IEEE/ACM
2023 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN
ICCAD
42nd Edition

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]

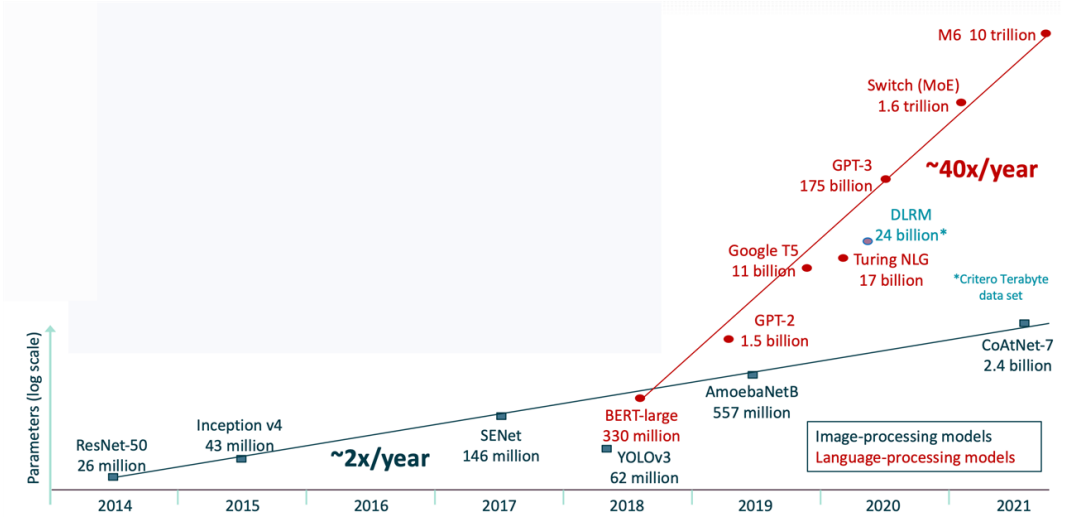
[1] The Chinese University of Hong Kong
[2] Hong Kong University of Science and Technology
[3] DAMO Academy, Alibaba Group

Nov. 01, 2023

Outline

Introduction

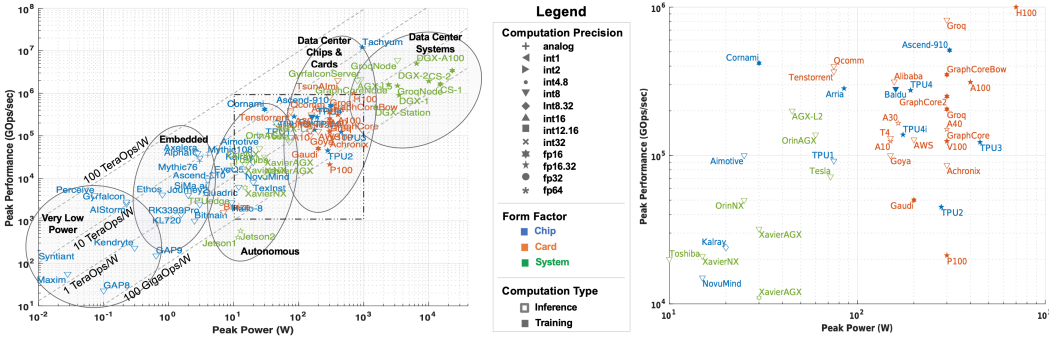
Introduction: AI Models Scaling Trends



An overview of AI models scaling trends [1].

[1] by courtesy: https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/linleyspr22

Introduction: AI Accelerators Scaling Trends



Peak performance vs. power scatter plot of publicly announced AI accelerators and processors (by July 2022)[2].

[2] Albert Reuther et al. (2022). "AI and ML Accelerator Survey and Trends". In: *IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, pp. 1–10.

Introduction: AI Accelerators Scaling Trends

- "Brawny" scaling.
- Scalable scaling.

Introduction: AI Accelerators Scaling Trends

- "Brawny" scaling.
  - Scale on-chip hardware resources[3][4][5][6][7].

[3] Zidong Du et al. (2015). "ShiDianNao: Shifting Vision Processing Closer to the Sensor". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 92–104.

[4] Yu-Hsin Chen, Joel Emer, and Vivienne Sze (2016). "Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 367–379.

[5] *The NVIDIA Deep Learning Accelerator (NVDLA)* (2017). http://nvdla.org/.

[6] Norman P Jouppi et al. (2017). "In-datacenter Performance Analysis of a Tensor Processing Unit". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 1–12.

[7] Hasan Genc et al. (2021). "Gemmini: Enabling Systematic Deep-learning Architecture Evaluation Via Full-stack Integration". In: *ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 769–774.

Introduction: AI Accelerators Scaling Trends

- Scalable scaling.
  - Scale DNN accelerators via an network-on-chip (NoC)[8][9][10][11].

[8] Swagath Venkataramani et al. (2017). "SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 13–26.

[9] Mingyu Gao et al. (2019). "Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 807–820.

[10] Dennis Abts, Jonathan Ross, et al. (2020). "Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 145–158; Dennis Abts, Garrin Kimmell, et al. (2022). "A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 567–580.

[11] Norm Jouppi et al. (2023). "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 1–14.

Introduction: Dataflow Architecture Accelerators

Main features of Dataflow Architecture Accelerators:

- Dataflow architecture accelerators are a new kind of scalable scaling-driven AI accelerators.
- A fundamental distinction from previous scalable DNN accelerators is the execution model.
- Permits asynchronous mechanism where multiple instructions operate on multiple data streams simultaneously (MIMD).

Dataflow Execution Model

The executability and execution of instructions is solely determined based on the availability of input operands to the instructions[12].

Compute only what is relevant to input proactively[13].

[12] Tony Nowatzki, Vinay Gangadhar, and Karthikeyan Sankaralingam (2019). "Heterogeneous Von Neumann/Dataflow Microprocessors". In: *Communications of the ACM* 62.6, pp. 83–91.

[13] Jasmina Vasiljevic et al. (2021). "Compute Substrate for Software 2.0". In: *IEEE Micro* 41.2, pp. 50–55.

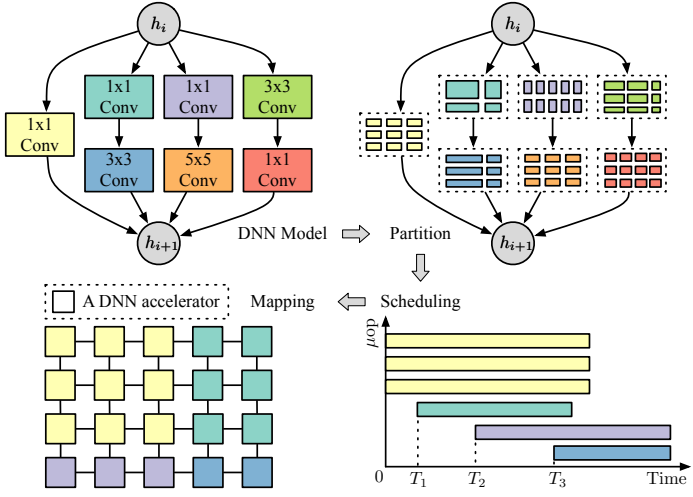
Introduction: DNN Model Orchestration

What is DNN Model Orchestration?

The orchestration of DNN models determine how to *partition*, *schedule* and *map* a DNN model to scalable DNN accelerators.

- Partition: partition a DNN computation graph into μops $\rightarrow$ Exploit higher execution parallelism.
- Schedule/Scheduling: The scheduling allocates *time slots* for each μop to attain the promising *makespan*.
- Mapping: decides the allocation of an accelerator for a μop.

Introduction: DNN Model Orchestration



A pipeline overview of DNN model orchestration for scalable DNN accelerators.

Preliminary

Preliminary: Previous Methodologies

- CNN-Partition[14]
- Tangram[15].
- Atomic dataflow[16].

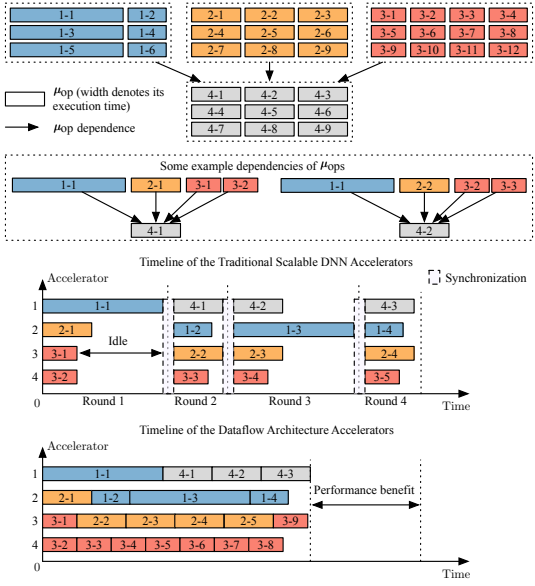
They are proposed for traditional scalable DNN accelerators rather than dataflow architecture accelerators.

[14] Yongming Shen, Michael Ferdman, and Peter Milder (2017). "Maximizing CNN accelerator efficiency through resource partitioning". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, pp. 535–547.

[15] Mingyu Gao et al. (2019). "Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 807–820.

[16] Shixuan Zheng et al. (2022). "Atomic Dataflow Based Graph-Level Workload Orchestration for Scalable DNN Accelerators". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 475–489.

Preliminary: DNN Model Orchestration Comparison



Comparison between traditional scalable DNN accelerators and dataflow architecture accelerators.

Preliminary: DNN Model Orchestration Comparison

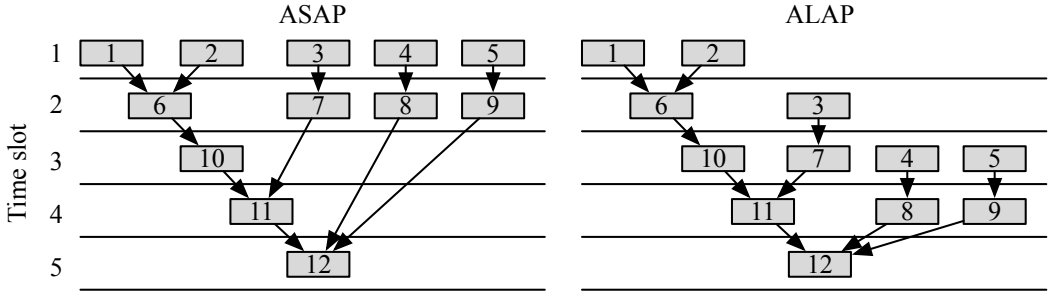
Summary of the traditional scalable DNN accelerators:

- μOps are scheduled per *round*.
- Synchronization latencies are produced between adjacent rounds.
- The "bad" μop determines the time interval of a round.

Summary of the dataflow architecture accelerators:

- "Nearly" no synchronizations.
- High throughputs.

Preliminary: ASAP & ALAP Scheduling



An overview of ASAP and ALAP scheduling.

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

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

Preliminary: Problem Formulation

Definition I: μOp

The execution granularity of an individual accelerator in dataflow architecture accelerators [17].

Definition II: μOp precedence constraints

The consumer μop should not begin to execute before the producer μops are completed [18].

Definition III: Accelerator

An accelerator executes one μop at a time until its completion. Other μops cannot preempt the execution.

[17] A μop's operands are termed μtensors.

[18] We use $u_i \prec u_j$ to denote that u_i is an immediate producer of u_j.

Preliminary: Problem Formulation

Problem I: Partition

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.

Problem II: Scheduling

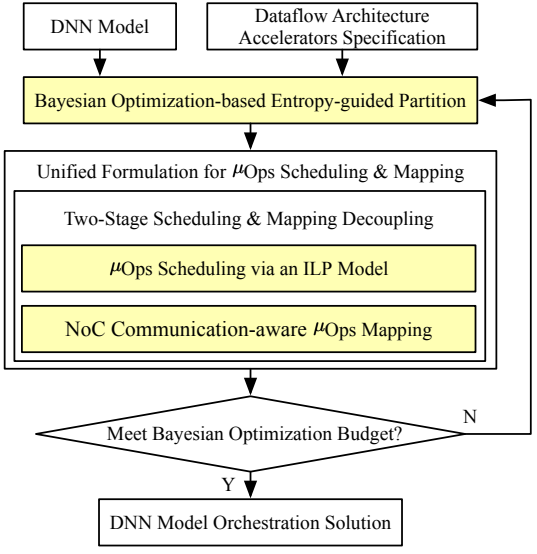
Allocate time slots for μops (the solution from Problem 1), aiming to minimize the makespan.

Problem III: Mapping

Allocate accelerators for μops (the solution from Problem 1), aiming to minimize the NoC communication costs during dataflow executions.

Klotski

Klotski: Overview [19]



An overview of Klotski framework.

[19] We focus on convolutional neural networks in Klotski.

Klotski: Bayesian Optimization-based Entropy-guided Partition

Two requirements in partition:

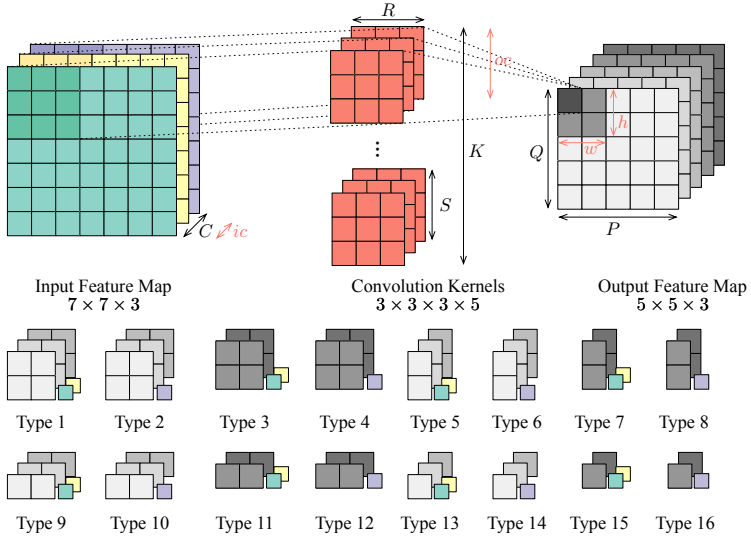
- The computation of each μop should fully utilize an accelerator's resources.
- The computation latency of all μops should be as close as possible to achieve load balancing.

How do we partition?

- A unified representation of the partition strategy.

Example: a convolution layer's shape is denoted as a tuple (R, S, P, Q, C, K), and we use $\boldsymbol{s}(h, w, ic, oc)$ to partition the output tensor.

Klotski: Bayesian Optimization-based Entropy-guided Partition



An example shows a partition with $\boldsymbol{s}(2, 2, 2, 3)$.

Klotski: Bayesian Optimization-based Entropy-guided Partition

Two requirements:

- The computation of each μop should fully utilize an accelerator's resources.
- The computation latency of all μops should be as close as possible to achieve load balancing.

Corresponding proposed solutions:

- The 1^{st} requirement $\rightarrow$ Divisible by corresponding PE dimensions.
- The 2^{nd} requirement $\rightarrow$ Bayesian optimization-based design space exploration solution.

Algorithm 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, and 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.

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

Klotski: Unified Formulation for μOps Scheduling & Mapping

The main algorithm flow for the unified formulation:

1. $\rightarrow$ Acquire the upper bound of the makespan by list scheduling[2021] [22].
2. Acquire the scheduling flexibility by ASAP & ALAP.
3. $\rightarrow$ Define the solution with a binary tensor $\boldsymbol{\mathcal{X}}$.
4. $\rightarrow$ Construct constraints for the scheduling & mapping.
5. $\rightarrow$ Construct optimization objectives.
6. Solve the model with off-the-shelf solvers.

[20] Ronald L. Graham (1966). "Bounds for Certain Multiprocessing Anomalies". In: *Bell system technical journal* 45.9, pp. 1563–1581.

[21] Ronald L. Graham (1969). "Bounds on Multiprocessing Timing Anomalies". In: *SIAM journal on Applied Mathematics* 17.2, pp. 416–429.

[22] $\rightarrow$ denotes that we only focus on these steps in slides.

Step 1: List scheduling gives the upper bound of the makespan for μops scheduling.

Theorem (Upper Bound of the Makespan for μOps Scheduling)

List scheduling achieves $2 - 1/\|\boldsymbol{a}\|$ *times the optimal makespan for dataflow architecture accelerators, where* $\|\boldsymbol{a}\|$ *is the number of individual accelerators.*

Denote the upper bound as T, μop set V, and the accelerator vector $\boldsymbol{a}$.

Step 3: A binary tensor $\boldsymbol{\mathcal{X}}$ with the size of $|V| \times T \times \|\boldsymbol{a}\|$.

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

Klotski: Unified Formulation for μOps Scheduling & Mapping

Step 4: list of constraints.

- μOp constraint: scheduling flexibility & issue constraint.

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

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

- μOps precedence constraint:

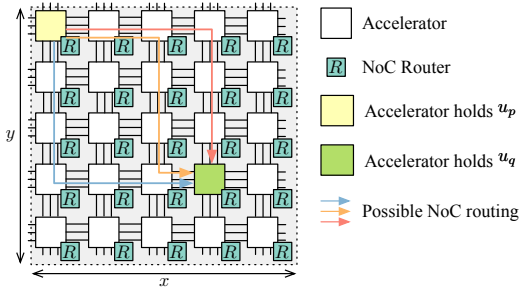
$$\sum_{k=1}^{\|\boldsymbol{a}\|}\sum_{j=S_p}^{L_p}j\cdot\boldsymbol{\mathcal{X}}_{pjk}-\sum_{k=1}^{\|\boldsymbol{a}\|}\sum_{j=S_q}^{L_q}j\cdot\boldsymbol{\mathcal{X}}_{qjk}\le -l(u_p), \tag{7}$$
$$\forall u_p,\ \forall u_q\in V \text{ and } u_p\prec u_q.$$

- Computing resource constraint:

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

Step 5: Optimization objectives include T and NoC communication cost.

A NoC with mesh topology and XY-YX routing algorithm[23].



An overview of an NoC communication.

[23] Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh (2017). "On-chip Networks". In: *Synthesis Lectures on Computer Architecture* 12.3, pp. 1–210.

The NoC communication cost from μop u_p to μop u_q is [24]

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

where u_p's assigned accelerator is at (x_1, y_1), and the acclerator for u_q is at (x_2, y_2).

[24] A more realistic model includes the transmitted data volume size.

Compute the locations of accelerators with following equations:

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

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

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

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

The entire NoC communication costs:

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

Step 6: Solve with off-the-shelf solvers.

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

The length of a single time slot is determined by $\min\ l(u_i),\ \forall u_i \in V$, where β is a coefficient to trade-off T and C.

Klotski: Unified Formulation for μOps Scheduling & Mapping

Limitations of the unified formulation:

- It costs high runtime to construct constraints like Equation (8).
- Non-linearity in Equation (9), Equation (10), and Equation (11).

We propose a two-stage scheduling & mapping decoupling methodology accordingly.

Klotski: Two-Stage Scheduling & Mapping Decoupling

The main algorithm flow for the two-stage scheduling & mapping decoupling:

1. → Acquire the upper bound of the makespan by list scheduling[25][26] [27].
2. Acquire the scheduling flexibility by ASAP & ALAP.
3. → Decouple the unified formulation with binary matrice $\boldsymbol{X}$ and $\boldsymbol{Y}$ [28].
4. → Construct the scheduling model & solve with off-the-shelf solvers.
5. → Construct the mapping model & solve with off-the-shelf solvers.

[25] Ronald L. Graham (1966). "Bounds for Certain Multiprocessing Anomalies". In: *Bell system technical journal* 45.9, pp. 1563–1581.

[26] Ronald L. Graham (1969). "Bounds on Multiprocessing Timing Anomalies". In: *SIAM journal on Applied Mathematics* 17.2, pp. 416–429.

[27] → denotes that we only focus on these steps in slides.

[28] → denotes that we only focus on these steps in slides.

Step 3: A $|V| \times T$ binary matrix $\boldsymbol{X}$ as a scheduling solution and a binary matrix $\boldsymbol{Y}$ with the size of $|V| \times \|\boldsymbol{a}\|$ as the mapping solution.

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

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

Step 4: Construct the scheduling model & solve with off-the-shelf solvers.

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

We relax the last constraint of Equation (18) with Equation (19) [29].

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

[29] The equation is customized to dataflow architecture accelerators only, but we omit rationales in slides.

Step 5: Construct the mapping model & solve with off-the-shelf solvers.

We introduce new variables to transform the mapping problem as mixed-integer linear programming. Take an example from Equation (10) to Equation (13).

Klotski: Two-Stage Scheduling & Mapping Decoupling

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, the communication between u_i and u_j is formulated.

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

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

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

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

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

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

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

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

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

Experiments

Experiments: Implementation Details

- We build an in-house simulator for the dataflow architecture accelerators.
- We use MAESTRO[30] as performance model for individual accelerators.
- We use *nn_dataflow*[31] as the front end of DNN models, and we implement the partition based on the framework.
- We use Gurobi v10.0[32] as the off-the-shelf solver.

[30] Hyoukjun Kwon et al. (2019). "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: A Data-Centric Approach". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 754–768.

[31] *nn_dataflow: a Neural Network Dataflow Scheduling Tool* (n.d.). https://github.com/stanford-mast/nn_dataflow.

[32] LLC Gurobi Optimization (2020). *Gurobi Optimizer Reference Manual (2020)*.

Experiments: Baselines & Workloads

Baselines:

- No previous methods for DNN model orchestration were proposed for dataflow architecture accelerators.
- We set up our baselines based on Tangram[33] and the atomic dataflow[34] with rationales.
- We term them as "baseline 1" and "baseline 2".

Workloads:

- VGG16, VGG19, ResNet50, ResNet152, and Inception v3.

Topologies:

- 3×3.
- 4×4.
- 5×5.

[33] Mingyu Gao et al. (2019). "Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 807–820.

[34] Shixuan Zheng et al. (2022). "Atomic Dataflow Based Graph-Level Workload Orchestration for Scalable DNN Accelerators". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 475–489.

Experiments: Comparison to Baselines

Table: 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

Experiments: Comparison to Baselines

Table: 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

Experiments: Comparison to Baselines

Table: The experimental results for the 5×5 topology

Workload	Method	5×5				
		Cycles	Ratio	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

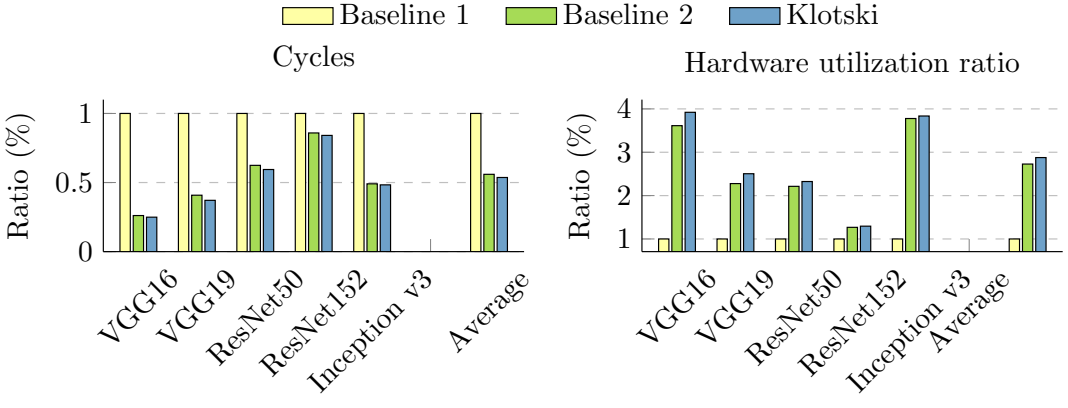
Experiments: Comparison to Baselines

Summary

- 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 costs higher runtime than baselines due to that Klotski leverages much time to solve the scheduling and mapping in the two-stage methodology.

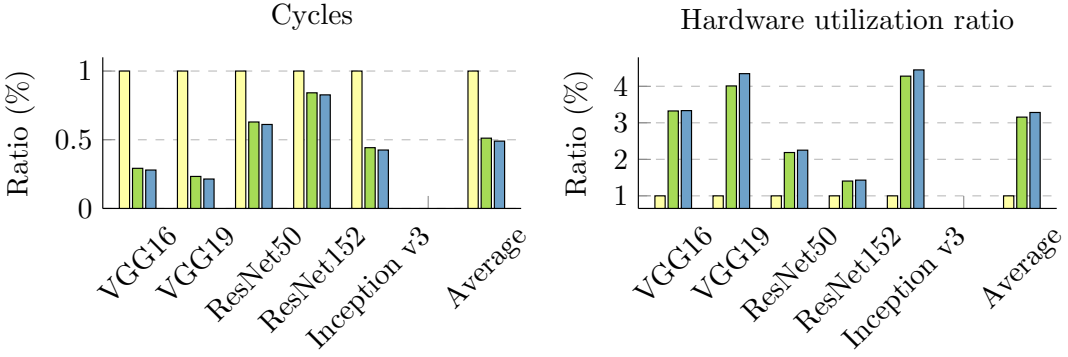
Experiments: Ablation Study

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.



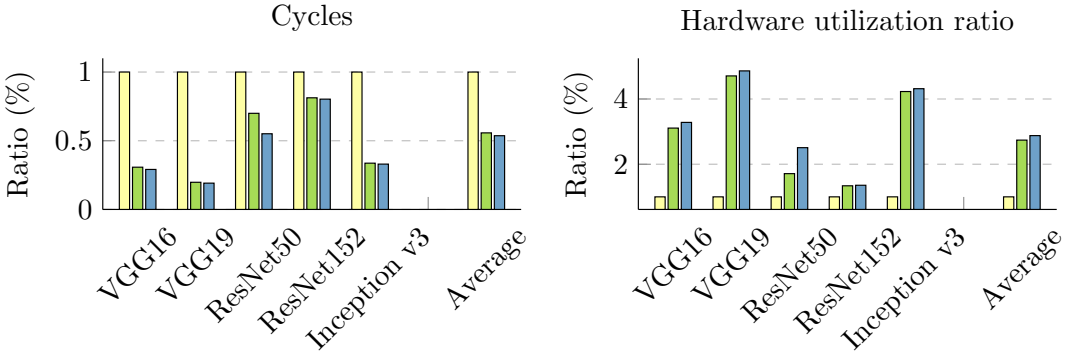
Results of the 3×3 topology.

Experiments: Ablation Study



Results of the 4×4 topology.

Experiments: Ablation Study



Results of the 5×5 topology.

Experiments: Ablation Study

Summary

- 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.

Experiments: Lessons Learned

Summary

- Partitioning a DNN model into μops allows better execution performance, even for cascaded layers structures.
- The improvement of the execution performance is non-linear to the increased hardware utilization ratio. $\rightarrow$ Analytical partition solution?

Conclusion

Conclusion

- 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 dataflow architecture accelerators.
- A two-stage methodology decoupling the unified formulation is proposed to make the solution feasible.
- Extensive results show that Klotski can achieve 9.55% and 48.48% higher execution performance improvement than baselines.

THANK YOU!

References I

Dennis Abts, Garrin Kimmell, et al. (2022). "A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 567–580.

Dennis Abts, Jonathan Ross, et al. (2020). "Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 145–158.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze (2016). "Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 367–379.

Zidong Du et al. (2015). "ShiDianNao: Shifting Vision Processing Closer to the Sensor". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 92–104.

Mingyu Gao et al. (2019). "Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 807–820.

References II

Hasan Genc et al. (2021). “Gemmini: Enabling Systematic Deep-learning Architecture Evaluation Via Full-stack Integration”. In: *ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 769–774.

Ronald L. Graham (1966). “Bounds for Certain Multiprocessing Anomalies”. In: *Bell system technical journal* 45.9, pp. 1563–1581.

— (1969). “Bounds on Multiprocessing Timing Anomalies”. In: *SIAM journal on Applied Mathematics* 17.2, pp. 416–429.

LLC Gurobi Optimization (2020). *Gurobi Optimizer Reference Manual (2020)*.

Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh (2017). “On-chip Networks”. In: *Synthesis Lectures on Computer Architecture* 12.3, pp. 1–210.

Norm Jouppi et al. (2023). “TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings”. In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 1–14.

Norman P Jouppi et al. (2017). “In-datacenter Performance Analysis of a Tensor Processing Unit”. In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 1–12.

References III

Hyoukjun Kwon et al. (2019). “Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: A Data-Centric Approach”. In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 754–768.

nn_dataflow: a Neural Network Dataflow Scheduling Tool (n.d.). https://github.com/stanford-mast/nn_dataflow.

Tony Nowatzki, Vinay Gangadhar, and Karthikeyan Sankaralingam (2019). “Heterogeneous Von Neumann/Dataflow Microprocessors”. In: *Communications of the ACM* 62.6, pp. 83–91.

Albert Reuther et al. (2022). “AI and ML Accelerator Survey and Trends”. In: *IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, pp. 1–10.

Yongming Shen, Michael Ferdman, and Peter Milder (2017). “Maximizing CNN accelerator efficiency through resource partitioning”. In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, pp. 535–547.

The NVIDIA Deep Learning Accelerator (NVDLA) (2017). http://nvdla.org/.

Jasmina Vasiljevic et al. (2021). “Compute Substrate for Software 2.0”. In: *IEEE Micro* 41.2, pp. 50–55.

References IV

Swagath Venkataramani et al. (2017). “SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks”. In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 13–26.

Shixuan Zheng et al. (2022). “Atomic Dataflow Based Graph-Level Workload Orchestration for Scalable DNN Accelerators”. In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 475–489.