, 46.34%, and 3.71%, respectively. The results demonstrate that Klotski effectively improves the scheduling and mapping solution quality. Moreover, with the BO-based entropy-guided partition, Klotski can outperform baselines more.

[2]In the experiments, we set the running budgets for the simulated annealing with 50 to align with Klotski's settings.

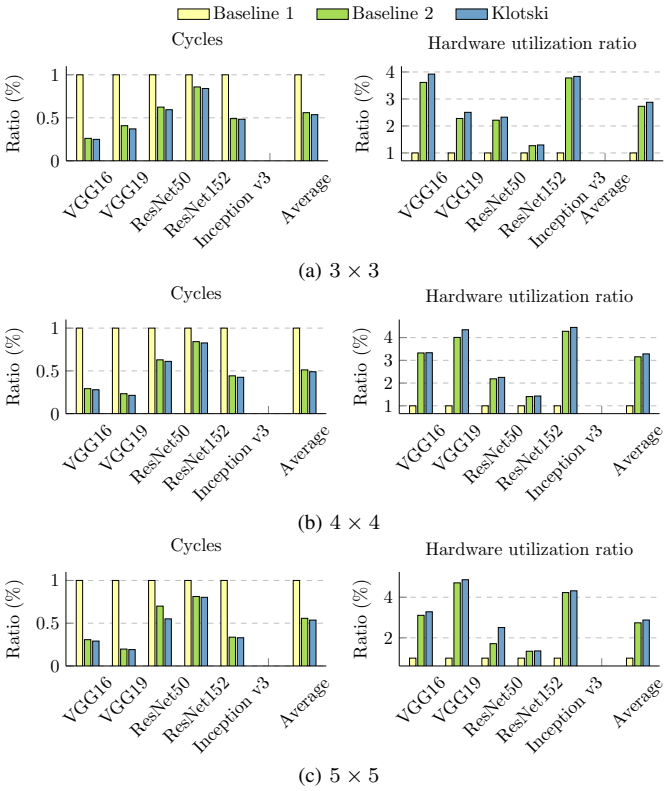

Fig. 7 The comparisons with different topologies.

We summarize two lessons learned from Section IV-C and Fig. 7. *Firstly, partitioning a DNN model into μops allows better execution performance, even for cascaded layers structures.* The claim can be concluded from comparisons between Klotski and baseline 1 with different DNN models. Particularly, baseline 1 fails to explore opportunities to parallelize the cascaded layers executions (VGG16 and VGG19). *Second, the improvement of the execution performance is non-linear to the increased hardware utilization ratio.* Compared to baseline 2, Klotski achieves higher the hardware utilization, as listed in TABLE I, TABLE II, TABLE III and Fig. 7. However, the gained hardware utilization ratio does not contribute to execution performance improvement expectedly. The reason stems from multiple factors. For example, different μops lead to various computation and memory access trade-offs. Smaller sizes in μtensors do not represent higher performance efficiency.

V. Conclusions

In this paper, we propose Klotski, a DNN model orchestration framework for the newly-emerged dataflow architecture accelerators. With BO-based entropy-guided partition, and two-stage decoupling of the scheduling and mapping, Klotski improves the solution qualities by an average of 9.55% and 48.48% compared to baselines.

REFERENCES

[1] OpenAI, "GPT-4 Technical Report," 2023.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models are Few-shot Learners," *Annual Conference on Neural Information Processing Systems (NIPS)*, vol. 33, pp. 1877–1901, 2020.

[3] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, "Lamda: Language Models for Dialog Applications," *arXiv preprint arXiv:2201.08239*, 2022.

[4] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical Text-conditional Image Generation with Clip Latents," *arXiv preprint arXiv:2204.06125*, 2022.

[5] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter Performance Analysis of a Tensor Processing Unit," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[7] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 367–379.

[8] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling Systematic Deep-learning Architecture Evaluation via Full-stack Integration," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[9] "NVIDIA NVDLA deep learning accelerator," 2017, http://nvdla.org.

[10] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017, pp. 751–764.

[11] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2016, pp. 380–392.

[12] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey *et al.*, "SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2017, pp. 13–26.

[13] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 807–820.

[14] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous Dataflow Accelerators for Multi-DNN Workloads," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[15] D. Abts, G. Kimmell, A. Ling, J. Kim, M. Boyd, A. Bitar, S. Parmar, I. Ahmed, R. DiCecco, D. Han *et al.*, "A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2022, pp. 567–580.

[16] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2023.

[17] J. Vasiljevic, L. Bajic, D. Capalija, S. Sokorac, D. Ignjatovic, L. Bajic, M. Trajkovic, I. Hamer, I. Matosevic, A. Cejkov *et al.*, "Compute Substrate for Software 2.0," *IEEE Micro*, vol. 41, no. 2, pp. 50–55, 2021.

[18] K. Sankaralingam, T. Nowatzki, V. Gangadhar, P. Shah, M. Davies, W. Galliher, Z. Guo, J. Khare, D. Vijay, P. Palamuttam *et al.*, "The Mozart Reuse Exposed Dataflow Processor for AI and Beyond," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2022, pp. 978–992.

[19] D. Ignjatović, D. W. Bailey, and L. Bajić, "The Wormhole AI Training Processor," in *IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 356–358.

[20] J. B. Dennis, "Data Flow Supercomputers," *Computer*, vol. 13, no. 11, pp. 48–56, 1980.

[21] T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Heterogeneous Von Neumann/Dataflow Microprocessors," *Communications of the ACM*, vol. 62, no. 6, pp. 83–91, 2019.

[22] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2017, pp. 535–547.

[23] S. Zheng, X. Zhang, L. Liu, S. Wei, and S. Yin, "Atomic Dataflow based Graph-Level Workload Orchestration for Scalable DNN Accelerators," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 475–489.

[24] "D. Ignjatović and D. Capalija, "Scale-out First Microarchitecture for Efficient AI Training", Linley Spring Processor Conference, 2021." 2021, https://www.youtube.com/watch?v=Id3enIOAY2Q.

[25] C. Min and Y. I. Eom, "Dynamic scheduling of irregular stream programs toward many-core scalability," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 26, no. 6, pp. 1594–1607, 2014.

[26] E. T. Jaynes, "Information Theory and Statistical Mechanics," *Physical review*, vol. 106, no. 4, p. 620, 1957.

[27] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[28] R. L. Graham, "Bounds for Certain Multiprocessing Anomalies," *Bell system technical journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[29] ——, "Bounds on Multiprocessing Timing Anomalies," *SIAM journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[30] N. E. Jerger, T. Krishna, and L.-S. Peh, "On-chip Networks," *Synthesis Lectures on Computer Architecture*, vol. 12, no. 3, pp. 1–210, 2017.

[31] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: A Data-Centric Approach," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019, pp. 754–768.

[32] "nn_dataflow: a Neural Network Dataflow Scheduling Tool," https://github.com/stanford-mast/nn_dataflow.

[33] L. Gurobi Optimization, "Gurobi Optimizer Reference Manual (2020)," 2020.