# MIREDO: MIP-Driven Resource-Efficient Dataflow Optimization for Computing-in-Memory Accelerator

Xiaolin He[†], Cenlin Duan[‡], Yingjie Qi[†], Xiao Ma[†], Jianlei Yang[†§]
[†] School of Computer Science and Engineering, Beihang University
[‡] School of Integrated Circuit Science and Engineering, Beihang University
[§] Qingdao Research Institute, Beihang University

***Abstract*—Computing-in-Memory (CIM) architectures have emerged as a promising solution for accelerating Deep Neural Networks (DNNs) by mitigating data movement bottlenecks. However, realizing the potential of CIM requires specialized dataflow optimizations, which are challenged by an expansive design space and strict architectural constraints. Existing optimization approaches often fail to fully exploit CIM accelerators, leading to noticeable gaps between theoretical and actual system-level efficiency. To address these limitations, we propose the MIREDO framework, which formulates dataflow optimization as a Mixed-Integer Programming (MIP) problem. MIREDO introduces a hierarchical hardware abstraction coupled with an analytical latency model designed to accurately reflect the complex data transfer behaviors within CIM systems. By jointly modeling workload characteristics, dataflow strategies, and CIM-specific constraints, MIREDO systematically navigates the vast design space to determine the optimal dataflow configurations. Evaluation results demonstrate that MIREDO significantly enhances performance, achieving up to $3.2\times$ improvement across various DNN models and hardware setups.**

***Index Terms*—Computing-in-Memory, Dataflow Optimization, Mixed-Integer Programming, DNN Accelerator.**

## I. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art (SOTA) performance across various domains, establishing themselves as the cornerstone of modern intelligent systems [1], [2]. However, the deployment of DNNs in resource-constrained edge devices faces significant challenges due to substantial computational and memory requirements. To reduce the memory bottlenecks and data transfer overhead inherent in traditional von Neumann architectures, computing-in-memory (CIM) has emerged as a promising paradigm [3]. By performing matrix-vector multiplications (MVMs) directly in-situ within memory arrays, CIM can significantly reduce costly data movement. Prior works [4], [5] have demonstrated the effectiveness of this paradigm, greatly advancing the field of specialized accelerators for DNN inference.

However, the efficient design of CIM accelerators faces unique challenges in dataflow optimization, which involves strategies for data partitioning and scheduling to optimize performance and energy efficiency. In contrast to traditional accelerators, dataflow optimization for CIM accelerators must consider additional factors: inherent row- and column-level parallelism constraints within CIM arrays, the bit-serial execution model, and the stalling introduced by weight reloading procedures [6]. Empirically tuned or manually crafted mapping strategies often fail to fully exploit the potential of CIM architectures, resulting in substantial performance degradation. Moreover, a single layer can have up to 13 million feasible dataflow options with drastically different performance characteristics [7], making exhaustive search computationally infeasible.

These optimization issues often lead to a commonly observed gap between the theoretical performance of CIM macros and actual system-level performance, which primarily stems from mismatches between dataflow strategies and hardware architecture. The challenge becomes more complex when considering the diversity of DNN workloads, each with distinct computational patterns and memory access requirements. Furthermore, the memory hierarchy external to the CIM macro introduces system-level bottlenecks that are heightened by limited on-chip resources [8]. Therefore, optimizing the entire processing system, rather than individual macro-level design, is essential for realizing the full potential of CIM accelerators [9], [10]. However, existing research offers limited insight into maximizing the system-level performance of CIM, particularly within resource-constrained accelerators. To bridge this gap, we introduce MIREDO, a dataflow optimization framework for SRAM-based CIM accelerators. MIREDO employs Mixed-Integer Programming (MIP) for optimization, with a focus on data transfer efficiency.

The main contributions of this work are summarized as follows:

- We present a hierarchical abstraction of CIM architectures and a corresponding data transfer analysis that reveals system-level performance bottlenecks.
- We formulate the dataflow optimization as an MIP problem, which leverages a novel analytical latency model to guide the search for high-efficiency solutions.
- We demonstrate that MIREDO achieves up to $3.2\times$ Energy-Delay Product (EDP) reduction across diverse workloads and hardware configurations.

## II. Background and Motivations

This section provides an overview of CIM architecture and reviews the landscape of related work in dataflow design and optimization.

This work is supported in part by the National Natural Science Foundation of China (Grant No. 62572036), the Beijing Natural Science Foundation (Grant No. L243031), the National Key R&D Program of China (Grant No. 2023YFB4503704 and 2024YFB4505601).

Corresponding author is Jianlei Yang. Email: jianlei@buaa.edu.cn.

TABLE I: Comparison of works on different criteria.

| Work | System Optimization | CIM Modeling | CIM-aware Optimization |
|---|---|---|---|
| NeuroSpector [7] / CoSA [11] | ✓ | – | – |
| SPCIM [12] | – | ✓ | ✓ |
| ZigZag-IMC [13]/CiMLoop [14] | ✓ | ✓ | – |
| **MIREDO** | ✓ | ✓ | ✓ |

## A. *CIM Architecture*

CIM architectures represent a paradigm shift from conventional accelerator designs by performing computation directly within memory arrays, which mitigates the data-transfer bottleneck, thus significantly enhancing inference efficiency. Previous studies [15]–[17] have shown that various technologies, such as RRAM, SRAM, and MRAM, are viable candidates for CIM accelerators. Among these technologies, SRAM is particularly attractive due to its lower latency, higher endurance, and compatibility with advanced CMOS logic processes [18]–[20]. While SRAM-based CIM architectures promise substantial performance gains for DNN inference, the inherent 6T cross-coupled structure of SRAM results in much lower integration density and capacity than other technologies. As the scale of DNN models increases dramatically, multi-core CIM architectures have been widely explored to reduce the data transfer overhead and increase computation parallelism. Although this approach improves execution efficiency, it introduces a series of system-level design complexities in memory mapping, data scheduling, and inter-core communication. Therefore, developing a dataflow optimization strategy tailored for these multi-core systems has become critical to unleash their full potential.

## B. *Dataflow Strategies*

Most existing CIM architectures leverage a weight-stationary (WS) dataflow, as it aligns well with in-situ computation principles [21]. However, relying on such a static dataflow can lead to poor performance due to the aforementioned challenges. This highlights the need for adaptive optimization methods capable of generating dataflows tailored to diverse hardware configurations and the heterogeneity across different DNN layers. A summary of several recent works addressing this problem is presented in Table I.

While many dataflow optimization methods exist for traditional Processing Element (PE) accelerators, they fall short of leveraging the efficiency benefits of CIM. For instance, NeuroSpector [7] is an analysis framework that highlights the impact of data movement on performance by modeling the memory hierarchy. CoSA [11] employs MIP to model and optimize factors such as buffer utilization and communication traffic, a technique also employed in this paper. The fundamental architectural differences between PE and CIM accelerators limit the effective application of these existing methods to CIM-based systems.

Among CIM-focused approaches, SPCIM [12] proposes a novel approach by introducing a reconfigurable cluster topology within CIM macro structures, which enables harnessing input parallelism or weight parallelism adaptively. However, the effectiveness of these optimizations is tightly coupled to a specific hardware topology, making it challenging to adapt efficiently to novel workloads and different hardware platforms.

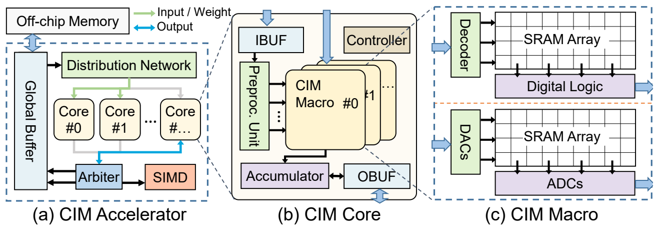

Fig. 1: Hierarchical abstraction of the oriented CIM accelerator.

ZigZag-IMC [13] and CiMLoop [14] provide comprehensive modeling and evaluation frameworks, effectively integrated into existing system-level infrastructures. These frameworks utilize random or heuristic search methods to find optimized dataflow configurations.

Although the aforementioned studies contribute valuable insights to specific aspects of CIM optimization, they exhibit the following limitations:

❶ **Oversimplified Performance Modeling:** Most existing works rely on simplistic performance models that assume data transfer latencies can be perfectly hidden through double-buffering [22], [23]. These idealistic assumptions fail to capture the complex interactions within CIM architectures, leading to temporal underutilization of hardware resources and suboptimal system-level performance [24].

❷ **Inefficient Design Space Exploration:** Manual and heuristic methods depend heavily on predefined rules or expert knowledge, which lack scalability and cannot guarantee optimal solutions. Meanwhile, brute-force search becomes computationally prohibitive and is typically restricted to narrow, predefined search spaces.

To address these limitations, this work introduces MIREDO, an optimization framework that formulates dataflow optimization as an MIP problem. Based on a comprehensive analysis of CIM architectural characteristics and data transfer behaviors, our model incorporates resource constraints to accurately predict inference latency. This approach enables the systematic generation of high-performance dataflows for complex, multi-core CIM accelerators.

## III. Architecture Abstraction

To support the proposed MIREDO framework, we introduce a multi-core CIM architecture abstraction, characterizing its inherent structural parallelism and unique dataflow behaviors.

## A. *Hardware Abstraction*

We illustrate the top-level accelerator architecture in Fig. 1(a), comprising a global buffer, a distribution network, multiple CIM cores, and a Single-Instruction Multiple-Data (SIMD) unit. The distribution network efficiently multicasts weights and features to the CIM cores for MVM operations. The SIMD unit is then responsible for post-processing, such as executing activation and pooling functions, and accumulating partial sums generated by individual cores.

The CIM core architecture shown in Fig. 1(b) features a configurable memory hierarchy of buffers and register files, which offers bypassable access to reduce latency and a double-buffered mode to overlap computation with communication. However, this mode comes at the cost of halving the effective

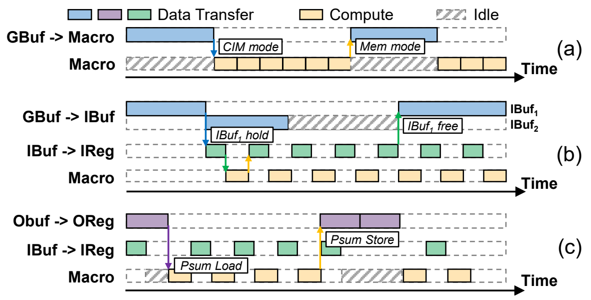

Fig. 2: Representative data-transfer timelines illustrating (a) mode-switch stalls of the CIM macro, (b) pipeline stall due to throughput mismatch, and (c) operand-synchronization stalls.

storage capacity compared to a single-buffered configuration. In addition, the core datapath involves several non-bypassable processing stages, including a preprocessing unit for bit-serial input and an accumulator for partial sum aggregation.

As depicted in Figure 1(c), the CIM macro stores weight data within its memory array. Input vectors are broadcast to the SRAM array, typically along the wordlines or bitlines, to perform MVM operations with weight data in either the analog or digital domain. CIM macros generally operate in two distinct modes: a standard *Memory Mode* for read/write or weight update and a *Compute Mode* for MVM execution. However, a critical limitation arises because both modes share peripheral circuits. This structural dependency prevents a conventional CIM array from performing computation and weight updates concurrently. Consequently, the resulting pipeline stalls create a severe performance bottleneck, an issue particularly acute for workloads that necessitate frequent weight reloading, as shown in Fig. 2(a). While custom macro designs can circumvent this limitation, they often sacrifice versatility and introduce significant overhead in area, power, and design complexity.

### B. Data Transfer Analysis

Previous analytical models for latency often oversimplify data transfer by assuming that latency equals the maximum access time across all memory hierarchy levels. However, this approach overlooks the complex interactions between operations executing within the accelerator. Even with double-buffering techniques, throughput mismatches between memory hierarchy levels can create pipeline stalls. As illustrated in Fig. 2(b), the data transfer from the global buffer (GBuf) to the input buffer (IBuf) may be faster than the downstream processing at the input registers (IReg). Consequently, once IBuf$_2$ is also filled, the data transfer from the GBuf to the IBuf is forced into an idle state, awaiting IBuf$_1$ to become free. This resulting pipeline stall significantly reduces the effective bandwidth of the datapath.

While the weight-stationary dataflow adopted by most CIM accelerators focuses on reducing weight data movement overhead, the transfer overhead of feature maps remains a significant factor in overall system performance. As shown in Fig. 2(c), partial sum (Psum) write-back operations cannot overlap with computation when the output register (OReg) employs single buffering, thereby lengthening the critical path. Furthermore, strict operand synchronization requirements create additional bottlenecks, where the delayed arrival of any operand stalls the entire pipeline, resulting in temporal underutilization of hardware resources.

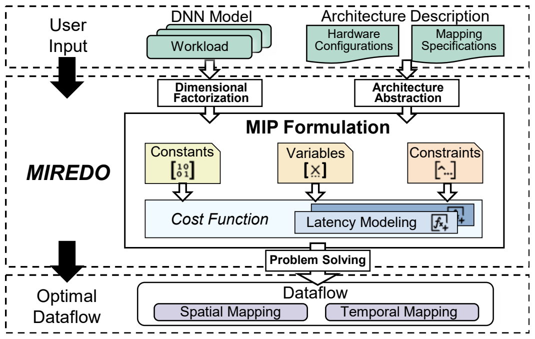

Fig. 3: Overview of the proposed MIREDO framework.

## IV. MIREDO Framework

This section introduces MIREDO, an MIP-based framework designed for dataflow optimization in CIM accelerators, building on the architecture abstraction outlined in Section III.

### A. Framework Overview

Dataflow optimization for DNN accelerators involves deriving optimal tiling factors and loop permutations for a given DNN workload, which can be represented via a loop nest format. To automate this complex process, we introduce MIREDO, a systematic framework that provides an adaptive approach to navigate the vast mapping space. The framework, shown in Fig. 3, accepts a DNN model in ONNX format and a detailed architecture description as its primary inputs, producing a complete dataflow that explicitly defines the optimal temporal and spatial mapping.

The core of MIREDO is the mathematical formulation of the above optimization problem. The framework encodes the workload and hardware parameters as a set of constants $S_c$, while the dataflow mapping space is treated as a set of decision variables $S_v$. A body of constraints links these constants and variables, restricting the search space by enforcing hardware limits and dataflow mapping specifications. This structured representation constitutes an MIP problem in which the objective is to minimize a cost function $obj_{\text{cost}}$, encompassing not only overall latency but also memory access overhead. Table II summarizes the notations used in our model. To enable an accurate cost evaluation, we further propose a novel analytical latency model that accurately accounts for pipeline stalls in Section IV-D. By solving the MIP problem formulated below:

$$\min_{S_v} obj_{\text{cost}}(S_c, S_v), \tag{1}$$

MIREDO produces a systematically optimized and highly efficient dataflow for the target CIM accelerator.

### B. Constant Definitions

The constants in our model are derived from the hardware architecture and the workload. A memory mapping matrix $C^M$ indicates whether each memory unit is dedicated to a single operand or shared among several operands. Furthermore, a key architectural characteristic for dataflow optimization is the available parallelism, which greatly influences computational efficiency and is implemented through spatial unrolling. This

TABLE II: MIREDO notation declarations.

| Index | Description |
|---|---|
| $d$ | Index for DNN tensor dimension. |
| $f$ | Index for loop tiling factor. |
| $i$ | Index for temporal loop nest level. |
| $m$ | Index for memory hierarchy level. |
| $u$ | Index for spatial unrolling axis. |
| $\lambda$ | Index for operand type (input / weight / output) |
| **Constant** | **Description** |
| $F$ | Sets of tiling factors for each dimension. |
| $C$ | Sets of constants defining the mapping specifications. |
| $H, Y$ | Pre-enumerated sets for data sizes and loop bounds. |
| $\mu$ | Weighting parameters for functions. |
| **Variable** | **Description** |
| $X$ | Binary matrices representing spatial-temporal mapping. |
| $V$ | One-hot encoded vectors for linear selection. |
| $\psi$ | Binary indicators for operational states. |
| $T$ | Transfer latency of the operands. |
| $P$ | Processing latency of the operands. |
| $L$ | Critical path latency at a loop level. |

is captured by a binary matrix $C^X$ specifying which tensor dimensions can be unrolled onto these axes. For instance, due to inherent hardware constraints, the CIM macro's wordline axis might only permit spatial unrolling at the output channel. These formulations and other constants, such as memory bus width ($\mathcal{BW}$) and capacity ($\mathcal{CA}$), collectively define the rigid hardware resource constraints upon which dataflow optimization is performed.

DNN operators are typically modeled as loop nests, with bounds defined by dimensional parameters of the weights and feature maps. While prior work often decomposes these bounds into prime factors for allocation, this approach can lead to a combinatorial explosion in the search space due to the large number of factors $F$. To mitigate this complexity, we propose *Flexible Factorization*: an algorithm that employs a greedy strategy to reduce the number of factors, as detailed in Alg. 1.

The algorithm starts with a complete set of prime factors and iteratively merges pairs of factors. This process is controlled by two user-definable parameters: a minimum factor count ($k_{\min}$) and a relative loss threshold ($\alpha$). At each step, the algorithm chooses to merge the factor pair that incurs the minimum loss to the flexibility score, which is calculated by *FlexScore* in Alg. 1. The *FlexScore* is a heuristic metric designed to quantify the mapping versatility of the factor set. Specifically, it enumerates all unique partitions of $F$ into $k$ disjoint subsets, where $k \in \{1, 2, 3\}$. The score is computed as a weighted sum of the number of unique partitions, utilizing decreasing positive weights ($\mu^P$) to prioritize factor sets that offer greater partitioning flexibility. Merging factors diminishes this flexibility, consequently resulting in a lower score. The merging process stops when the factor count reaches $k_{\min}$, or when the relative loss from the best available merge exceeds the threshold $\alpha$. This algorithm effectively reduces the search space complexity, providing a high-quality set of factors for the subsequent MIP optimization.

**Algorithm 1:** Flexible Factorization

```
Input: An integer $N > 1$, a minimum factor count $k_{\min}$, a relative
       loss threshold $\alpha \in [0, 1]$ ;
Output: A factor list $F$ where $\prod F = N$.
1  Function FlexibleFactorization($N, \alpha, k_{\min}$):
2      $F \leftarrow$ PrimeFactors($N$);
3      if $length(F) \le k_{\min}$ then
4          return $F$;
5      $Score_{\text{full}} \leftarrow$ FlexScore($F$);
6      while $length(F) > k_{\min}$ do
7          $Score_{\text{base}} \leftarrow$ FlexScore($F$);
8          $(\Delta_{\text{best}}, F_{\text{best}}) \leftarrow (\infty, \text{null})$;
9          for each pair $(f_i, f_j)$ in $F$ do
10             $F_{\text{merged}} \leftarrow (F \setminus \{f_i, f_j\}) \cup \{f_i \cdot f_j\}$;
11             $Score_{\text{merged}} \leftarrow$ FlexScore($F_{\text{merged}}$);
12             $\Delta \leftarrow (Score_{\text{base}} - Score_{\text{merged}})/Score_{\text{full}}$;
13             if $\Delta < \Delta_{best}$ then
14                 $\Delta_{\text{best}} \leftarrow \Delta, F_{\text{best}} \leftarrow F_{\text{merged}}$;
15         if $\Delta_{best} > \alpha$ then
16             break
17         $F \leftarrow F_{\text{best}}$;
18     return $F$;
19 Function FlexScore($F_{input}$):
20     $P_1, P_2, P_3 \leftarrow \varnothing, \varnothing, \varnothing$;
21     $P_{\text{all}} \leftarrow$ All partitions of $F_{\text{input}}$ into $k \in \{1, 2, 3\}$ subsets;
22     for each partition $\pi$ in $P_{all}$ do
23         $k \leftarrow |\pi|$;
24         $prods \leftarrow$ sorted tuple of products from subsets of $\pi$;
25         Add prods to $P_k$;
26     return $\mu_1^P \cdot |P_1| + \mu_2^P \cdot |P_2| + \mu_3^P \cdot |P_3|$;
```

## C. Variables and Constraints

The formulation of the dataflow mapping space can be represented by a set of binary decision variables. The fundamental mapping decision for each factor $F_{d,f}$ is its placement in either the temporal loop nest or a spatial unrolling dimension. This binary choice is captured by variables $X^L_{d,f,i}$ and $X^U_{d,f,u}$, respectively, governed by a uniqueness constraint:

$$\sum_i X^L_{d,f,i} + \sum_u X^U_{d,f,u} = 1 \quad \forall d, f, \tag{2}$$

that ensures each factor is mapped exactly once.

To support flexible mapping strategies such as uneven mapping [25], we introduce a granular variable $X^M_{d,f,\lambda,m}$ that enables independent definition of loop blocks for each operand type ($\lambda$) at each memory level ($m$). Loop blocks represent groups of loops assigned to the same memory level. These variables are subject to the following constraints:

$$\begin{matrix} X^M_{d,f,\lambda,m} \le C^M_{m,\lambda},\ X^U_{d,f,u} \le C^X_{u,d}, \\ X^Z_{i,\lambda,m} \ge X^M_{d,f,\lambda,m} \wedge X^L_{d,f,i} \end{matrix} \quad \forall d, f, \lambda, u, m, i. \tag{3}$$

The variable $X^Z$ indicates whether memory level $m$ is mapped to loop $i$. For the formulation of loop permutation, the number of available temporal loop slots is set equal to the total number of loop factors, with the indicator $\psi^L_i$ identifying which of these slots are active. Similarly, the indicator $\psi^U_{m,\lambda}$ specifies whether a memory level is utilized or bypassed for a particular operand.

$$\psi^L_i = \sum_{d,f} X^L_{d,f,i},\ \psi^U_{m,\lambda} = \bigvee_{d,f} X^M_{d,f,\lambda,m} \quad \forall i, \lambda, m. \tag{4}$$

To formally define the data transfer path, we introduce $X^N$ as a binary variable indicating a direct transfer for operand $\lambda$ from $m$ to $m'$. The following constraints are introduced to link $X^N$ with the utilization status $\psi^U$, ensuring path uniqueness and preventing illegal bypasses:

$$\sum_{m' \ge m+1} X^N_{m,m',\lambda} = \psi^U_{m,\lambda},\ 1 - \psi^U_{m_1,\lambda} + X^N_{m,m_1,\lambda} \ge X^N_{m,m_2,\lambda}. \tag{5}$$

Here, $m$, $m_1$, and $m_2$ are memory level indices satisfying $m < m_1 < m_2$, where a larger index value $m$ denotes a memory level closer to the CIM macros.

To enforce hardware capacity constraints, the data size of each operand at every memory level must be precisely formulated. The dimensional bound at a specific level is determined by multiplying all loop factors mapped to the current level and all levels below. Since this multiplicative calculation is inherently non-linear, it is unsuitable for the MIP formulation. The loop bound for each dimension ($B^S_{m,\lambda,d}$) is therefore calculated as a linear sum in the logarithmic domain:

$$B^S_{m,\lambda,d} = \sum_f \log(F_{d,f}) \cdot \Big( \sum_{m'=m} X^M_{d,f,\lambda,m'} + \sum_u X^U_{d,f,u} \Big), \tag{6}$$

where the summation over $u$ is performed for all indices satisfying the condition $C_u \geq m$. Inspired by prior work [23], we circumvent the non-linear product by enumerating all valid data sizes and using a one-hot encoded vector ($V^S_{m,\lambda}$) to select a single candidate from the set $H_{m,\lambda}$. Let $\mathcal{P}$ denote the operand precision, the data size is given by

$$Size_{m,\lambda} = H_{m,\lambda} \cdot V^S_{m,\lambda} \cdot \psi^U_{m,\lambda} \cdot \mathcal{P}_{m,\lambda} \quad \forall m, \lambda. \tag{7}$$

A pre-calculated dimension vector $Y_{m,\lambda,d}$ holds the corresponding bound for each enumerated data size. The constraint (6) ensures that the loop bound derived from the mapping variables is consistent with the specific entry selected by the one-hot vector:

$$\log(Y_{m,\lambda,d}) \cdot V^S_{m,\lambda} = B^S_{m,\lambda,d} \quad \forall m, \lambda, d\,. \tag{8}$$

The bit-serial and highly parallel nature of the CIM paradigm imposes unique demands on data transfer, necessitating flexible buffering strategies. Each memory level can operate in a standard single-buffered mode, which maximizes storage capacity but risks pipeline stalls due to mutually exclusive access. Alternatively, double-buffering can be employed to overlap data transfer with computation, at the expense of halving the effective storage capacity. To allow the dataflow optimizer to navigate this trade-off, the binary variable $\psi^{DM}$ is introduced to select the optimal mode, enabling the capacity constraint to be formally expressed as follows:

$$\sum_\lambda (1 + \psi^{DM}_{m,\lambda}) \cdot Size_{m,\lambda} \leq \mathcal{CA}_m \quad \forall m. \tag{9}$$

The size of the data tile transferred into a memory level is distinct from the data stored within it, due to data reuse and multicast opportunities. We introduce a new dimensional bound ($B^T$) by modifying the formulation (6) to exclude temporal loop factors mapped at the current memory level:

$$B^T_{m,\lambda,d} = \sum_f \log(F_{d,f}) \cdot \Big( \sum_{m' \geq m+1} X^M_{d,f,\lambda,m'} + \sum_u X^U_{d,f,u} \Big). \tag{10}$$

A corresponding one-hot vector $V^T_{m,\lambda}$ links this new bound to the same constant $Y_{m,\lambda,d}$ via an analogous constraint.

### D. Performance Modeling

The total execution latency of a workload is determined by our loop-based analytical model, which recursively calculates the operand processing latency ($P_{i,\lambda}$) at each temporal loop level $i$. Using the previously defined symbols, the cycle count for a transfer operation ($T_{i,\lambda}$) is defined by the following constraint:

TABLE III: Classification of operand-processing latency for different buffering strategies and operand types (I: Input Feature, W: Weight, O: Output Feature).

| **Buf. Strategy** | **Operand** | **Latency expression** ($P_{i,\lambda}$) |
|---|---|---|
| Single | I / W | $L_i \cdot (N_i - 2) + 2 * T_{i,\lambda} + P_{i+1,\lambda}$ |
| Single | O | $L_i \cdot (N_i - 1) + 2 * T_{i,\lambda} + P_{i+1,\lambda}$ |
| Double | I / W | $\max\{L_i \cdot (N_i - 3) + 2 * T_{i,\lambda} + \max\{T_{i,\lambda}, P_{i+1,\lambda}\}, T_{i,\lambda} \cdot N_i\}$ |
| Double | O | $L_i \cdot (N_i - 2) + T_{i,\lambda} + \max(T_{i,\lambda}, L_i) + \max(T_{i,\lambda}, P_{i+1,\lambda})$ |
| (No Transfer) | (Any) | $L_i \cdot (N_i - 1) + P_{i+1,\lambda}$ |

$$X^Z_{i,\lambda,m} = 1 \rightarrow T_{i,\lambda} \cdot \mathcal{BW}_m = H_{m,\lambda} \cdot V^T_{m,\lambda} \cdot \psi^U_{m,\lambda} \cdot \mathcal{P}_{m,\lambda}. \tag{11}$$

The detailed formulations under all possible scenarios are listed in Table III.

The critical path latency $L_i$ is formulated as the maximum of two primary components: 1) the cumulative latency of its nested inner loop, calculated as the product of its critical path $L_{i+1}$ and the loop count $N_{i+1}$; and 2) the combined latency of transfer and processing from operands at the current loop. Since the product term is non-linear, MIREDO employs a one-hot vector $V^F$ to select a pre-enumerated bound, ensuring MIP compatibility. This combination depends on the buffering strategy, captured by the binary indicator $\psi^{DL}_{i,\lambda}$:

$$\psi^{DL}_{i,\lambda} = \bigvee_{m,m' \geq m+1} (X^Z_{i,\lambda,m} \wedge X^N_{m,m',\lambda} \wedge \psi^{DM}_{m,\lambda}). \tag{12}$$

This allows the model to select either a sequential summation ($T_{i,\lambda} + P_{i+1,\lambda}$) or an overlapped calculation ($\max(T_{i,\lambda}, P_{i+1,\lambda})$) when double-buffering is enabled ($\psi^{DL}_{i,\lambda} = 1$).

Operand-transfer latency is accounted for only in the innermost loop block of each memory level where the actual data transfer occurs. The model also incorporates the data-stationary scenario, which applies when an operand is reused at loop level $i$ and thus incurs no transfer latency.

For the innermost MVM operation within a nested loop, the latency is a constant ($L_{\text{MVM}}$) determined by the operand precision and the CIM macro design. The recursion boundary condition is therefore expressed as $L_{i_{max}+1} = P_{i_{max}+1,\lambda} = L_{\text{MVM}}$. To maintain model consistency, inactive temporal slots propagate latency from their inner loop, as expressed by the relation:

$$\psi^L_i = 0 \rightarrow (L_i, P_{i,\lambda}) = (L_{i+1}, P_{i+1,\lambda}) \quad \forall i, \lambda. \tag{13}$$

Based on the recursive components, the final optimization objective is formulated as:

$$obj_{\text{cost}} = \mu^C_1 \cdot \max_\lambda (P_{0,\lambda}) - \mu^C_2 \cdot \sum_{m,\lambda} m * Size_{m,\lambda}. \tag{14}$$

This cost function balances two competing goals via weighting parameters $\mu^C$: minimizing total latency and maximizing data locality. The latter goal incentivizes storing data at lower memory hierarchy levels closer to the CIM macros to reduce data movement overhead. Minimizing this objective function yields a dataflow optimized for both execution performance and resource efficiency.

## V. Evaluation Results

### A. Experiment Setup

To ensure a fair and accurate evaluation of MIREDO, we developed a custom simulator based on the architecture from

TABLE IV: Hardware component configurations.

| Component | Configuration | |
|---|---|---|
| CIM Macro | Array | 128 × 32 |
| Local Buffer | Capacity | 256 KB |
| | Bus Width | 128 bit |
| Core | Number | 8 |
| Global Buffer | Capacity | 8 KB |
| | Bus Width | 256 bit |
| Off-chip Memory | Bus Width | 64 bit |

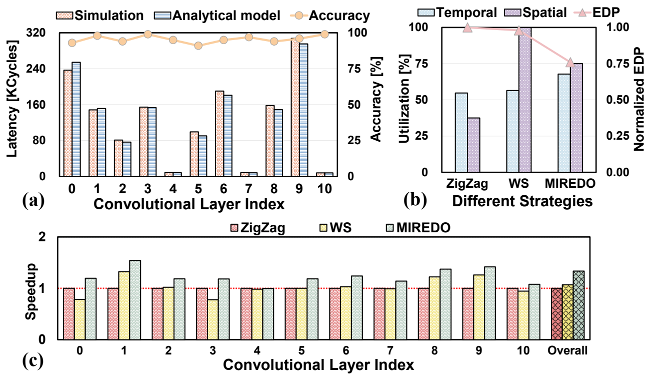

Fig. 4: MIREDO performance evaluation. (a) Analytical model accuracy validation. (b) Utilization and EDP comparison. (c) Per-layer and overall speedup comparison.

Section III. Latency and power of memory are modeled using PCACTI [26], and the statistics of other logic components are derived from prior works [13]. The detailed configurations of these hardware components are summarized in Table IV. We employ Gurobi [27], a general-purpose optimizer for MIP and other constrained programming, as the solver. By automatically identifying convolutional layers from the DNN model, MIREDO defines the constants, variables, constraints, and objective functions before invoking the solver. We equip a 2GHz Intel Xeon Gold 6330 CPU, with the time for solving a single layer capped at 5 minutes.

To evaluate MIREDO, we benchmark its performance against two representative dataflow schemes: 1) A heuristic search inspired by the ZigZag framework [13] for its support of uneven mappings; 2) A conventional Weight-Stationary (WS) dataflow derived by imposing additional constraints within our own MIP formulation. This setup provides a comparison against both an empirical strategy and a heuristic optimization approach. Our baseline workload utilizes ResNet-18 inference on ImageNet [28], with weight and activation quantized to INT8 format. Subsequently, we extended our analysis to encompass various networks and architectures.

## B. Performance Evaluation

**Accuracy.** The accuracy of our analytical model was validated by comparing its latency predictions against results from a detailed hardware simulation. As illustrated in Fig. 4(a), the model achieves an average accuracy of 95.5% across all evaluated model layers.

**Latency.** We further compare the latency of different dataflow strategies per layer, as depicted in Fig. 4(c). The heuristic scheme from ZigZag and the constrained weight-stationary dataflow exhibit comparable overall latency, yet

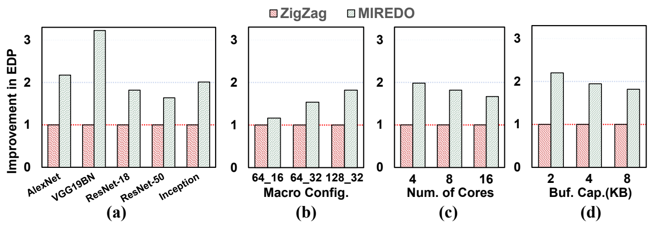

Fig. 5: Performance comparison across various DNN models and hardware configurations.

their effectiveness varies across different layers. In contrast, MIREDO achieves a consistent speedup across most layers.

**Efficiency.** Fig. 4(b) illustrates the trade-off between macro utilization and overall system efficiency in dataflow optimization for a representative layer. Compared to a baseline with limited optimization, the weight-stationary method maximizes spatial utilization to improve weight reuse, but restricts the search space and leads to a suboptimal solution. By jointly optimizing latency and resource allocation, MIREDO explores a considerably larger mapping space and consequently improves system-level performance.

## C. Adaptability and Robustness

We conduct extensive evaluations to validate the adaptability and robustness of MIREDO across diverse workloads and hardware configurations. Our evaluation first assesses MIREDO's adaptability against the ZigZag-based heuristic across representative DNN models. As shown in Fig. 5(a), our approach consistently achieves significant EDP reductions, ranging from 1.6× to 3.2× over the baseline. These results demonstrate MIREDO's ability to generate optimized dataflows tailored to different workloads.

To assess robustness, we analyze performance across various hardware configurations, including different macro configurations, core counts, and buffer capacities. As illustrated in figures 5(b)-(d), MIREDO achieves significant improvements in EDP reduction compared to the baseline across various hardware configurations. This highlights the value of systematic optimization in memory-constrained environments, where conventional approaches struggle to balance efficiency with severe memory limits.

## VI. Conclusions

In this work, we introduce MIREDO, a novel framework utilizing MIP to optimize dataflows for executing DNN workloads on CIM accelerators. Through a hierarchical hardware abstraction, MIREDO systematically models the complex mapping processes and hardware constraints, allowing accurate estimation of execution latency. This structured approach enables MIREDO to efficiently determine the optimal dataflow configuration in a single iteration, significantly improving CIM accelerator efficiency. Experimental results show that MIREDO effectively reduces the EDP by up to 3.2×, highlighting its adaptability to diverse workloads and hardware constraints. Moreover, the flexible MIP formulation ensures that MIREDO is highly scalable and easily extendable to various DNN workloads and CIM architectures.

## REFERENCES

[1] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar *et al.*, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2016.

[4] P. Houshmand, G. M. Sarda, V. Jain, K. Ueyoshi, I. A. Papistas *et al.*, "DIANA: An End-to-End Hybrid DIgital and ANAlog Neural Network SoC for the Edge," *Journal of Solid-State Circuits (JSSC)*, vol. 58, no. 1, pp. 203–215, 2022.

[5] M. Chang, S. D. Spetalnick, B. Crafton, W.-S. Khwa, Y.-D. Chih *et al.*, "A 40nm 60.64TOPS/W ECC-Capable Compute-in-Memory/Digital 2.25MB/768KB RRAM/SRAM System with Embedded Cortex M3 Microprocessor for Edge Recommendation Systems," in *Proceedings of International Solid-State Circuits Conference (ISSCC)*, 2022.

[6] Z. Yang, K. Liu, Y. Duan, M. Fan, Q. Zhang *et al.*, "Three Challenges in ReRAM-Based Process-In-Memory for Neural Network," in *Proceedings of Artificial Intelligence Circuits and Systems (AICAS)*, 2023.

[7] C. Park, B. Kim, S. Ryu, and W. J. Song, "NeuroSpector: Systematic Optimization of Dataflow Scheduling in DNN Accelerators," *Transactions on Parallel and Distributed Systems (TPDS)*, vol. 34, no. 8, pp. 2279–2294, 2023.

[8] W. Sun, J. Yue, Y. He, Z. Huang, J. Wang *et al.*, "A Survey of Computing-in-Memory Processor: From Circuit to Application," *Open Journal of the Solid-State Circuits Society (OJ-SSCS)*, vol. 4, pp. 25–42, 2023.

[9] Y. Qi, J. Yang, Y. Wang, Y. Wang, D. Wang *et al.*, "Cimflow: An integrated framework for systematic design and evaluation of digital cim architectures," in *Proceedings of Design Automation Conference (DAC)*, 2025.

[10] P. Houshmand, S. Cosemans, L. Mei, I. Papistas, D. Bhattacharjee *et al.*, "Opportunities and Limitations of Emerging Analog in-Memory Compute DNN Architectures," in *Proceedings of International Electron Devices Meeting (IEDM)*, 2020.

[11] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah *et al.*, "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2021.

[12] Y. Wang, F. Tu, L. Liu, S. Wei, Y. Xie *et al.*, "SPCIM: Sparsity-Balanced Practical CIM Accelerator With Optimized Spatial-Temporal Multi-Macro Utilization," *Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 70, no. 1, pp. 214–227, 2022.

[13] J. Sun, P. Houshmand, and M. Verhelst, "Analog or Digital In-Memory Computing? Benchmarking Through Quantitative Modeling," in *Proceedings of International Conference on Computer Aided Design (ICCAD)*, 2023.

[14] T. Andrulis, J. S. Emer, and V. Sze, "CiMLoop: A Flexible, Accurate, and Fast Compute-In-Memory Modeling Tool," in *Proceedings of International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2024.

[15] F.-H. Meng, Y. Wu, Z. Zhang, and W. D. Lu, "TT-CIM: Tensor Train Decomposition for Neural Network in RRAM-Based Compute-in-Memory Systems," *Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 71, no. 3, pp. 1172–1183, 2024.

[16] C. Duan, J. Yang, X. He, Y. Qi, Y. Wang *et al.*, "DDC-PIM: Efficient Algorithm/Architecture Co-Design for Doubling Data Capacity of SRAM-Based Processing-in-Memory," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 3, pp. 906–918, 2024.

[17] X. Wang, J. Yang, Y. Zhao, Y. Qi, M. Liu *et al.*, "TCIM: Triangle Counting Acceleration With Processing-In-MRAM Architecture," in *Proceedings of Design Automation Conference (DAC)*, 2020.

[18] C. Duan, J. Yang, Y. Wang, Y. Wang, Y. Qi *et al.*, "Towards efficient sram-pim architecture design by exploiting unstructured bit-level sparsity," in *Proceedings of Design Automation Conference (DAC)*, 2024.

[19] J.-W. Su, X. Si, Y.-C. Chou, T.-W. Chang, W.-H. Huang *et al.*, "Two-way transpose multibit 6t sram computing-in-memory macro for inference-training ai edge chips," *Journal of Solid-State Circuits (JSSC)*, vol. 57, no. 2, pp. 609–624, 2021.

[20] C. Duan, J. Yang, Y. Wang, Y. Wang, Y. Qi *et al.*, "Efficient sram-pim co-design by joint exploration of value-level and bit-level sparsity," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2025, in press.

[21] X. Sun, X. Wang, W. Li, L. Wang, Y. Han *et al.*, "PIMCOMP: A Universal Compilation Framework for Crossbar-based PIM DNN Accelerators," in *Proceedings of Design Automation Conference (DAC)*, 2023.

[22] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "dMazeRunner: Optimizing Convolutions on Dataflow Accelerators," *Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[23] E. Russo, M. Palesi, G. Ascia, D. Patti, S. Monteleone *et al.*, "Memory-Aware DNN Algorithm-Hardware Mapping via Integer Linear Programming," in *Proceedings of International Conference on Computing Frontiers (CF)*, 2023.

[24] L. Mei, H. Liu, T. Wu, H. E. Sumbul, M. Verhelst *et al.*, "A Uniform Latency Model for DNN Accelerators with Diverse Architectures and Dataflows," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

[25] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators," *Transactions on Computers (TC)*, vol. 70, no. 8, pp. 1160–1174, 2021.

[26] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-Scaled FinFET Devices," in *Proceedings of Computer Society Annual Symposium on VLSI (ISVLSI)*, 2014.

[27] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of computer vision and pattern recognition (CVPR)*, 2009.