Adapt-Flow: A Flexible DNN Accelerator Architecture for Heterogeneous Dataflow Implementation

Jiaqi Yang
Yang_Jiaqi_Cute@gwu.edu
The George Washington University
Washington DC, USA

Hao Zheng
hao.zheng@ucf.edu
University of Central Florida
Orlando, Florida, USA

Ahmed Louri
louri@gwu.edu
The George Washington University
Washington DC, USA

ABSTRACT

Deep neural networks (DNNs) have been widely applied to various application domains. DNN computation is memory and compute-intensive requiring excessive memory access and a large number of computations. To efficiently implement these applications, several data reuse and parallelism exploitation strategies, called dataflows, have been proposed. Studies have shown that many DNN applications benefit from a heterogeneous dataflow strategy where the dataflow type changes from layer to layer. Unfortunately, very few existing DNN architectures can simultaneously accommodate multiple dataflows due to their limited hardware flexibility. In this paper, we propose a flexible DNN accelerator architecture, called Adapt-Flow, which has the capability of supporting multiple dataflow selections for each DNN layer at runtime. Specifically, the proposed Adapt-Flow architecture consists of (1) a flexible interconnect, (2) a dataflow selection algorithm, and (3) a dataflow mapping technique. The flexible interconnect provides dynamic support for various traffic patterns required by different dataflows. The proposed dataflow selection algorithm selects the optimal dataflow strategy for a given DNN layer with the aim of much improved performance. And the dataflow mapping technique efficiently maps the dataflow amenable to the flexible interconnect. Simulation studies show that the proposed Adapt-Flow architecture reduces execution time by 46%, 78%, 26%, and energy consumption by 45%, 80%, 25% as compared to NVDLA [1], ShiDianNao [2], and Eyeriss [3] respectively.

CCS CONCEPTS

• **Computer systems organization → Interconnection architectures**; • **Hardware → Networking hardware**.

KEYWORDS

DNN, accelerator, Network on Chip, dataflow, DRAM access, flexible architecture

ACM Reference Format:
Jiaqi Yang, Hao Zheng, and Ahmed Louri. 2022. Adapt-Flow: A Flexible DNN Accelerator Architecture for Heterogeneous Dataflow Implementation. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526241.3530311

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GLSVLSI '22, June 6–8, 2022, Irvine, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9322-5/22/06…$15.00
https://doi.org/10.1145/3526241.3530311

1 INTRODUCTION

Deep neural networks (DNNs) have been widely deployed in many application domains such as virtual reality [4], image recognition [5], signal processing [6], and many others [7]. These DNN applications are structured with a collection of layers with heterogeneous size and connectivity due to their distinct properties [8–11]. Such heterogeneity leads to a wide variation in computation needs and data movement volume [8, 11–13], and it also indicates the opportunity in the selection of parallelism and data reuse strategies, called dataflow, from layer to layer. This could lead to up to 51% of difference in DRAM access volume when considering different dataflows per our simulation studies using AlexNet model [5].

Though a large number of architectures have been proposed and implemented in support of various dataflows, few prior works are agile enough to simultaneously accommodate multiple dataflows within a single application. The key obstacle is the limited flexibility of the underlying communication fabrics for implementing data movement between the global buffer and the processing array. The interconnect fabric plays a critical role in implementing data reuse strategies needed for a given dataflow.

Significant research efforts have been devoted to the design of flexible interconnects, but many have limited applicability in DNN accelerators, in particular supporting multiple dataflows. For example, SMART [14] and Adapt-NoC [15, 16] have been proposed to enable shortcut or path diversity on top of grid-like (e.g., mesh) topology for general purpose processors. However, the long-distance communication between processing elements (PEs) is rarely seen in DNN accelerators. Reconfigurability [8, 17] has been explored in DNN accelerators to customize the interconnects for multiply-accumulate (MAC) units. As such, a collection of MAC units can adapt to different dataflows eliminating the data sparsity in a fine-grained manner. However, such fine-grained reconfigurability results in prohibitive latency and hardware overheads.

In this paper, we address the main bottleneck for the dynamic support of multiple dataflows and the ability to switch dataflow strategies from layer to layer. We identify the communication between the PE array and the global buffer as a major impediment, as it directly impacts the choice of data reuse and parallelism. Leveraging this understanding, we propose a flexible DNN accelerator architecture, called Adapt-Flow, which allows each DNN layer to select optimal dataflow with its desired data reuse and parallelism exploitation. Specifically, Adapt-Flow consists of (1) a flexible interconnect, (2) a dataflow selection algorithm, and (3) a dataflow mapping strategy. The flexible interconnect exploits the unique properties of the

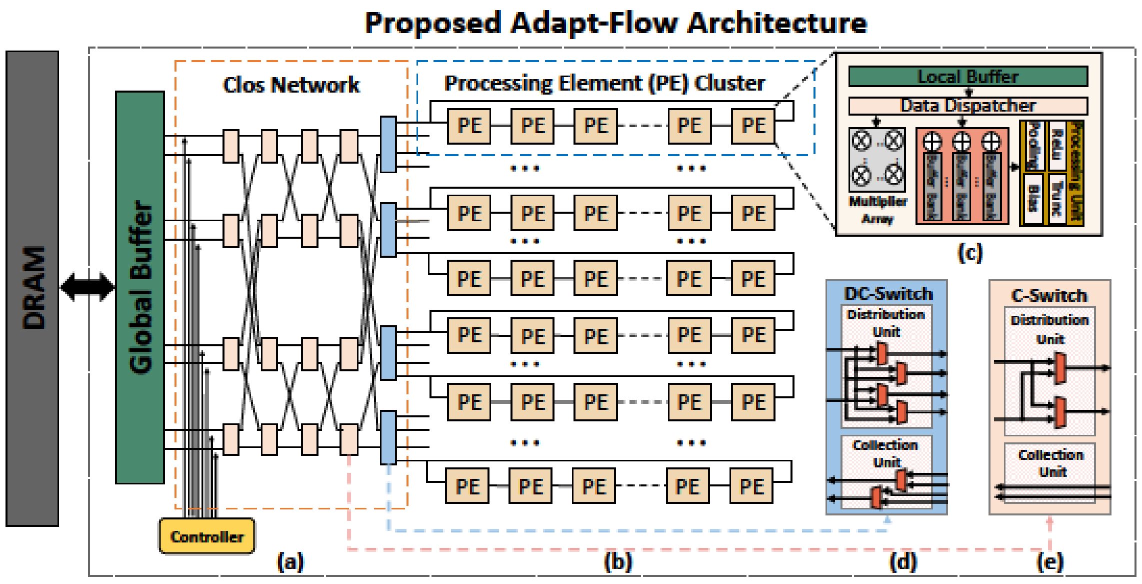


Figure 1: Overview of the proposed Adapt-Flow architecture. (a)Architecture of bidirectional Clos Network, (b)Layout of PE array(16×8) with ring topology, (c)Architecture of Processing Element (PE), (d)Architecture of diverged bidirectional Clos switch (DC-Switch), (e)Architecture of bidirectional Clos switch (C-Switch).

Clos network to seamlessly support unicast, multicast, and broadcast communication functions often seen in various dataflows [18]. The dataflow selection strategy selects the optimal dataflow at each layer. And the dataflow mapping technique efficiently maps the selected the dataflow onto the architecture. We evaluate the proposed Adapt-Flow using several popular DNN benchmarks such as AlexNet, VGG16 [19], ResNet50 [20] and UNet [21]. Simulation results show that Adapt-Flow design reduces execution time by 46%, 78%, 26%, and energy consumption by 45%, 80%, 25% as compared to NVDLA [1], ShiDianNao [2], and Eyeriss [3] respectively.

2 ADAPT-FLOW ARCHITECTURE

The goal of Adapt-Flow architecture is to enable flexible dataflow selection for each DNN layer, thus maximizing data-reuse opportunities and minimizing DRAM accesses. More specifically, Adapt-Flow provides several dataflow options including weight stationery (WS), input stationery (IS), output stationery (OS) dataflows [7].

For the sake of simplicity, we illustrate the Adapt-Flow architecture using a 128-PE setup. As shown in Figure 1, Adapt-Flow consists of a global buffer, a bi-directional Clos network [18, 22] including 2×4 diverged Clos switches as shown in Figure 1 (d) and 2×2 Clos switches as shown in Figure 1 (e), and 128 PEs arranged as an array. The global buffer is connected to the PE array using the Clos network as shown in Figure 1 (a). Within the PE array illustrated in Figure 1 (b), PEs are connected by a bi-directional ring at each row. Each PE consists of a 4×4 multiplier array for vector multiplication and a local buffer for the temporary storage of weights, input activations, and output activations as shown in Figure 1 (c).

2.1 Proposed Flexible Interconnect

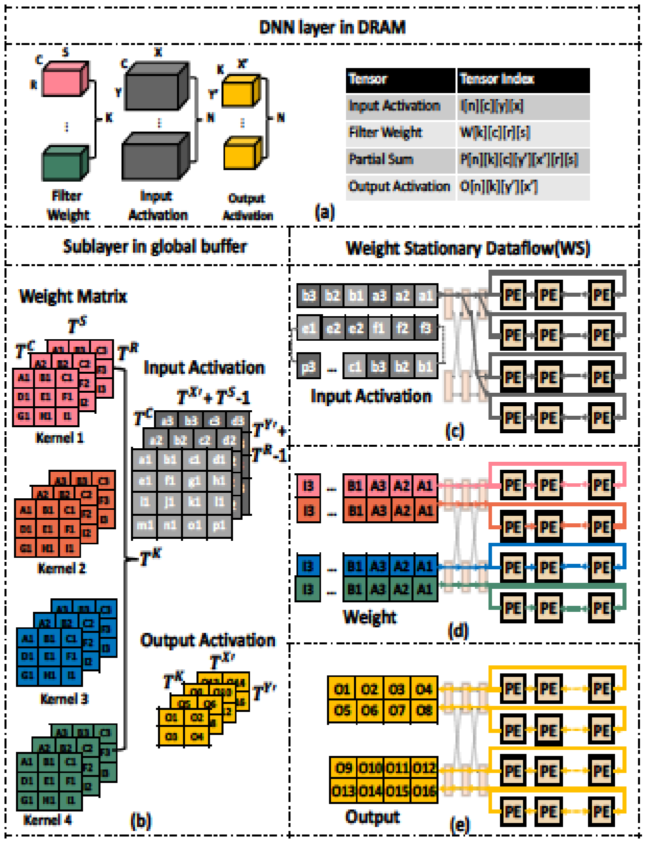
The proposed flexible interconnect is shown in Figure 1, and consists of a bi-directional Clos network with the ability to provide adequate support for various communication functions required by the dataflows considered. As shown in Table 1, various dataflows require distinct communication patterns for handling weights, input activations, and psum to take advantage of the spatial-temporal data reuse opportunities. This puts a huge burden on the communication fabric and the proposed Clos Network provides the needed flexibility and capability.

Table 1: Traffic Pattern of Different Dataflows

Dataflow	Weight	Input Activation	Psum
WS	Unicast	Multicast	Unicast
OS	Broadcast	Unicast	Unicast
IS	Multicast	Unicast	Unicast

2.1.1 Communication Between Global Buffer and PE Array. The bi-directional switches of the Clos network (shown in Figure 1) can provide non-blocking and parallel communication when transmitting the data from the global buffer to the the PE array and can provide bypassing connection when transmitting the data back from the PE array to the global buffer. These switches can be set up by the controller for each DNN layer.

To better illustrate the process of communication between global buffer and PE array, we use WS dataflow as an example with the



Tensor	Tensor Index
Input Activation	I[n][c][y][x]
Filter Weight	W[k][c][r][s]
Partial Sum	P[n][k][c][y'][x'][r][s]
Output Activation	O[n][k][y'][x']

Figure 2: An example of our proposed accelerator processing a simple tiled convolutional layer using WS dataflow.

stride of 1 (Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video). As shown in Figure 2(a,b), weight (K, C, S, R), input activation (N, C, X, Y), and output (N, K, X', Y') are tiled into small portion of data, called sublayer, fitting the global size, where tiling factors are considered as $(T_i^k, T_i^c, T_i^s, T_i^r, T_i^{x'}, T_i^{y'})$. As such, weight matrix $(T_i^k, T_i^c, T_i^s, T_i^r)$, input activation $(T_i^c, T_i^{x'} + T_i^s - 1, T_i^{y'} + T_i^r - 1)$, and output activation $(T_i^k, T_i^{x'}, T_i^{y'})$ will be moved from the global buffer to the PE array.

In such a case, the bi-directional switches can be set to broadcast input activation data (a1, a2, a3, etc.) to multiple rows of the PE array, where the circuit switching allows the data to be broadcast to multiple rows of PE array shown in Figure 2(c). In Figure 2(d), multiple data paths can be simultaneously set to transmit weights to different rows of the PE array. For the output, partial sums are accumulated across each row and eventually written back to the global buffer using the Clos network as shown in Figure 2(e). As can be seen, WS communication requirements, namely input activation broadcast, weight transmission, and psum gathering and accumulation all happen efficiently.

2.1.2 Inter-PE Communication. The communication between PEs within the PE array can be of two types, namely psum accumulation

```
Algorithm : Dataflow Selection
  Input   :Features of n dataflows: $DF_i, i \in \{1, \cdots, n\}$
  Input   :Capacity of global buffer: $GLB$
  Input   :Data dimension of DNN layer
  Output  :Optimal dataflow
begin
    $DA \leftarrow \{\}$ ; // The DRAM access volume for all dataflow.
    /* Iterating all dataflows */
    for $i = 1$ to $n$ do
        Calculate $V_i^d, d \in \{wt, ifmap, psum\}$ ; // Data volume involved in each invocation
        Calculate $R_i^d, d \in \{wt, ifmap, psum\}$ ; // Number of invocations
        Calculate $G_i$ ; // Global buffer capacity requirement
        if $G_i \leq GLB$ then
            /* Calculate the DRAM access volume if satisfying global buffer requirement. */
            $DA_i \leftarrow \sum_d V_i^d \times R_i^d, d \in \{wt, ifmap, psum\}$
        else
            /* Otherwise don't consider by assigning as infinity. */
            $DA_i \leftarrow \infty$
        end
    end
    /* Return the dataflow with minimum DRAM access volume. */
    return $\{DF_i | DA_i = \min\{DA\}\}$
end
```

Figure 3: Dataflow Selection Algorithm.

as well as input activation/weight transmission (unicast) from PE to PE [23]. For psum accumulation, the psums are forwarded from PE to PE calculating the final result. In other words, the average hop count for psum accumulation would be a constant number, and this can be handled by any simple topology such as a torus or a ring. In order to manage various psum movement directions, we use a bi-directional ring to support the unicast of weights and input activations for the proposed architecture. The bi-directional ring can provide sufficient bandwidth, reduced diameter, with a reasonable cost and design complexity for the proposed architecture.

2.2 Dataflow Selection Strategy

In this section, we present the algorithm for determining the optimal dataflow for each DNN layer. As shown in Figure 3, the process consist of estimating the number of DRAM access volume for each DNN layer with different data dimensions (N, K, C, S, R, X', Y'). In general, our proposed model considers unique features of each dataflow such as loop order (LO_i) and tiling factors $(T_i^k, T_i^c, T_i^s, T_i^r, T_i^{x'}, T_i^{y'})$.

To estimate the DRAM access volume (DA_i) for the i-th dataflow of the dataflow candidates, we calculate the product of the data volume involved in each invocation (V_i^d) and the total number of invocations (R_i^d) for all data types (i.e., weight(wt), input activation($ifmap$), psum($psum$)) as shown in Equation 1:

$$DA_i = \sum_d V_i^d \times R_i^d, d \in \{wt, ifmap, psum\} \tag{1}$$

The data volume involved in each invocation for different data type can be calculated as V_i^{wt}, V_i^{ifmap}, V_i^{psum} as depicted in Equation 2:

$$\begin{aligned} V_i^{wt} &= T_i^k \times T_i^c \times T_i^s \times T_i^r \\ V_i^{ifmap} &= T_i^c \times (T_i^{x'} + T_i^s - 1) \times (T_i^{y'} + T_i^r - 1) \\ V_i^{psum} &= T_i^k \times T_i^{x'} \times T_i^{y'} \end{aligned} \tag{2}$$

The tiling factors determine the accessed array regions of each sublayer, thus different tiling factor configurations imply different dataflows. We observed that different dataflows exhibit large differences in DRAM access volume. The total number of invocations can be calculated as R_i^{wt}, R_i^{ifmap}, R_i^{psum}. According to the different loop orders (LO_i), the total number of invocations (R_i^d) has different formulas to calculate:

If the loop order (LO_i) is weight stationary, the total number of invocations (R_i^d) can be calculated as depicted in Equation 3:

$$\begin{aligned} R_i^{wt} &= \frac{K \times C \times S \times R}{T_i^k \times T_i^c \times T_i^s \times T_i^r} \\ R_i^{ifmap} &= \frac{N \times K \times C \times S \times R \times X' \times Y'}{T_i^k \times T_i^c \times T_i^s \times T_i^r \times T_i^{x'} \times T_i^{y'}} \\ R_i^{psum} &= \frac{N \times K \times S \times R \times X' \times Y' \times (\frac{2C}{T_i^c} - 1)}{T_i^k \times T_i^s \times T_i^r \times T_i^{x'} \times T_i^{y'}} \end{aligned} \tag{3}$$

Similarly, the total number of invocations (R_i^d) can be calculated using Equation 4 for input stationary.

$$\begin{aligned} R_i^{wt} &= \frac{N \times K \times C \times S \times R \times X' \times Y'}{T_i^k \times T_i^c \times T_i^s \times T_i^r \times T_i^{x'} \times T_i^{y'}} \\ R_i^{ifmap} &= \frac{N \times C \times X' \times Y'}{T_i^c \times T_i^{x'} \times T_i^{y'}} \\ R_i^{psum} &= \frac{N \times K \times S \times R \times X' \times Y' \times (\frac{2C}{T_i^c} - 1)}{T_i^k \times T_i^s \times T_i^r \times T_i^{x'} \times T_i^{y'}} \end{aligned} \tag{4}$$

The total number of invocations (R_i^d) can be calculated as depicted in Equation 5 for output stationary.

$$\begin{aligned} R_i^{wt} &= \frac{N \times K \times C \times S \times R \times X' \times Y'}{T_i^k \times T_i^c \times T_i^s \times T_i^r \times T_i^{x'} \times T_i^{y'}} \\ R_i^{ifmap} &= \frac{N \times K \times C \times S \times R \times X' \times Y'}{T_i^k \times T_i^c \times T_i^s \times T_i^r \times T_i^{x'} \times T_i^{y'}} \\ R_i^{psum} &= \frac{N \times K \times X' \times Y'}{T_i^k \times T_i^{x'} \times T_i^{y'}} \end{aligned} \tag{5}$$

In this work, we also include hardware parameter to refine the selection. The global buffer capacity of the proposed architecture (GLB) needs to be larger than the global buffer capacity requirement. In other words, any dataflow exceeding the global buffer capacity requirement will not be considered in the proposed algorithm. Given the different data type P_{wt}, P_{ifmap}, P_{psum} and the data volume involved in each invocation for different data type V_i^{wt}, V_i^{ifmap}, V_i^{psum}, the global buffer capacity requirement (G_i) can be calculated using Equation 6:

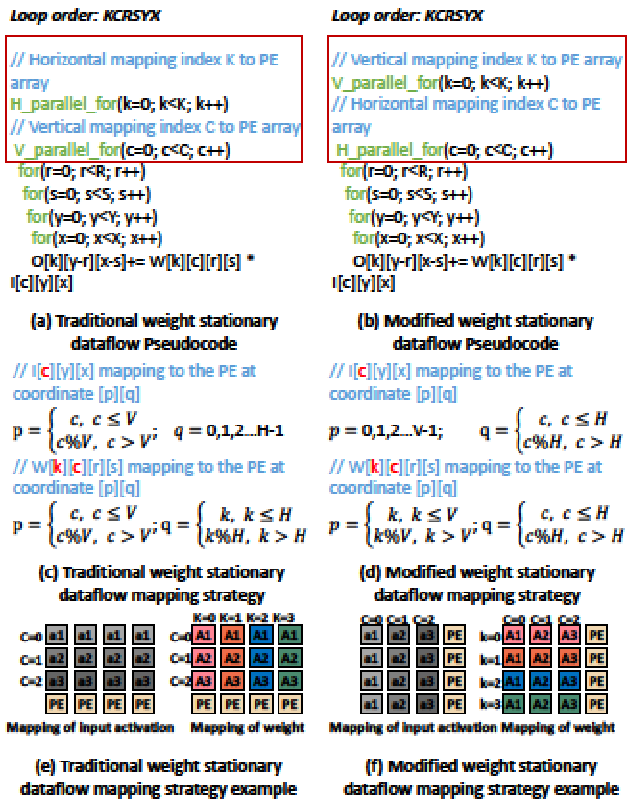
$$G_i = \sum_d V_i^d \times P_d, d \in \{wt, ifmap, psum\} \tag{6}$$

We compare the total DRAM access volume (DA_i) of different dataflows and consider the dataflow with minimal DRAM access volume as the optimal one. We note that the layer-wise dataflow selection will be calculated before execution time, and therefore it will not have any negative impact on the execution. The dataflow selection algorithm is described in detail in Figure 3.

2.3 Dataflow Mapping for Adapt-Flow Architecture

Once a selected dataflow strategy is selected (data reuse and parallelism), we need to map it onto the architecture. Without loss of generality, we illustrate the mapping strategy using WS dataflow. It should be noted that the mapping can be applied to all the dataflows.

In WS dataflow, the matrix-matrix multiplication is performed in a nested loop $KCRSYX$, where each input activation ($I[c][y][x]$) is multiplied with each weight ($W[k][c][r][s]$) as shown in Figure 4 (a,b). To execute the multiplication in parallel, input channels (C) and output channels (K) should be partitioned and distributed to an array of PEs ($PE[p][q]$). In a typical WS-optimized accelerator (i.e. NVDLA-style) as shown in Figure 4 (c,e), weights and input activations are distributed to PEs in form of unicast and broadcast communications through the row-wise bus. However, the ring topology would be inefficient in handling the broadcast communication at each row. In Figure 4 (d,f), the modified mapping strategy distributes the input activations and weights column-wise, resulting in column-wise broadcast communication. The key idea of our strategy is to map computations with broadcast communication demand (spatial optimization) in the same column and the ones with unicast communication demand (temporal optimization) in the same row.



Loop order: KCRSYX

```
// Horizontal mapping index K to PE array
H_parallel_for(k=0; k<K; k++)
// Vertical mapping index C to PE array
V_parallel_for(c=0; c<C; c++)
 for(r=0; r<R; r++)
  for(s=0; s<S; s++)
   for(y=0; y<Y; y++)
    for(x=0; x<X; x++)
     O[k][y-r][x-s]+= W[k][c][r][s] * I[c][y][x]
```

(a) Traditional weight stationary dataflow Pseudocode

Loop order: KCRSYX

```
// Vertical mapping index K to PE array
V_parallel_for(k=0; k<K; k++)
// Horizontal mapping index C to PE array
H_parallel_for(c=0; c<C; c++)
 for(r=0; r<R; r++)
  for(s=0; s<S; s++)
   for(y=0; y<Y; y++)
    for(x=0; x<X; x++)
     O[k][y-r][x-s]+= W[k][c][r][s] * I[c][y][x]
```

(b) Modified weight stationary dataflow Pseudocode

// I[c][y][x] mapping to the PE at coordinate [p][q]

$$p = \begin{cases} c, & c \leq V \\ c\%V, & c > V \end{cases}; \quad q = 0,1,2 \ldots H-1$$

// W[k][c][r][s] mapping to the PE at coordinate [p][q]

$$p = \begin{cases} c, & c \leq V \\ c\%V, & c > V \end{cases}; q = \begin{cases} k, & k \leq H \\ k\%H, & k > H \end{cases}$$

(c) Traditional weight stationary dataflow mapping strategy

// I[c][y][x] mapping to the PE at coordinate [p][q]

$$p = 0,1,2 \ldots V-1; \quad q = \begin{cases} c, & c \leq H \\ c\%H, & c > H \end{cases}$$

// W[k][c][r][s] mapping to the PE at coordinate [p][q]

$$p = \begin{cases} k, & k \leq V \\ k\%V, & k > V \end{cases}; q = \begin{cases} c, & c \leq H \\ c\%H, & c > H \end{cases}$$

(d) Modified weight stationary dataflow mapping strategy

(e) Traditional weight stationary dataflow mapping strategy example

(f) Modified weight stationary dataflow mapping strategy example

Figure 4: The differences between proposed mapping strategy and traditional mapping strategy for weight stationary dataflow.

3 EVALUATION METHODOLOGY

3.1 Simulation Setup

We extend MAESTRO[11] simulator to support the non-uniform latency and bandwidth between PEs. The extended simulation framework is used to calculate arithmetic operation, and access to memory both to the global buffer and the local buffers, taking the dataflow and system configuration parameters into account. The number of arithmetic operations is used to calculate the computation time, while the number of accesses to memory is used to calculate the on-package communication time. We take the Clos network and bandwidth limit into account when calculating the on-package communication time. The overall execution time is derived by calculating computation time, the on-package communication time, and the off-package communication time, considering the overlap caused by the buffering of the GB and other memory accesses.

Baseline: We compare the proposed design with three accelerator architectures, namely NVDLA[1], ShiDianNao[2], and Eyeriss[3]. We use AlexNet[5], Vgg16[19], ResNet50[20], and UNet[21] DNN models as our workloads. The proposed Adapt-Flow architecture can support multiple dataflows at the same time. For a fair comparison, we keep the configurations consistent for all designs: All

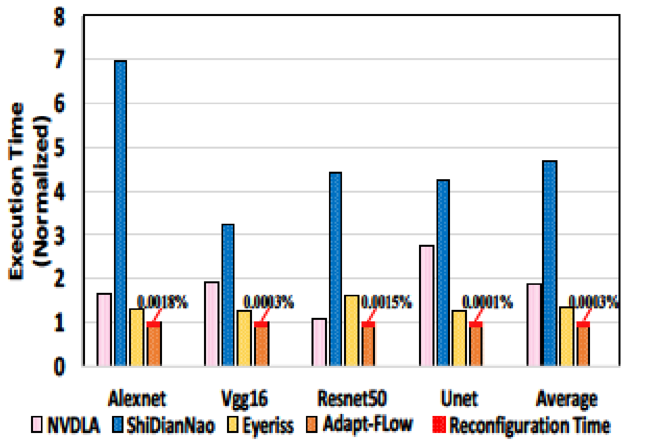


Figure 5: Normalized latency of different accelerators for DNN models(AlexNet[5], Vgg16[19], ResNet50[20], and UNet[21]), normalized to Adapt-Flow and the reconfiguration time of Clos network.

designs use 256 processing elements, and each processing element contains 16 MAC units, the SRAM of each processing elements is 5KB and the global buffer capacity is 20MB.

3.2 Execution Time Analysis

Figure 5 shows the normalized execution time for NVDLA[1], ShiDianNao[2], and Eyeriss[3], and for the proposed Adapt-Flow. As can be seen, Adapt-Flow outperforms other architectures, because prior designs can only select fixed dataflow across different DNN applications and DNN layers. This limits their ability in exploiting data reuse and minimizing DRAM access. And Adapt-Flow architecture can support multiple dataflow at runtime and can minimize DRAM access by dataflow selection algorithm. Furthermore, the average hop count of Adapt-Flow is lower than the previous designs (NVDLA[1], ShiDianNao[2], and Eyeriss[3]). The proposed work reduces overall execution time by 46%, 78% and 26%, when compared to NVDLA[1], ShiDianNao[2], and Eyeriss[3] on average.

We also study the latency overhead of Adapt-Flow for setting up communication paths for each DNN layer. For a 32 by 8 Clos network, the latency overhead would be 96 cycles. This latency appears to be very small when compared to the layer-wise execution time of DNN applications. Figure 5 shows the ratio of the reconfiguration time to the total execution time. The reconfiguration time represents 0.0003% of the total execution time on average.

3.3 Energy Consumption

For energy analysis, we use DSENT[24] to obtain power consumption and MAESTRO[11] to calculate execution time. It should be noted that the evaluation includes the energy consumption of the entire system including PE array, DRAM, global buffer, and interconnects. Figure 6 shows the normalized overall energy analysis of the proposed Adapt-Flow. As can be seen, Adapt-Flow improves

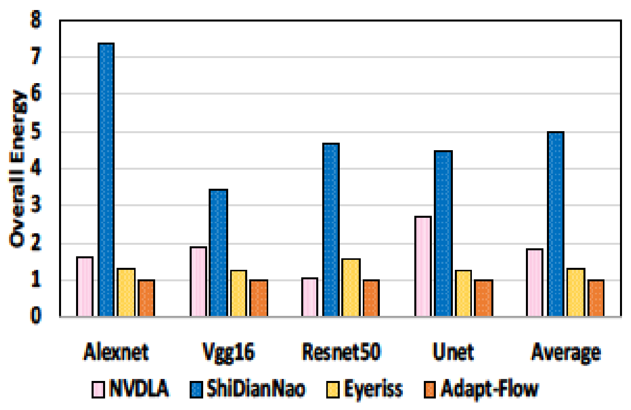


Figure 6: Normalized overall energy consumption of different accelerators for DNN models(AlexNet[5], Vgg16[19], ResNet50[20], and UNet[21]), normalized to Adapt-Flow.

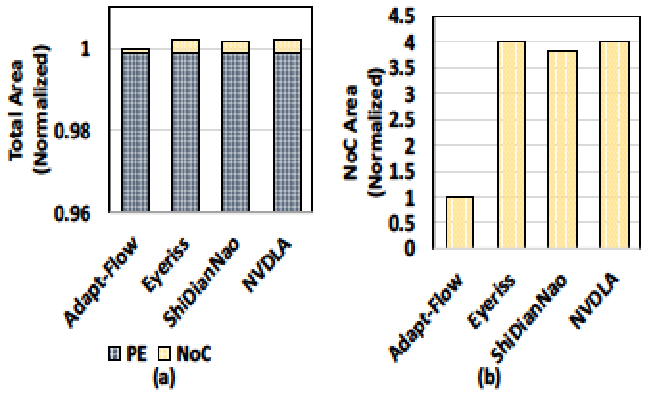


Figure 7: (a) Overall area (includes PE and NoC) and (b) NoC area overhead of different accelerators (Adapt-Flow, Eyeriss[3], ShiDianNao[2], and NVDLA[1]), normalized to Adapt-Flow.

overall energy savings by 45%, 80% and 25% when compared to NVDLA[1], ShiDianNao[2], and Eyeriss[3] on average. The energy savings mainly result from the reduced execution time, simplified architecture design, and reduced DRAM access.

3.4 Area Overhead

We evaluate the area overhead of various architectures through detailed synthesis using 45nm technology. Figure 7 shows the breakdown of the area overhead for NVDLA[1], ShiDianNao[2], and Eyeriss[3] respectively, normalized to the proposed Adapt-Flow. Adapt-Flow uses a multi-stage topology between global buffer and PE array, and for the chosen size (32×8), the Clos network contains 7 stages requiring 56 simple switches and 368 links. For a similar accelerator size, the Eyeriss[3] and NVDLA[1] use 16 buses and 256 links, ShiDianNao[2] uses a MUX array and 512 links. The NoC area of the proposed Adapt-Flow architecture is 74% smaller compared to the previous design on average, and the overall area of the proposed architecture is also reduced by 2% on average. The area savings mainly result from the simplified architecture design.

4 CONCLUSIONS

In this paper, we propose a flexible DNN accelerator architecture, called Adapt-Flow, which can provide optimal dataflow for each DNN layer at runtime. With the ability to support multiple dataflows and data reuse capabilities at runtime, the proposed architecture provides significant energy saving and reduction in overall execution time. Detailed simulation studies show that Adapt-Flow reduces execution time by 46%, 78%, 26%, and energy consumption by 45%, 80%, 25% as compared to NVDLA [1], ShiDianNao [2], and Eyeriss [3] respectively while incurring minimal overheads.

REFERENCES

[1] Nvdla deep learning accelerator, http://nvdla.org, 2017.
[2] Zidong Du et al. Shidiannao: Shifting vision processing closer to the sensor. In *proc.of ISCA*. IEEE, 2015.
[3] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 2016.
[4] Carole-Jean Wu et al. Machine learning at facebook: Understanding inference at the edge. In *proc.of HPCA*. IEEE, 2019.
[5] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.
[6] David Snyder et al. X-vectors: Robust dnn embeddings for speaker recognition. In *proc. of ICASSP*. IEEE, 2018.
[7] Vivienne Sze et al. Efficient processing of deep neural networks: A tutorial and survey. In *Proc.of the IEEE*, 2017.
[8] H. Kwon et al. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. In *Proc. of ASPLOS*, 2018.
[9] Mingyu Gao et al. Tangram: Optimized coarse-grained dataflow for scalable nn accelerators. In *Proc.of the ASPLOS*. ACM, 2019.
[10] Yu-Hsin Chen et al. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE JETCES*, 2019.
[11] Hyoukjun Kwon et al. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proc.of Micro*. ACM, 2019.
[12] Angshuman Parashar et al. Timeloop: A systematic approach to dnn accelerator evaluation. In *Proc.of the ISPASS*. IEEE, 2019.
[13] Xuan Yang et al. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In *Proc.of the ASPLOS*. ACM, 2020.
[14] Chia-Hsin Owen Chen et al. Smart: A single-cycle reconfigurable noc for soc applications. In *proc.of DATE*. IEEE, 2013.
[15] Hao Zheng et al. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *proc.of HPCA*. IEEE, 2021.
[16] Hao Zheng et al. A versatile and flexible chiplet-based system design for heterogeneous manycore architectures. In *Proc.of DAC*. IEEE, 2020.
[17] Eric Qin et al. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *Proc. of HPCA*. IEEE, 2020.
[18] Lizhong Chen et al. Mp3: Minimizing performance penalty for power-gating of clos network-on-chip. In *Proc.of HPCA*. IEEE, 2014.
[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc.of the ICLR*, 2014.
[20] Kaiming He et al. Deep residual learning for image recognition. In *Proc.of the CVPR*. IEEE, 2016.
[21] Olaf Ronneberger et al. U-net: Convolutional networks for biomedical image segmentation. In *Proc.of MICCAI*. Springer, 2015.
[22] Andrzej Jajszczyk. Nonblocking, repackable, and rearrangeable clos networks: fifty years of the theory evolution. In *Communications Magazine*. IEEE, 2003.
[23] Soroush Ghodrati et al. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *proc.of MICRO*. IEEE, 2020.
[24] Chen Sun et al. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proc.of the NOCS*. IEEE, 2012.