Efficient Multi-Array Parallel Scheduling for In-Memory Computing

Rongliang Fu[†], Ran Zhang[†], Libo Shen, Wei Xuan, Ning Lin, Junying Huang[*], Bei Yu, and Tsung-Yi Ho *Fellow, IEEE*

***Abstract*—In-memory computing (IMC) for logic functions executes a target function via a series of logic operations supported by peripheral devices. Because these operations are performed on multiple memory rows in lockstep, this paradigm follows a single-instruction multiple-data (SIMD) style. Existing multi-array schedulers mainly focus on reducing copy instruction lines but often overlook array-level parallel scheduling. This paper presents MAPSIM, an efficient multi-array parallel SIMD-IMC scheduler that reduces both copy instructions and execution cycles. To alleviate input congestion, we model input assignment as a bipartite-graph partitioning problem to optimize input distribution across arrays. For multi-array parallel scheduling, we represent the data state of each array in a state-transition graph and assign each state a scheduling potential energy. The scheduler then greedily selects actions that maximize instantaneous energy gain. To accelerate state transitions, we further employ priority-queue filtering and maximum-weight matching to select optimal, feasible actions per cycle. Experimental results on EPFL benchmarks demonstrate the effectiveness and efficiency of our framework. MAPSIM reduces execution cycles by 23.43% and copy instructions by 19.78% over the state-of-the-art, while achieving a $15.02\times$ speedup in runtime. Our code and data are available at https://github.com/Flians/MAPSIM**

***Index Terms*—in-memory computing, SIMD, multi-array scheduling, XOR–majority graph, parallel scheduling**

I. Introduction

THE von Neumann bottleneck has long been recognized as a major limitation to the advancement of computer systems [1], primarily due to the significant energy overhead associated with frequent data transfers between memory and processor. To address this challenge, various alternative computing paradigms have been explored, among which in-memory computing (IMC) architectures have attracted considerable attention. Several IMC-based architectures have been designed for logic computing tasks [2]–[4]. These architectures typically employ a single instruction multiple data (SIMD) execution model, where a single instruction simultaneously controls operations across multiple memory rows, enabling efficient logical computations. However, due to limitations in peripheral circuitry, IMC architectures generally support only a restricted set of logic operations. Consequently, the input Boolean network must first be transformed into a representation compatible with SIMD-IMC architectures, typically through logic synthesis tools. The synthesized Boolean network is then fed into a scheduler, which generates a scheduling sequence S. Each action in S specifies detailed execution parameters, including the target vertex, its location, and the type of the operation, collectively forming the instruction set for SIMD-IMC. In essence, the scheduler translates a Boolean network into an executable instruction sequence optimized for SIMD-IMC.

The research work described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust and was supported in part by the Research Grants Council of Hong Kong SAR (Grant No. CUHK14207523), and in part by the National Natural Science Foundation of China (Grants No. 62302477).

Rongliang Fu, Libo Shen, Bei Yu and Tsung-Yi Ho are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China. E-mail: {rlfu, lbshen24, byu, tyho}@cse.cuhk.edu.hk.

Ran Zhang and Junying Huang are with the State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing 100190, China. E-mail: {zhangran23s, huangjunying}@ict.ac.cn.

Wei Xuan is with the ACCESS – AI Chip Center for Emerging Smart Systems, InnoHK Centers, Hong Kong Science Park and the Hong Kong University of Science and Technology, Hong Kong, China. E-mail: weixuan@ust.hk.

Ning Lin is with the University of Hong Kong, Hong Kong, China. E-mail: linning@hku.hk.

[†] Equal contribution. [*] Corresponding author: huangjunying@ict.ac.cn.

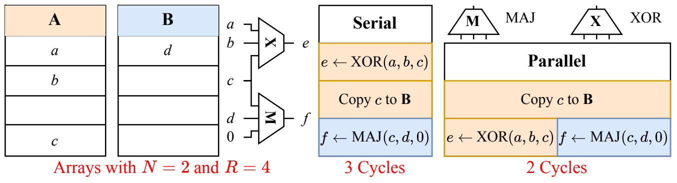

A
B
a
b
c
d
a
b
c
d
0
X
M
e
f
Serial
$e \leftarrow \mathrm{XOR}(a,b,c)$
Copy c to **B**
$f \leftarrow \mathrm{MAJ}(c,d,0)$
M
MAJ
X
XOR
Parallel
Copy c to **B**
$e \leftarrow \mathrm{XOR}(a,b,c)$
$f \leftarrow \mathrm{MAJ}(c,d,0)$
Arrays with $N = 2$ and $R = 4$
3 Cycles
2 Cycles

Fig. 1. Comparison of serial vs. parallel multi-array scheduling sequences.

In recent years, numerous schedulers have been proposed to optimize different performance metrics. These schedulers broadly fall into two categories: single-array schedulers, which focus on scheduling within a single memory array, and multi-array schedulers, which leverage parallelism across multiple arrays. Single-array schedulers remove the characteristics of physical devices and primarily aim to minimize hardware resource usage for implementing logical functions within one array [3], [5]–[7]. While these methods are relatively straightforward and often effective, their abstraction neglects device-specific constraints, potentially leading to resource inefficiencies in practice.

In contrast, multi-array schedulers like MASIM [8] explicitly consider the capacity limits of SIMD-IMC architectures and introduce algorithms for scheduling across multiple arrays. Their optimization objective shifts from minimizing the memory footprint to reducing the number of duplicated instructions during scheduling, a goal more aligned with reducing energy consumption in real applications [3], [9]. MASIM [8] further improves performance by incorporating randomization and multithreading to reduce copy instructions and accelerate scheduling.

Despite the encouraging progress made in these efforts, they still suffer from several limitations. Firstly, although some schedulers employ multi-array scheduling techniques, they do not account for parallelism at the array level. These works emphasize executing one instruction in one array within one cycle, without considering that, in actual situations, different instructions can be executed between arrays in the same cycle. For example, as illustrated in Fig. 1, the initial states of two arrays A and B are depicted on the left, while the Boolean network to be scheduled is shown in the center. By exploiting multi-array parallelism, one execution cycle can be eliminated, thereby improving scheduling efficiency. Secondly, these approaches merely assign the primary inputs (PIs) of the Boolean network in a serial manner, which tends to induce congestion in a single array at the initial stage of scheduling, thereby negatively impacting the overall scheduling process.

This paper proposes MAPSIM, an efficient multi-array parallel scheduler for SIMD-IMC to address the problems outlined above. By partitioning input features into a bipartite graph, MAPSIM effectively alleviates initial array congestion and enhances parallelism. Leveraging a combination of greedy search and maximum-weight matching algorithms, MAPSIM achieves near-optimal scheduling in each SIMD-IMC cycle. Overall, our contributions are as follows:

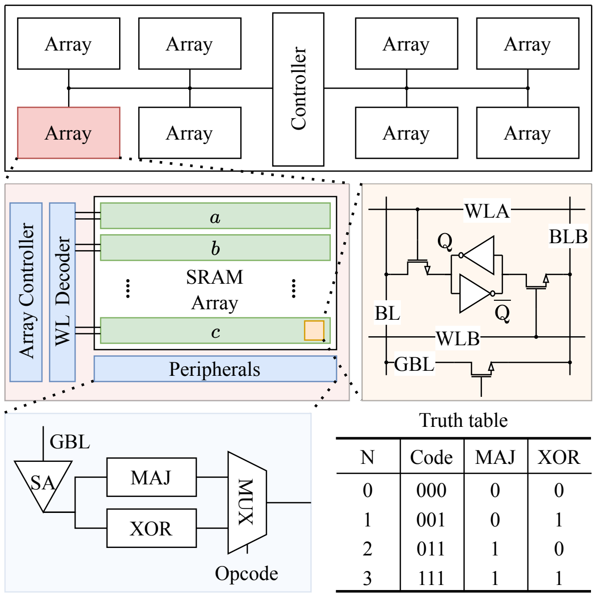
- An efficient multi-array parallel scheduler that reduces the number of copy instructions and execution cycles.
- A novel bipartite graph construction method for input features, which mitigates input congestion and establishes a good initial state for subsequent scheduling.
- A scheduling potential energy calculation method that guarantees transition feasibility for any vertex within the state transition graph.
- An optimal state selection method based on priority queues and maximum-weight matching, significantly reducing the computational overhead of exhaustive state enumeration.
- Experimental results show that MAPSIM reduces 23.43% of execution cycles, 19.78% of the number of copy instructions, and achieves a 15.02× speedup compared to the state-of-the-art scheduler.

The remainder of the paper is organized as follows. Section II presents preliminaries on IMC, scheduling, and related graph algorithms. Related work is discussed in Section III. Section IV formulates the problem and highlights its challenges. Our methodology is detailed in Section V, followed by a comprehensive evaluation of MAPSIM in Section VI. Finally, Section VII concludes the paper.

II. Preliminaries

A. Compute In Cache

Processor caches are built from SRAM arrays and are positioned closest to the CPU core to leverage the principle of data locality. The compute-in-cache paradigm [10]–[13] modifies part of the cache by enabling computation directly within the SRAM fabric itself to receive data-intensive tasks allocated by the CPU, thereby reducing the overhead associated with data movement. A global controller translates CPU instructions into a series of micro-instructions, which are then dispatched to the corresponding cache arrays. To achieve multi-array parallelism, the wordlines (WLs) driver in each array can be replaced by a local controller, primarily implemented using a finite state machine (FSM), that orchestrates the micro-operations required to realize specific macro-instructions. Through state transitions and output logic, FSMs generate cycle-by-cycle control signals to drive the SRAM arrays for computation.



Truth table

N	Code	MAJ	XOR
0	000	0	0
1	001	0	1
2	011	1	0
3	111	1	1

Fig. 2. IMC hardware architecture. (SA: sense amplifier, N: number of '1's, Code: thermometer code of SA output voltage).

This can be achieved through minor modifications to the SRAM's row decoder to allow for simultaneous activation of multiple WLs in a single clock cycle. Furthermore, by employing a Transpose Memory Unit (TMU) to dynamically reorganize the data layout, the design enables efficient row-wise computation with only a 7.5% area overhead [11]. As shown in Fig. 2, the green rows are activated operands. By leveraging physical phenomena like charge sharing on the shared bitlines (BLs) of these activated rows, massively parallel bitwise logical operations (*e.g.*, AND, XOR, Majority) can be executed in-place. A two-level control architecture enables parallel operations across multiple arrays, as shown in Fig. 2. The CPU sends commands to a central global controller, which directs individual array controllers. Equipped with an FSM, voltage generator, latches, and a datapath, these local controllers coordinate with the WL decoder to perform logical computations on specific rows. This design effectively converts the cache into a massively parallel high-throughput SIMD engine with $\sim 10\%$ area overhead [11] to achieve KB-scale data parallelism [10].

By performing computations in caches, this approach drastically reduces data movement, directly mitigating the Von Neumann bottleneck and yielding significant gains in both energy efficiency and performance.

B. Logic in Memory

The logic-in-memory architecture in this work relies on deep microarchitectural customizations of the memory array [3]. We utilize dual-wordline 6T SRAM cells and modify the row decoder to support concurrent multi-row activation. During computation, the two bitlines (BL and BLB) of each cell are shortened together to form a single global bitline (GBL). To efficiently implement complex applications on this SRAM array, we adopt a logic synthesis methodology based on XOR-majority graphs (XMGs) [14]. Compared with traditional And-Inverter graphs (AIGs) [15], XMGs can represent arithmetic-intensive functions with a significantly more compact representation [16]–[18]. The fundamental logic primitives of an XMG are the three-input Majority (MAJ(a, b, c)=$(a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$) and XOR operations, which are realized by a unified circuit in hardware, as shown in Fig. 2.

When the CPU receives instructions related to IMC, it forwards them to the global controller within our SIMD-IMC architecture for coordinated execution. At the beginning of a computation, the GBL is **pre-charged** to VDD. Subsequently, the controller **activates** the wordlines of the three input operands simultaneously. Crucially, based on the inversion flags in the instruction, the controller independently selects for each row whether to activate WLA (to access the true value) or WLB (to access the inverted value). The activation of a wordline turns on the corresponding access transistor, causing the SRAM cell to conditionally **discharge** the GBL according to its internally stored value. Consequently, the final analog voltage on the GBL is determined by the total number of discharging cells, *i.e.*, the effective count of logical '1's in the inputs, thus creating four distinct and distinguishable voltage levels. This analog voltage is captured by a custom multi-level **sense** amplifier and converted by a built-in ADC into a 3-bit "thermometer code" (*e.g.*, 000, 001, 011, 111) [3], [19]. Finally, this code is fed into two configurable logic units, with the output selected by the instruction's opcode: for MAJ instructions, the output is '1' if two or more inputs are '1'; for XOR instructions, the output is '1' if the number of input '1's is odd.

C. Synthesis and Scheduling

Executing an application on our target architecture involves two main stages: synthesis and scheduling. First, a given function is synthesized into a Boolean network represented as a directed acyclic graph (DAG). The vertices of this DAG correspond to PIs, primary outputs (POs), and internal 3-input XOR/MAJ logic operations supported by the hardware. The edges represent the data dependencies between these operations. Next, a scheduler converts this DAG into a hardware-executable instruction sequence. The computation proceeds vertex-by-vertex, governed by the following critical hardware constraints:

- **Memory Integrity:** The initial PIs cannot be overwritten, as they may still be required when computing other functions. However, copies of these PIs stored in other arrays during scheduling can be safely overwritten. Similarly, the computed POs must be preserved in memory once scheduling is completed. In contrast, intermediate vertices can be overwritten as soon as their results are no longer needed for subsequent computations.
- **Operand Locality:** A logic operation (a vertex) can be computed only if all of its fan-ins (FIs) reside within the **same memory array** [20]. The output is then written back to a designated row in that same array, and each vertex can only be computed once.

This operand locality rule creates a central challenge: for large networks that exceed the capacity of a single array, a sophisticated multi-array scheduling strategy is required to manage data placement and inter-array transfers. Developing such a strategy is the core problem addressed in this paper.

Algorithm 1: Weighted Edmonds' Blossom Algorithm.

```
   Input: A weighted graph $G(V, E, w)$.
   Output: Maximum-weight perfect matching $M$.
1  Initialize matching $M \leftarrow \emptyset$;
2  Initialize dual variables $y(v) \leftarrow \max_{(v,u)\in E} \frac{w(v,u)}{2}$;
3  while $\exists$ unmatched vertex $u \in V$ do
4      if No augmenting path and no change in $y$ then
5          break;
6      Grow alternating forest from $u$ using tight edges;
7      if found augmenting path $P$ then
8          $M \leftarrow M \oplus P$ ;   // Augment matching
9      else if found blossom $B$ then
10         Shrink $B$ to super-vertex;
11     else
12         Update $y$;
13 return $M$
```

D. Graph Algorithms

The graph algorithms critical to our multi-array parallel scheduling method, namely maximum weight matching and graph partitioning, will be described in detail below.

Maximum Weight Matching in Graphs: The maximum weight matching is a classical optimization problem defined on a weighted graph $G(V, E, w)$, where V is the set of vertices, E is the set of edges, and $w : E \to \mathbb{R}$ assigns a weight to each edge. A matching is a subset of edges, *i.e.*, $M \subseteq E$, such that no two edges in M share a common vertex in V. The goal is to find a matching M^* that maximizes the sum of edge weights, *i.e.*,

$$M^* = \arg\max \sum_{e \in M} w(e). \tag{1}$$

Depending on the properties of the graph, suitable algorithms can be applied. For weighted bipartite graphs, the Hungarian algorithm [21], [22] can be applied to find the maximum weight matching, but for more general weighted graph structures, the Edmonds' Blossom algorithm [23] is more applicable. The Edmonds' Blossom algorithm operates in polynomial time and guarantees the discovery of an optimal matching. The complete procedural steps are presented in Algorithm 1. Fundamentally, the Edmonds' Blossom algorithm

Algorithm 2: METIS Algorithm.

```
Input: A graph $G(V, E)$, partition number $k$.
Output: Partitions $P$.
1 while $|V|$ is large do
2     $G \leftarrow$ Coarsen graph by contracting vertices;
3 $P \leftarrow$ Compute initial $k$-way partition on coarse graph;
4 while graph is uncoarsened do
5     $P \leftarrow$ Project partition to finer graph;
6     $P \leftarrow$ Refine partition locally;
7 return $P$
```

addresses the dual problem of the original maximum weight matching problem.

In this dual problem, the variable y represents the dual variable associated with each vertex (line 2). For any pair of vertices u and v, the dual variable y is required to satisfy the following complementary *slack* conditions:

$$y(u) + y(v) + \Sigma_{B \supset \{u,v\}} y(B) \geq w(u, v), \forall (u, v) \in E. \quad (2)$$

We usually define $slack(u, v) = y(u) + y(v) - w(u, v) + \Sigma_{B \supset \{u,v\}} y(B)$, and the edges with $slack = 0$ are defined as tight edges, where $y(B)$ indicates the dual variable of a blossom B that is an odd-length alternating cycle. The objective of the dual problem is to minimize the total value of the dual variables, *i.e.*, $\min \left(\sum_{v \in V} y(v)\right)$. To achieve this, the algorithm iteratively executes when an augmenting path is found or the dual variable changes (lines 3-12). Specifically, the algorithm performs a breadth-first traversal starting from unmatched vertices, exploring tight edges to construct an alternating tree, in which the edges along any path alternate between being in M and not in M (line 6). When the expansion reaches a vertex without an incident edge in the current matching M, an augmenting path is found, and the matching is updated by alternating the matched and unmatched edges along this path (lines 7–8). Upon detecting a blossom, the algorithm contracts it into a super-vertex and reinserts it into G (lines 9–10). Otherwise, the alternating tree continues to grow. This iterative procedure ensures consistent progress toward optimality while maintaining feasibility with respect to dual constraints.

When no augmenting path can be found, the algorithm updates the dual variables y by decreasing them while satisfying the constraint in Equation (2) (lines 11-12). This update preserves feasibility while reducing the dual objective function. The introduction of new edges that satisfy $slack = 0$ enables further exploration in subsequent iterations. The algorithm terminates when neither additional augmenting paths nor dual variable updates are possible, at which point an optimal solution to both the primal and dual problems is achieved.

Graph Partitioning: Graph partitioning plays a crucial role in managing design complexity and improving overall design efficiency. Given a graph, the problem partitions the vertex set into k disjoint subsets while simultaneously satisfying two key objectives: the balance constraint and cut minimization. The balance constraint requires that each subset has approximately equal size, whereas cut minimization focuses on reducing the total weight of the edges crossing between different subsets. Owing to these properties, graph partitioning has been extensively employed to address critical challenges in physical design. Among various methods, METIS [24], [25] has emerged as a widely adopted and highly efficient partitioning algorithm. It leverages a multilevel partitioning framework, as illustrated in Algorithm 2, and proceeds in three main phases. First, the input graph is coarsened by iteratively collapsing vertices and edges, yielding a smaller yet structurally similar graph (lines 1–2). Second, an initial partition is generated on the coarsened graph using simple heuristics (line 3). Finally, the solution is projected back to the original graph through an uncoarsening process, during which refinement strategies are applied at each level to improve balance and reduce edge cuts (lines 4–6). For a general k-way partitioning problem, METIS often employs a recursive bisection strategy, repeatedly applying the above three phases until the required number of partitions is achieved.

III. Related Work

Early scheduling techniques primarily targeted the optimization of computations within a single memory array, with the objective of minimizing resource consumption. Approaches such as XMG-GPPIC [3], SIMPLER MAGIC [5], [26], as well as [27] and [28], employed different strategies to reduce either the number of logic gates or the required storage capacity. Nevertheless, these single-array strategies become impractical when applied to large-scale circuits that surpass the capacity of a single array, as they fail to address the significant energy overhead introduced by inter-array data communication.

To address the limitations of single-array designs, subsequent research advanced toward multi-array scheduling, with the primary objective of reducing inter-array data movement, particularly copy instructions. MASIM [8] represented one of the first multi-array scheduling frameworks, emphasizing the minimization of copy operations. Other approaches [6], [7], [29] similarly sought to mitigate inter-array transfers, either by partitioning circuits in a more structured manner or by leveraging hardware resources more effectively. However, a fundamental limitation of these methods lies in their lack of true parallelism, as they continue to execute only a single instruction per cycle. Moreover, MASIM [8] depends on randomized strategies, which results in non-deterministic performance outcomes. In contrast, our proposed scheduler, MAPSIM, achieves genuine parallel execution by enabling the concurrent scheduling of multiple instructions, thereby substantially decreasing both copy operations and overall execution latency in a deterministic and consistent manner.

IV. Problem Formulation

A. Terminology

A Boolean network usually can be regarded as a DAG, which can be denoted as $G(V, E)$. The vertex set V is typically a collection of PIs, POs, and internal logic gates within the Boolean network. The directed edge set $E \subseteq V \times V$ captures the connectivity between vertices, where the orientation of an edge represents the direction of data flow in the network. For each vertex $v \in V$, let $\text{FIs}(v)$ and $\text{FOs}(v)$ denote its fan-in and fan-out vertex sets, respectively. A directed edge $(u, v) \in E$ exists if and only if $u \in \text{FIs}(v)$ (equivalently, $v \in \text{FOs}(u)$). By definition, every PI has an empty fan-in set. The *depth* of a Boolean network is defined as the maximum logic level among all POs, while its *size* is the total number of vertices in G. The logic level $\iota(v)$ of a vertex v is recursively determined as

$$\iota(v) = \begin{cases} 0, & \text{if } v \in \text{PIs}, \\ \max\limits_{u \in \text{FIs}(v)} \iota(u) + 1, & \text{otherwise.} \end{cases} \quad (3)$$

In the scheduling process, vertices whose FIs reside in the array but have not yet been calculated are called **candidate vertices**. Scheduling decisions, whether to execute copy or computation instructions, are made exclusively based on these candidate vertices.

A bipartite graph is a graph denoted by $G = (U \cup V, E)$, where the vertex set is partitioned into two disjoint subsets U and V such that $U \cap V = \emptyset$, and every edge in E connects one vertex in U with one vertex in V. Formally, for all edges $(u, v) \in E$, we have $u \in U$ and $v \in V$. Equivalently, the graph G is bipartite if and only if it is 2-colorable; that is, there exists a function $col : U \cup V \rightarrow 1, 2$ such that $col(u) \neq col(v)$ for every edge $(u, v) \in E$. A fundamental property of bipartite graphs is that they contain no odd-length cycles; conversely, a graph without odd cycles is bipartite.

Due to their structural simplicity, bipartite graphs have broad applications in computer science, operations research, and related fields. Bipartite graphs are central to well-known algorithms, such as the Hungarian algorithm for maximum weight matching, which exploits the unique structural constraints of bipartite graphs to achieve efficient solutions.

B. Problem Formulation

The objective of MAPSIM is to schedule a given Boolean network onto a specified SIMD-IMC architecture. The effectiveness of the scheduler can be evaluated from multiple aspects. Our goal is to implement the logical functionality of the Boolean network using as few execution cycles as possible for a given number of arrays, while minimizing data movement caused by copy instructions, which, as reported in [3], leads to increased energy consumption. Hence, the optimization problem addressed in this paper can be formulated as follows:

Input:
- A Boolean network $G(V, E)$.
- N: The number of arrays available.
- R: The number of rows in each array.

Output: A schedule sequence S.

Constraints: All actions in each cycle satisfy the scheduling rules proposed in Section II-C.

Goal:

$$\min_{S \in \mathcal{S}} (g_{\text{copy}}(S), g_{\text{cycle}}(S)), \quad (4)$$

where $g_{\text{copy}}(S)$ and $g_{\text{cycle}}(S)$ represent the total number of copy instructions and the total number of execution cycles, respectively, under a given sequence S. The goal is to determine the sequence S that optimizes both $g_{\text{copy}}(S)$ and $g_{\text{cycle}}(S)$.

TABLE I. Description of frequently used symbols.

Symbol	Description
f	Scheduling potential energy in the SIMD-IMC.
g	Scheduling potential energy in each array.
α	The reward of the input existing in the array.
β	The penalty due to congestion in the array.
c	The reward of computing vertices in the Boolean network.
ρ	A small tuning parameter representing the potential benefit of adding a vertex into the array.
max_{fo}	Maximum fan-out for vertices in the Boolean network.

By addressing the scheduling task as a combinatorial optimization problem, MAPSIM generates solutions that trade off performance and energy, enabling designers to make informed choices under various architectural and application constraints.

C. Challenge

Scheduling is inherently a challenging task, and multi-objective scheduling optimization in a multi-array environment further exacerbates this complexity. The coexistence of multiple optimization objectives imposes unprecedented challenges on the scheduler. Firstly, it is essential to determine the assignment of the PIs of the Boolean network within the array, as this initialization step critically influences the effectiveness of the subsequent scheduling process. Secondly, the multi-objective nature of the problem necessitates careful trade-offs between conflicting goals, as minimizing execution cycles alone may lead to an increase in copy instructions in some cases, thereby incurring higher data movement overhead and energy consumption. Consequently, both objectives must be jointly considered when designing scheduling strategies. Finally, given that scheduling problems are generally NP-hard, the scheduler must efficiently navigate the exponentially growing search space to identify solutions that are as close as possible to the optimal instruction sequence within acceptable computational time.

V. Methodology

This section presents MAPSIM, a novel and efficient multi-array parallel scheduler designed to effectively address the challenges identified in Section IV-C. Specifically, Section V-B details a bipartite graph-based input assignment method to alleviate the congestion issues observed in prior work. Subsequently, Section V-C introduces a greedy scheduling procedure that leverages the computation of scheduling potential energy to navigate the complex state transition graph and determine an effective scheduling path. Finally, Section V-D presents an action decision strategy formulated using priority queues and the maximum weight matching algorithm. Together, these sections form an efficient and flexible framework capable of effectively scheduling Boolean networks in a given number of arrays. For clarity, the commonly used symbols in this section are summarized in TABLE I. These symbols serve as the foundation for the subsequent formulations and analyses, facilitating a clearer understanding of the proposed method.

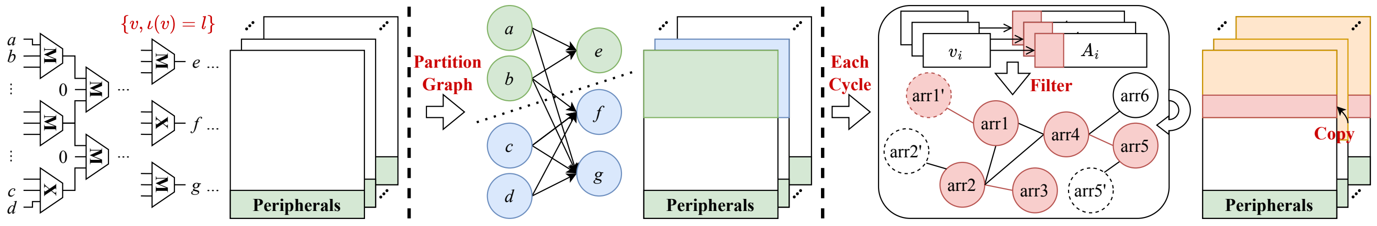


Fig. 3. MAPSIM overall flow.

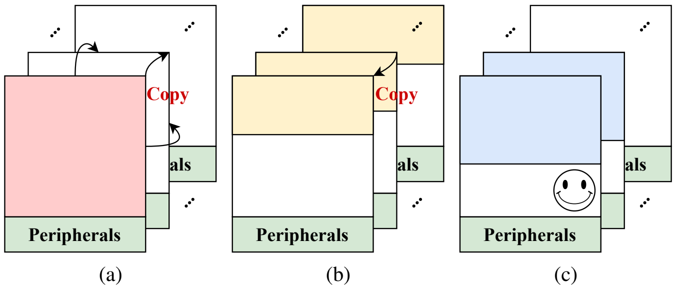


Fig. 4. Different input assignment methods with (a) serial assignment, (b) balanced assignment, and (c) partition assignment.

Algorithm 3: Input Assignment Method.

```
Input: A Boolean network $G(V, E)$, $k$, $l$.
Output: Input assignment $I$.
1 $V_l \leftarrow \{v \mid \iota(v) = l, v \in V\}$;
2 $E' \leftarrow \{\}$, $w' \leftarrow \{\}$;
3 for $n \in V_l$ do
4     for $pi \in PIs$ do
5         $E' \leftarrow E' \cup \{(pi, n)\}$;
6         $w'(pi, n) \leftarrow$ The shortest path count from $pi$ to $n$;
7 $I \leftarrow$ Run METIS on $G'(\text{PIs} \cup V_l, E', w')$ into $k$ parts;
8 return $I$
```

A. Overall Flow

The overall workflow of the proposed method is illustrated in Fig. 3. In the initial stage, we address the input assignment problem by employing an l-level input-feature bipartite graph partitioning method, which will be proposed in Section V-B. As shown on the left side of Fig. 3, the input vertices are partitioned into two disjoint parts, a, b and c, d, enabling independent access to multiple memory arrays and avoiding read conflicts. Next, for each cycle, we utilize the scheduling potential energy method introduced in Section V-C to evaluate the component of the proposed scheduling potential energy. This step aims to quantitatively measure how much each action contributes to the progress of the computation. A greedy strategy is then adopted to select the action combination with the highest overall benefit, ensuring efficient utilization of array-level parallelism. Finally, we apply the action selection strategy proposed in Section V-D to determine the final action combination for each execution cycle. Specifically, a selection strategy is used by filtering them through a priority queue. Then it employs a maximum weight matching algorithm to select a set of non-conflicting, legal actions with the highest cumulative weight. As illustrated on the right side of Fig. 3, the red-highlighted vertices and edges represent the matched edges with maximum weight and their corresponding vertices. To ensure the strategy is compatible with hardware constraints, a routing legalization step is performed, adapting the selected action combination to various physical interconnect structures. The following sections provide a detailed description of each step of MAPSIM.

B. Input Assignment

The foundation of enhancing parallelism is the availability of sufficient resources within each array to enable concurrent execution across these arrays. This necessitates the distributed storage of the PIs of the network across distinct arrays during the initial phase. Fig. 4 illustrates three input assignment strategies. In particular, [7], [8] adopt a serial assignment approach, which results in severe congestion. The balanced assignment method distributes inputs too sparsely, failing to provide sufficient inputs for each unprocessed vertex within each array for computation, thereby necessitating copying rows from other arrays. To balance input congestion and distribution sparsity, the input assignment must satisfy two key criteria. First, each array should contain a sufficient number of inputs to support the execution of unprocessed vertices. Second, the additional overhead introduced by parallel scheduling should be minimized.

It is impractical to perform some partitioning strategies directly on the entire DAG, as the DAG exhibits a complex structure of vertices and edges. In contrast, our objective is to determine the assignment of PIs to specific arrays. To address this challenge, this paper proposes an input-feature bipartite graph assignment method, which is outlined in Algorithm 3.

First, the algorithm selects a specific set of vertices V_l, which have no edges among themselves, together with all DAG PIs, and denotes the vertices union as PIs $\cup V_l$. Then it constructs a bipartite graph that connects PIs exclusively to V_l, serving as the input-feature bipartite graph $G'(\text{PIs}\cup V_l, E', w')$ for the DAG. This procedure is referred to as the l-level method (lines 1–2). In a DAG, there are no edges between two vertices at the same logic level ι, nor between POs. However, since POs are typically far from PIs, using POs as the other side of the input-feature bipartite graph may not benefit scheduling, as scheduling is generally a local process. As a result, we define V_l as

$$V_l = \{v \mid \iota(v) = l, v \in V\}, \text{ where } \forall l' > l, |V_{l'}| < R. \quad (5)$$

For any vertex $pi \in$ PIs and $n \in V_l$, there exists an edge

(pi, n) in the new edge set E', whose weight w' is defined as the number of distinct paths from pi to n (lines 3-6). The weight w' of each edge in E' reflects an estimate of the number of copy instructions required if pi and n are placed in different partitions. This assumption reflects the trend that if more paths connect pi and n, their dependencies will be stronger, so we need to allocate them to the same partition. However, since enumerating all paths is computationally expensive, we only compute the number of shortest paths from pi to n, which can be efficiently obtained using a breadth-first search algorithm. The algorithm then employs METIS [24], [25] to partition the input-feature bipartite graph into k parts, denoted as I (line 7). The value of k depends on the number of PIs, the number of rows per array, and the size of the Boolean network. Specifically, k is determined such that the inputs assigned to each partition occupy approximately 1/4 or 1/2 of the array capacity R, *i.e.*,

$$k = \left\lceil \frac{4|\text{PIs}|}{R} \right\rceil \text{ if } |V| > Size_t \text{ else } \left\lceil \frac{2|\text{PIs}|}{R} \right\rceil, \tag{6}$$

where R denotes the number of rows per array, $|V|$ is the number of vertices and $Size_t$ is a vertex count threshold. When $|V| > Size_t$, more space is reserved in each array to accommodate subsequent computations.

C. Multi-Array Parallel Scheduler

Scheduling in multi-array poses significant challenges, as finding an optimal solution within polynomial time is generally intractable. Nevertheless, heuristic approaches can often yield promising results in both temporal and spatial dimensions. This section proposes a greedy scheduling strategy. First, the following concepts about the usage of the array are defined:

- **Array State:** describes the data state across all arrays of the SIMD-IMC architecture at a certain cycle.
- **State Transition:** describes the state transition process through the execution of a given set of instructions, which are also called actions.
- **State Transition Graph:** describes all array states and the possible transitions among them. In a state transition graph, each node s represents a state, and an edge between two nodes corresponds to a set of actions executed within a single cycle, capturing the state transitions induced by either computation or data movement. Each node is associated with a property f, and each edge is assigned a weight Δf, defined as the difference of f between the sink and source nodes.

As illustrated in Fig. 5, s_0 denotes the initial state obtained from Section V-B, and s_{end} denotes the state in which all vertices in the Boolean network have been computed. Hence, the scheduling problem becomes how to find a path from the initial state s_0 to the target state s_{end} while minimizing the cost, *i.e.*, the shortest path problem. To guide this search, we introduce a metric termed the scheduling potential energy, defined as

$$f(s) = \sum_{v \in V} \max_{arr \in Arr} g_{v,arr}, \tag{7}$$

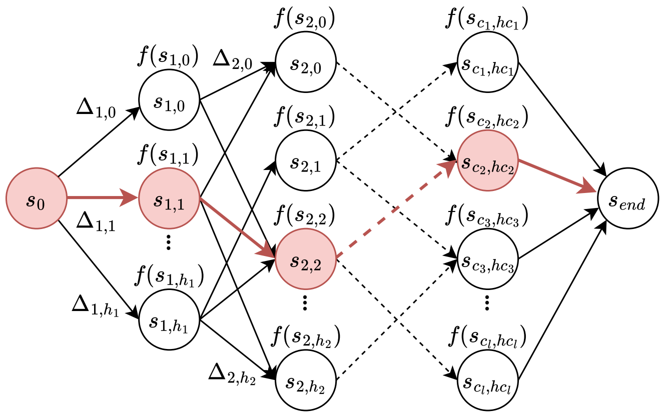


Fig. 5. State transition graph of the scheduler, where a shortest path is highlighted in red.

where $g_{v,arr}$ represents the scheduling potential energy of vertex v in the Boolean network with respect to a given array arr. If there is an input vertex of v that has not been computed in the SIMD-IMC architecture, then $g_{v,arr} = 0$. Otherwise, $g_{v,arr}$ is given by

$$g_{v,arr} = \alpha_{v,arr} - \beta_{v,arr} + c_{v,arr} + \rho_{v,arr}, \tag{8}$$

where $\alpha_{v,arr}$ depends on the number of FIs to vertex v in arr, and reflects the likelihood of computing v in that array; $\beta_{v,arr}$ measures congestion in arr and indicates resource availability; $c_{v,arr}$ quantifies the increase in scheduling potential energy resulting from computing v (set to zero If v cannot be computed in arr); and $\rho_{v,arr}$ is a small tuning parameter that represents the potential benefit of adding vertex v into arr.

As the state transition graph is huge, it is impractical to traverse all array states. As an alternative, we use a greedy method to select the next state s_{t+1} at each step from the current state s_t by maximizing the increase of f. This ensures that, for each state s_t, at least one next state can be obtained. In other words, the transition of each state can be viewed as the partial derivative of the scheduling potential energy, which can be defined as

$$\frac{\partial f(s_t)}{\partial s_{t+1,k}} = f(s_{t+1,k}) - f(s_t) = \Delta_{t,k}. \tag{9}$$

And the rule that we must obey is

$$\max_k \Delta_{t,k} > 0. \tag{10}$$

This implies that at least one path exists from s_0 to s_{end} with monotonically increasing potential energy when scheduling is considered. As a prerequisite for the successful identification of such a path, we analyze the computation of the parameters α, β, and c under four distinct scenarios.

Case 1: The array arr may contain empty rows, and if not all required FIs for computing the vertices are present in arr, the corresponding FIs must be copied from other arrays ($\beta = 0, c = 0$). To support this operation, the reward factor $\alpha_{v,arr}$ is defined to vary with the number of FIs already present in the array, satisfying $\alpha^3_{v,arr} > \alpha^2_{v,arr} > \alpha^1_{v,arr}$, where the superscript is the number of FIs in arr. Fig. 6 illustrates an example in this scenario. In arr, the FIs that have already

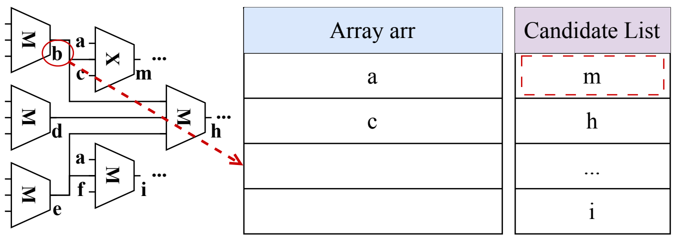


Fig. 6. Example of **Case 1**: Selecting vertex m from the candidate list requires copying vertex b from another array for computation.

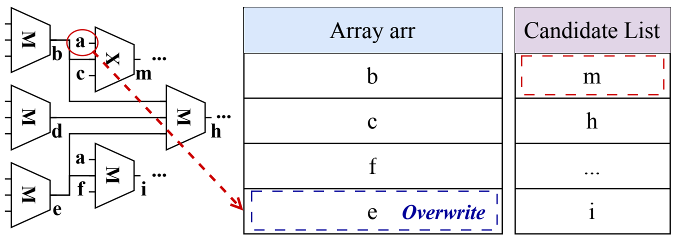


Fig. 7. Example of **Case 2**: Selecting vertex m from the candidate list for computation requires overwriting vertex e with copying vertex a from another array.

appeared for m, h, and i in the candidate list are 2, 0, and 1 respectively, so m should be executed first. The next step is to copy vertex b into arr.

Case 2: If the array arr contains no empty rows, no vertices can be computed directly, and some rows in arr are replicated in other arrays, a copy operation should be performed within arr to enable future computation opportunities ($\beta \neq 0, c = 0$). In this case, overwriting any row from arr will result in a significant change in the overall scheduling potential energy g. In the worst case, each row contributes equally to g, so evicting any row from arr results in the same loss. This implies that all candidate vertices in arr share a common FI count, denoted as $|fi|$. In this scenario, adding a row inevitably increases the number of FIs associated with at least one candidate vertex, which is reflected by an increase in its α value from $\alpha_{v,arr}^{|fi|}$ to $\alpha_{v,arr}^{|fi+1|}$, where the superscript denotes the number of such FIs. On the other hand, evicting a row may decrease the number of FIs associated with up to max_{fo} candidate vertices, each potentially experiencing a reduction in its α value from $\alpha_{v,arr}^{|fi|}$ to $\alpha_{v,arr}^{|fi-1|}$. Here, max_{fo} denotes the maximum FOs number among all vertices in this DAG. And in the worst case, the congestion parameter remains unchanged during the copy operation, as the number of occupied rows stays constant ($\Delta\beta = 0$). To ensure that the overall potential energy gain still satisfies the requirement of Equation (10), the increase in readiness for the vertex v due to the added row must be at least equal to the total readiness loss of all affected vertices u due to the evicted row. That is, the updated readiness $\alpha_{v,arr}^{|fi+1|}$ must satisfy the inequality $\alpha_{v,arr}^{|fi+1|} \geq \sum_{u \in max_{fo}} \alpha_{u,arr}^{|fi|}$. Based on this relationship, we define the readiness factor $\alpha_{v,arr}$ of a vertex v in array arr as the proportion of its input count that is already available in the array, which can be defined as

$$\alpha_{v,arr} = max_{fo}^{|fi_{v,arr}|}, \tag{11}$$

where $|fi_{v,arr}|$ means the input number of vertex v in array

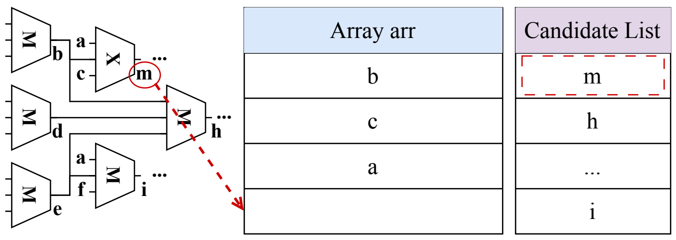


Fig. 8. Example of **Case 3**: Selecting vertex m from the candidate list for computation in arr.

arr. Fig. 7 illustrates an example of this scenario, where each vertex in the candidate list has two FIs in arr. arr is congested, and vertex e has copies in other arrays. To allow candidate vertices to be processed in arr, vertex e is overwritten. Congestion is not alleviated, and the penalty β remains unchanged. However, vertices m and i both receive an increased α reward due to the addition of vertex a.

Case 3: When array arr contains available rows and all the FIs to a vertex v are already present in the array, v can be computed directly within arr ($\beta = 0, c \neq 0$). In order to meet the requirement specified in Equation (10), the computation of this vertex must yield a positive profit, formally expressed as $c > 0$. The influence of available rows on the computational profit is captured by defining the profit c as a function of both the structural readiness of the vertex and the storage flexibility of the array. Specifically, the more available rows in arr, the higher the potential for the scheduler to accommodate more rows for future operations, thereby decreasing the benefit of the computation of v in the current cycle. Hence, the profit c is modeled as

$$c_{v,arr} = max_{fo}^{\max(\lceil 4 \times r / R \rceil, 2)}, \tag{12}$$

where the parameter r denotes the number of rows occupied in arr. When r is less than half of the total rows R, arr favors the addition of more replicated rows, thereby enhancing its parallel scheduling capability. Under this condition, the computational benefit satisfies $c_{v,arr} < \alpha_{v,arr}^3$. When r exceeds half but remains below three-quarters of R, the computational benefit increases moderately, reaching $c_{v,arr} = \alpha_{v,arr}^3$. However, when r exceeds three-quarters of R, arr begins to experience congestion. At this stage, the scheduler should prioritize vertices that can be computed directly within arr, resulting in $c_{v,arr} > \alpha_{v,arr}^3$. Fig. 8 illustrates an example of computing a vertex. All FIs to vertex m are stored in arr, so vertex m can be computed. Since the data capacity of arr is equal to $\frac{3}{4}$ of the total capacity, the benefit of computing the vertex is $c_{m,arr} = max_{fo}^3$.

Case 4: When array arr does not have available rows, and none of its rows have replicas in other arrays, it becomes necessary to evict a row to make one row available for further scheduling ($\beta \neq 0, c = 0$). In such scenarios, we prioritize the execution of vertices that are immediately ready, whose input number satisfies $|fi| = 3$. Accordingly, for vertices with $|fi| < 3$, we expect their scheduling potential energy $g_{v,arr}$ to be close to zero. In this situation, the inequality $\beta_{v,arr} \geq \alpha_{v,arr}^2$ must be satisfied, ensuring that the congestion term β effectively suppresses scheduling when readiness is

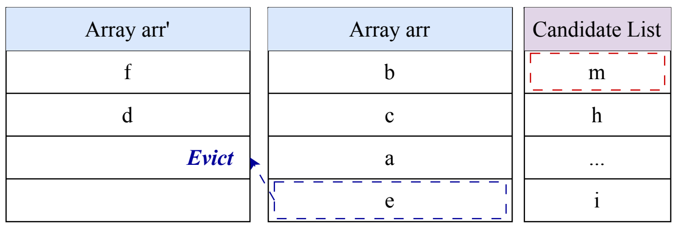


Fig. 9. Example of **Case 4**: Selecting vertex m from the candidate list for computation requires evicting vertex e to another array.

insufficient. Now, assume all candidate vertices have $|fi| = 3$, and consider the worst case where each row contributes equally to the same number of candidate vertices, denoted as $|fo|$. In this case, evicting any row will result in a decrease of α, while causing an increase in β for at least $|fo| \times \lceil R/3 \rceil$ candidate vertices, where R is the number of rows in arr. By canceling the common factor $|fo|$, the condition to maintain a non-declining overall scheduling potential energy simplifies to $\beta_{v,arr} \geq \frac{max_{fo}^3}{\lceil R/3 \rceil}$. This inequality guarantees that the reduction in congestion, as measured by β, sufficiently compensates for the loss in computational readiness represented by α during row eviction. In other words, the scheduler must carefully balance the trade-off between the decline caused by congestion and the benefits of maintaining some data to optimize scheduling performance. Failure to satisfy this condition could lead to suboptimal scheduling decisions that degrade overall efficiency. Based on this analysis, we define the calculation of β as follows:

$$\beta_{v,arr} = \max\left(\frac{max_{fo}^3}{\lceil R/3 \rceil}, max_{fo}^2\right). \tag{13}$$

Fig. 9 illustrates an example in this scenario, where the scheduling potentials g of three vertices m, h, and i in the candidate list within this array are $\alpha^3 - \beta$, $\alpha^2 - \beta$, and $\alpha^2 - \beta$, respectively. Since $\beta > \alpha^2$, evicting e does not result in a potential loss. For m, this action generates a gain of β, allowing the scheduler to execute successfully.

Moreover, $\rho_{v,arr}$ is a small tuning parameter that does not affect the overall monotonicity of the scheduling metric. Instead, it serves as a fine-grained adjustment to enhance the screening capability when other parameters are identical or similar. Specifically, it can be defined as

$$\rho_{v,arr} = |\{u \mid u \in \text{FIs}(b) \wedge u \in arr, b \in \text{FOs}(v)\}|, \tag{14}$$

which represents the number of FIs with replicates in arr that belong to the set of FIs of v's fan-out nodes.

D. Action Decision

Although the greedy approach is employed to limit the exploration of the state transition graph, the complexity of the underlying scheduling rules described in Section II-C makes it difficult to identify a set of legal actions to maximize the f in each cycle. An action selection strategy based on the maximum weight matching algorithm is proposed to address this challenge. It efficiently approximates the optimal scheduling decision while adhering to system constraints. First, the following structure about action selection is defined:

- **Action Graph:** is introduced to determine the set of actions for each cycle, denoted as $G_a(V, E, w)$. Here, V corresponds to the arrays or the virtual vertices derived from them, such as arr and arr'. The edge $(arr_1, arr_2) \in E$ denotes a feasible action between vertices arr_1 and arr_2, and the edge weight $w(arr_1, arr_2)$ quantifies the increase in potential energy associated with the action.

The overall selection process is illustrated in Algorithm 4. The input of the algorithm A is a set that contains all executable actions in the current cycle. The following information characterizes each action $a \in A$: (1) its instruction type (either computation or copy), (2) the associated array(s), and (3) the increment on f, denoted by Δf. The objective is to select a subset $A' \subseteq A$ of legal actions that maximizes the increase of $\sum \Delta f$. The vertices V in the action graph G_a represent physical arrays, and additional virtual vertices may be introduced to satisfy the constraints of the algorithm (line 2). There is no doubt that A in each cycle is large.

Algorithm 4: Action Decision Algorithm.

```
   Input: Action set $A$, candidate vertex set $C$, array set $Arr$.
   Output: Selected action set $A'$.
1  $E \leftarrow \{\}, w \leftarrow \{\}$;
2  $V \leftarrow$ Regard each $arr \in Arr$ as a vertex;
3  $A' \leftarrow$ Filter $A$ with max $\Delta f$ for each $v \in C$;
4  for $a \in A'$ do
5      if $a$ is associated with computation instruction then
6          $arr \leftarrow$ The array associated with action $a$;
7          Create a virtual vertex $arr'$ for $arr$ into $V$;
8      else
9          $arr, arr' \leftarrow$ The arrays associated with action $a$;
10     if $(arr, arr') \notin E \vee \Delta f > w(arr, arr')$ then
11         $w(arr, arr') \leftarrow \Delta f$;
12         $E \leftarrow E \cup \{(arr, arr')\}$;
13 $A' \leftarrow$ Run maximum weight matching on $G_a(V, E, w)$;
14 $A' \leftarrow$ Legalize actions in $A'$;
15 return $A'$
```

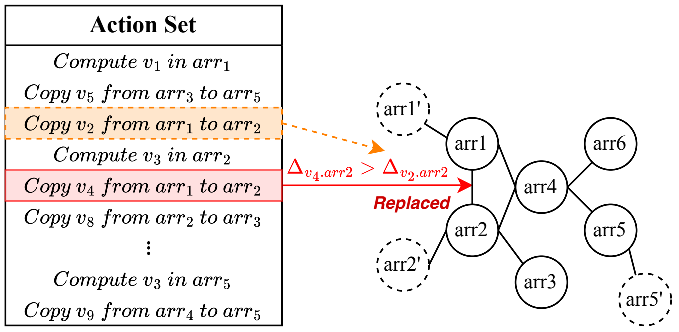

Action Set
Compute v_1 in arr_1
Copy v_5 from arr_3 to arr_5
Copy v_2 from arr_1 to arr_2
Compute v_3 in arr_2
Copy v_4 from arr_1 to arr_2
Copy v_8 from arr_2 to arr_3
⋮
Compute v_3 in arr_5
Copy v_9 from arr_4 to arr_5
$\Delta_{v_4,arr2} > \Delta_{v_2,arr2}$
Replaced
arr1'
arr1
arr2
arr2'
arr3
arr4
arr5
arr5'
arr6

Fig. 10. Action graph generation, where the red instruction replaces the yellow instruction due to better gain.

To reduce its size, we first evaluate all actions associated with each candidate vertex $v \in C$ in the Boolean network and only select the action that yields the largest increase in scheduling potential energy, denoted $\max \Delta f$, while other actions in other arrays associated with this candidate vertex will be discarded (line 3). This step ensures that each candidate vertex will be considered in at most one action in the decision period, effectively filtering redundant operations and reducing computational overhead.

Subsequently, an action graph is constructed based on the selected actions a (lines 4-12) to facilitate the application of a maximum weight matching algorithm. For a computation instruction that operates on a single array arr, a virtual vertex arr' is introduced to form an action pair (arr, arr') (lines 5-7). This transformation enables computation instructions, which inherently involve only one array, to be incorporated into the matching framework defined over pairs of nodes. In contrast, a copy instruction involves two arrays, denoted as arr and arr', which can be identified directly (lines 8-9). If the edge does not exist or $\Delta f > w(arr, arr')$, then an edge is added between them with weight $w(arr, arr') = \Delta f$, indicating the benefit of executing this instruction (lines 10-12). This construction encodes all actions into an action graph representation, thereby facilitating the application of the maximum weight matching algorithm to identify a subset of actions that is both conflict-free and maximally beneficial for execution in the current scheduling cycle. The procedure for constructing an action graph is illustrated in Fig. 10. In this graph, virtual vertices are marked with dotted lines, and edges are generated by the algorithm based on the filtered action sequence, establishing a one-to-one correspondence between edges and actions. During graph construction, if an action with a higher weighted benefit is scheduled to occupy the same array, the existing action represented by the corresponding edge is replaced, which is highlighted in red in the graph.

During the actual scheduling process, each array can only execute one operation per cycle, which corresponds exactly to the constraints in maximum graph matching. Since the virtual vertices generated by the computation instructions do not have edges with other vertices, the edges between them can well reflect the scheduling benefits brought by the computation of the vertices. Based on these, the algorithm then applies the maximum weight matching algorithm mentioned in Section II-D to find the best match in the action graph (line 13). Although the time complexity of the Edmonds' Blossom algorithm [23] is approximately $\mathcal{O}(|V_a|^3)$, we define V_a as the number of arrays, which is typically a small constant. As a result, the computational overhead introduced by the Edmonds' Blossom algorithm is negligible in practice, which significantly accelerates our scheduling procedure.

Finally, the selected action sequence is validated under different interconnection schemes (line 14). Because the proposed maximum weight matching algorithm operates independently of the underlying hardware routing constraints, a post-selection validation step is required. This step filters the candidate action set to ensure feasibility with respect to the specific interconnection architecture. The actual set of executable actions may vary depending on routing limitations, such as bandwidth constraints, simultaneous access conflicts, or architectural restrictions on array connections. This validation ensures that the final action set not only maximizes the scheduling potential energy but also adheres to the physical constraints of the system, thereby enabling practical deployment in real-world hardware environments.

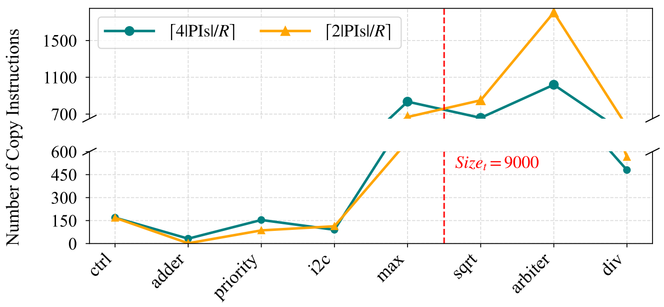


Fig. 11. Number of copy instructions of the two k-value selection methods, with EPFL [30] circuits arranged in ascending order of vertex count.

VI. Experimental Results

This section presents the experimental results of the proposed MAPSIM framework. The framework was implemented in C++ and evaluated using circuits from the EPFL benchmark suite [30]. All experiments were conducted on a machine equipped with an Intel(R) Xeon(R) Platinum 8350C processor and 1.5 TB of memory. Due to execution constraints of peripheral devices [3], all Boolean networks were converted to the XMG format before scheduling. This conversion is carried out using the open-source logic synthesis and optimization library mockturtle [31]. By applying the same transformation process to all scheduling methods, we ensured structural consistency across the Boolean networks, allowing for a more accurate comparison of the intrinsic performance of the scheduling algorithms.

A. Setting of $Size_t$ value

In the input assignment strategy, selecting an appropriate value of k is critical, as it directly affects the partitioning outcome and consequently influences the subsequent scheduling process. The choice of k is partially determined by the parameter $Size_t$. To analyze this effect, we conducted experiments using two distinct selection strategies. During the experiments, k was fixed according to one of the two calculation methods described in Section V-B to evaluate their respective impacts. To more intuitively illustrate the rationale behind our choice of $Size_t$, the circuits were sorted in ascending order based on the number of vertices. We selected eight representative circuits that exhibited notably different results on copy instruction count under the two selection methods of k, and the corresponding outcomes are presented in Fig. 11.

As illustrated in Fig. 11, when the number of circuit vertices is smaller than `i2c`, selecting the method $k = \lceil 2|\text{PIs}|/R \rceil$ significantly reduces the number of copy instructions. Conversely, when the number of circuit vertices exceeds `sqrt`,

TABLE II. Comparison results with different single-array schedulers.

Circuit	R	XMG-GPPIC [3]		MAGIC [5]		STAR [6]		TCAD25 [7]		[7]+IG		[7]+Impr		MAPSIM	
		Copy	Cycle	Copy	Cycle	Copy	Cycle	Copy	Cycle	Copy	Cycle	Copy	Cycle	Copy	Cycle
int2float	16	252	451	220	419	222	421	211	420	105	304	179	378	151	211
router	64	188	389	170	371	169	370	176	377	88	281	152	353	46	221
cavlc	64	421	1021	205	805	281	881	54	654	24	624	49	649	14	614
priority	128	238	789	238	789	238	789	238	789	128	679	238	789	86	603
dec	256	18	322	20	324	18	322	18	322	9	313	18	322	10	314
adder	256	512	768	512	768	512	768	512	768	256	512	512	768	2	258
max	256	2294	4118	1726	3550	1725	3549	1723	3547	1250	3074	1723	3547	668	1456
sin	256	687	4169	1574	3636	572	3534	423	3905	132	3612	361	3841	71	3548
sqrt	256	4915	14155	4639	13879	4701	13941	4331	13571	1307	10547	4039	13279	659	9082
multiplier	256	16009	30185	7303	21479	11231	25407	5961	20137	1736	15912	5525	19701	2437	11946
div	256	6144	18680	6013	18549	7609	20145	5812	18348	926	13462	4986	17522	480	12725
log2	256	11388	31148	14702	34462	12581	32341	7043	26803	3878	23638	6414	26174	5063	19422
Ave. ratio	/	28.22	1.84	27.57	1.69	27.21	1.73	25.06	1.63	11.99	1.31	24.69	1.59	1.00	1.00

TABLE III. Comparison results with the multi-array scheduler.

Circuit	R	|V|	MASIM [8]			MAPSIM(W.O.I.A)			MAPSIM(POs Method)			MAPSIM(l-level Method)		
			Copy	Cycle	Time(s)	Copy	Cycle	Time(s)	Copy	Cycle	Time(s)	Copy	Cycle	Time(s)
adder	256	256	256	512	0.45	256	512	0.19	2	258	0.12	2	258	0.20
arbiter	256	11839	5853	17692	275.17	1433	7739	82.28	2126	7725	35.27	1019	7666	76.28
bar	256	3336	468	3804	27.44	582	3564	22.44	443	3490	30.44	346	3280	30.44
cavlc	64	600	19	619	13.05	14	614	1.23	14	614	0.38	14	614	1.23
ctrl	16	174	53	227	5.04	59	168	0.09	59	168	0.09	59	168	0.17
dec	256	304	9	313	12.86	10	314	0.32	10	314	0.33	10	314	0.74
div	256	12536	872	13408	39.62	688	12894	34.16	480	12725	24.18	480	12725	34.16
hyp	512	214335	53084	267419	103679.00	27187	176275	10954.78	18503	220147	10936.64	14227	204491	11665.78
i2c	256	1342	74	1416	13.64	19	1361	3.42	113	1247	3.36	113	1247	3.36
int2float	16	199	87	286	12.73	147	205	0.14	152	213	0.38	151	211	0.34
log2	256	19760	3311	23071	669.37	5437	18087	79.32	5063	19422	182.02	5063	19422	83.46
max	256	1824	937	2761	14.65	691	1953	3.21	816	1776	3.09	668	1456	3.11
memctrl	512	46836	17118	63954	5566.22	13657	31359	382.55	11134	30007	290.57	11281	31612	389.12
multiplier	256	14176	1440	15616	126.75	2161	11914	85.46	2637	12024	79.36	2437	11946	83.46
priority	128	551	128	679	5.03	128	679	2.78	86	603	3.96	86	603	3.96
router	64	201	70	271	2.47	86	275	0.16	46	221	0.16	46	221	0.16
sin	256	3482	120	3602	20.13	71	3548	7.94	71	3548	10.99	71	3548	9.44
sqrt	256	9240	1290	10530	49.33	867	8637	19.38	659	9082	10.68	659	9082	21.38
square	256	18484	1760	20240	80.01	1380	18736	154.99	1143	18955	148.77	1143	18955	156.99
voter	256	13758	2212	15970	5270.35	1847	11622	42.43	2727	6480	136.44	899	5290	39.41
Ave. ratio	/	/	7.99	1.43	15.02	7.59	1.14	0.88	1.20	1.03	1.00	1.00	1.00	1.00

the opposite trend is observed. This is because larger circuits contain more vertices, which demand greater computational space and force the input data to be distributed across more arrays. For circuits of intermediate size, both calculation methods yield comparable performance. However, considering that the `max` circuit requires a larger number of copy operations, we ultimately set the value of $Size_t$ to 9000.

B. Results on Different Schedulers

The performance of MAPSIM was compared against recent single-array schedulers [3], [5]–[7] and the state-of-the-art multi-array scheduler [8]. To provide a comprehensive evaluation, we used the improved results in single-array schedulers from [8] for our experiments. Specifically, when there is a tie in priority among these schedulers, the scheduler will make a random selection instead of selecting the one with the lowest index. These schedulers ran the random version multiple times until it reached the same runtime as MASIM and reported the best result. The `[7]+IG` variant introduces the greedy strategy described in its work, and `[7]+Impr` applies iterative improvement to the original scheduling method.

The comparison results against various single-array schedulers are summarized in TABLE II. MAPSIM consistently achieves the lowest execution cycles across nearly all benchmark circuits. This performance advantage is primarily attributed to its ability to exploit parallelism via multi-array scheduling. In our SIMD-IMC architecture, we configured the number of arrays N to 8, with each array containing R rows, where R ranges from 16 to 256. The threshold was set to $Size_t = 9000$. Regarding interconnection schemes, we adopt the most restrictive approach, allowing only a single copy instruction per cycle. Overall, MAPSIM outperforms all competitors in both execution cycles and instruction overhead, demonstrating superior effectiveness and efficiency. On average, MAPSIM reduces the number of execution cycles by 39.50%, 35.19%, 36.34%, 32.48%, 19.21%, and 30.85% compared to [3], [5]–[7] and the enhanced variants `[7]+IG`, and `[7]+Impr`. Furthermore, MAPSIM substantially reduces the number of copy instructions, benefiting from an optimized input allocation strategy and an effective selection of scheduling candidates. In terms of execution time, MAPSIM achieves average reductions by 74.99%, 73.15%, 72.86%, 66.04%,

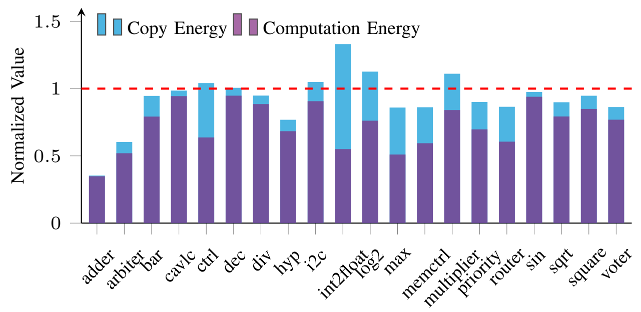


Fig. 12. Normalized energy consumption of MAPSIM relative to MASIM [8], including energy from copy and computation instructions across various benchmarks.

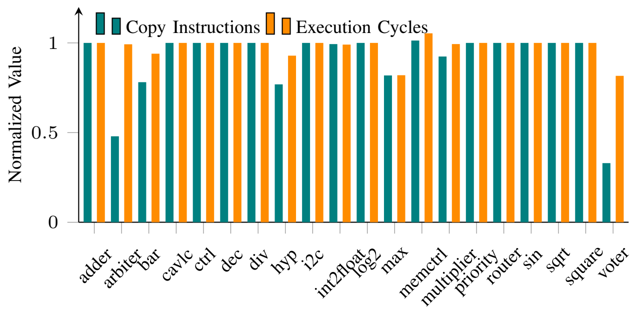


Fig. 13. Normalized numbers of copy instructions and execution cycles for the l-level Method relative to the POs method across various benchmarks.

23.84%, and 63.11% relative to the same set of methods and variants.

We also compare MAPSIM and its variants with the state-of-the-art multi-array scheduler MASIM. The results are summarized in TABLE III. The vertex number of each circuit is represented as $|V|$. Specifically, `MAPSIM(W.O.I.A)` denotes the results obtained without employing the input-assignment strategy. `MAPSIM(POs Method)` refers to the variant that uses POs as the counterpart of input-feature bipartite graph construction during input-assignment. `MAPSIM(l-level Method)` (will be referred to as MAPSIM below) corresponds to the scheduling results of the method described in Section V.

Firstly, without employing the input-assignment strategy, MAPSIM reduces the number of copy instructions and execution cycles by 7.57% and 16.87%, respectively, compared with MASIM. These results clearly demonstrate the effectiveness of the proposed greedy scheduling strategy based on scheduling potential, as well as the action selection mechanism that utilizes maximum-weight matching in Section V-C and Section V-D. Moreover, MAPSIM exhibits remarkable efficiency. Without the input-assignment strategy, its runtime is approximately 19.89× faster than that of MASIM, further highlighting the computational advantage of our approach. The advantages in copy instructions and execution cycles are amplified after the input-assignment strategy is applied. Overall, MAPSIM significantly reduces copy instruction count by 19.78%, execution cycle count by 23.43%, and has a 15.02× faster runtime. Among the 20 circuits evaluated, MAPSIM outperformed MASIM in execution cycle count except one (the `dec` circuit). In particular, for circuits such as `arbiter`, `memctrl`, and `voter`, MAPSIM required less than half the execution cycles of MASIM, indicating a substantial improvement in scheduling efficiency. In the `adder` circuit, the number of copy instructions is reduced by nearly 99.21%, attributed to the novel input-assignment strategy that effectively alleviates congestion and preserves more rows for computation. Moreover, the efficiency advantage of MAPSIM is particularly pronounced for large-scale circuits. For example, in `voter`, MAPSIM achieves a runtime speedup of approximately 133.75× over MASIM. Such results highlight the high computational efficiency of the proposed method.

C. Results on Energy Consumption

Based on the energy breakdown of computing stages reported in [3], a copy instruction consumes 1.87× the energy of a computation instruction (accounting for double precharge, discharge, and single write-back). Using this energy model, Fig. 12 presents the normalized energy relationship between MAPSIM and MASIM across benchmarks. The energy consumption is divided into copy instruction energy (blue) and computation instruction energy (purple), allowing for a clearer analysis of MAPSIM's energy consumption across different circuits. MAPSIM outperforms the state-of-the-art scheduler in 14 out of the 20 benchmark circuits. In the best case, MAPSIM reduces energy consumption to 35.21% of MASIM. On average, it achieves a 7.81% reduction in energy consumption compared to MASIM. These results highlight the effectiveness of MAPSIM in reducing energy usage.

D. Results on Different Input-Assignment Strategies

Note that the proposed MAPSIM method employs the l-level assignment described in Section V-B. An alternative strategy is to use POs as vertices on one side of the bipartite graph, referred to as the POs method. To evaluate the impact of these two input-assignment strategies on subsequent scheduling performance, we conducted experiments across all benchmark circuits in the EPFL suite, with the results shown in TABLE III and Fig. 13. The experimental results show that, in general, both methods achieve comparable total execution cycles and numbers of copy instructions. Nevertheless, for certain benchmark circuits, the l-level method outperforms the POs method, reducing the number of instruction copies and total execution cycles by an average of 9.46% and 2.33%, respectively. These results corroborate our discussion in Section V-B that using POs as one side of the bipartite graph may compromise scheduling efficiency by favoring locally optimal decisions over globally optimized scheduling.

VII. Conclusion

This paper presented MAPSIM, an efficient multi-array parallel scheduler. To mitigate input congestion, we introduced a novel partitioning strategy that constructs a bipartite graph from input features to effectively solve the input assignment problem. For multi-array parallel scheduling, we modeled the data state of each array as an array state and assigned each state a scheduling potential energy. By greedily searching for the maximum achievable energy gain within each state, the scheduler rapidly determines scheduling decisions. To speed up the transition between states, we employed priority queue filtering and maximum-weight matching to identify the near-optimal and feasible next state. Experimental results demonstrated that MAPSIM reduced execution cycles by 23.43% and copy instructions by 19.78% compared to the state-of-the-art scheduler, while achieving a $15.02\times$ speedup in runtime.

References

[1] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, no. 2, pp. 39–50, 2017.

[2] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. a. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: a framework for bit-serial SIMD processing using DRAM," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, 2021, p. 329–345.

[3] C. Nie, X. Cai, C. Lv, C. Huang, W. Qian, and Z. He, "XMG-GPPIC: Efficient and robust general-purpose processing-in-cache with XOR-majority-graph," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2023, p. 183–187.

[4] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, 2020.

[5] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, and S. Kvatinsky, "SIMPLER MAGIC: Synthesis and mapping of in-memory logic executed in a single row to improve throughput," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 10, pp. 2434–2447, 2020.

[6] F. Wang, G. Luo, G. Sun, J. Zhang, J. Kang, Y. Wang, D. Niu, and H. Zheng, "STAR: Synthesis of stateful logic in RRAM targeting high area utilization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 5, pp. 864–877, 2021.

[7] X. Qian, C. Lv, Z. He, and W. Qian, "A recursive partition-based in-memory SIMD computation scheduler for memory footprint minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 44, no. 6, pp. 2105–2118, 2025.

[8] X. Qian, C. Nie, Z. He, and W. Qian, "MASIM: An energy-efficient multi-array scheduler for SIMD logic-in-memory architectures," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025, pp. 1–9.

[9] N. Talati, A. H. Ali, R. Ben Hur, N. Wald, R. Ronen, P.-E. Gaillardon, and S. Kvatinsky, "Practical challenges in delivering the promises of real processing-in-memory machines," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2018, pp. 1628–1633.

[10] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 481–492.

[11] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: bit-serial in-cache acceleration of deep neural networks," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2018, p. 383–396.

[12] W. A. Simon, Y. M. Qureshi, M. Rios, A. Levisse, M. Zapater, and D. Atienza, "BLADE: An in-cache computing architecture for edge devices," *IEEE Transactions on Computers*, vol. 69, no. 9, pp. 1349–1363, 2020.

[13] R. Fan, Y. Cui, W. Li, M. Wang, and Z. Li, "MagiCache: A virtual in-cache computing engine," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2025, p. 1806–1818.

[14] W. Haaswijk, M. Soeken, L. Amarù, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2017, pp. 151–156.

[15] L. Hellerman, "A catalog of three-variable or-invert and and-invert logical circuits," *IEEE Transactions on Electronic Computers*, vol. EC-12, no. 3, pp. 198–223, 1963.

[16] R. Fu, J. Huang, M. Wang, Y. Nobuyuki, B. Yu, T.-Y. Ho, and O. Chen, "BOMIG: A majority logic synthesis framework for AQFP logic," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2023, pp. 1–2.

[17] R. Fu, R. Zhang, Z. Zheng, Z. Shi, Y. Pu, J. Huang, Q. Xu, and T.-Y. Ho, "Late breaking results: Hybrid logic optimization with predictive self-supervision," in *ACM/IEEE Design Automation Conference (DAC)*, 2025.

[18] R. Fu, R. Zhang, Z. Zheng, Z. Shi, Y. Pu, J. Huang, B. Yu, Q. Xu, and T.-Y. Ho, "CHOP: Clustered hybrid optimization for logic synthesis with self-supervised prediction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–14, 2026.

[19] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, Apr. 2010.

[20] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 7, pp. 994–1007, 2012.

[21] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[22] H. W. Kuhn, "Variants of the Hungarian method for assignment problems," *Naval Research Logistics Quarterly*, vol. 3, no. 4, pp. 253–258, 1956.

[23] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *Journal of Research of the National Bureau of Standards B*, vol. 69, no. 125-130, pp. 55–56, 1965.

[24] G. Karypis and V. Kumar, "METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997, retrieved from the University Digital Conservancy.

[25] Karypis, George, Kumar, and Vipin, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, p. 359–392, 1998.

[26] R. Ben Hur, N. Wald, N. Talati, and S. Kvatinsky, "Simple magic: Synthesis and in-memory mapping of logic execution for memristor-aided logic," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 225–232.

[27] M. Soeken, S. Shirinzadeh, P.-E. Gaillardon, L. G. Amarú, R. Drechsler, and G. De Micheli, "An MIG-based compiler for programmable logic-in-memory architectures," in *ACM/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[28] M. R. H. Rashed, S. K. Jha, and R. Ewetz, "Logic synthesis for digital in-memory computing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022.

[29] M. R. H. Rashed, S. Thijssen, S. Jha, and R. Ewetz, "LOGIC: Logic synthesis for digital in-memory computing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 30, no. 2, 2025.

[30] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IEEE/ACM International Workshop on Logic Synthesis (IWLS)*, 2015.

[31] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, and G. Marakkalage, Dewmini Sudara De Micheli, "The EPFL logic synthesis libraries," 2022, arXiv:1805.05121v3.

Rongliang Fu received his Ph.D. in Computer Science and Engineering from The Chinese University of Hong Kong in January 2026, following an M.S. from the University of Chinese Academy of Sciences in June 2021 and a B.S. in Software Engineering from Northwestern Polytechnical University in June 2018. He has authored over 30 papers across major journals (IEEE TC and IEEE TCAD) and conferences(DAC, DATE, ICCAD, etc.). His research spans electronic design automation and EDA for superconducting electronics.

Ran Zhang received his B.S. degree from Northeastern University, Shenyang, in 2023. He is currently a master's student at the University of Chinese Academy of Sciences (UCAS), under the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include electronic design automation and computer architecture.

Libo Shen received his B.S. degree in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2021 and his MS degree in computer technology from the University of Chinese Academy of Sciences, Beijing, China, in 2024. He is currently a Ph.D. student at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include electronic design automation and computer architecture.

Wei Xuan received his Ph.D degree in computer science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2023. He is currently a Postdoctoral fellow in the Department of Electronic and Computer Engineering at The Hong Kong University of Science and Technology and a member of the AI Chip Center for Emerging Smart Systems (ACCESS) under InnoHK. His research focuses on computer architecture and machine learning security.

Ning Lin is a Postdoctoral Research Fellow at the University of Hong Kong. He earned his doctorate from the Institute of Computing Technology, Chinese Academy of Sciences (Beijing) in 2022. Dr. Lin has authored over 40 publications including Nature Computational Science, Nature Communications, Science Advances, IEEE TCAD, DAC, ICCAD. He serves as a reviewer for leading journals and conferences such as IEEE TCAD, NeurIPS, ICLR, ICML, and AAAI. Dr. Lin's current research centers on hardware security for AI accelerator.

Junying Huang received her Ph.D. degree in microelectronics and solid-state electronics from the University of Chinese Academy of Sciences in 2016. Currently she is an associate professor with the Department of High-throughput Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include superconductive RSFQ logic, computer architecture, electronic design automation, and hardware security.

Bei Yu (M'15-SM'22) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served as TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He received eleven Best Paper Awards from ICCAD 2024 & 2021 & 2013, IEEE TSM 2022, DATE 2022, ASPDAC 2021 & 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, nine ICCAD/ISPD contest awards, IEEE CEDA Ernest S. Kuh Early Career Award in 2021, DAC Under-40 Innovator Award in 2024, and Hong Kong RGC Research Fellowship Scheme (RFS) Award in 2024.

Tsung-Yi Ho (F'24) is a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK). He received his Ph.D. in Electrical Engineering from National Taiwan University in 2005. His research interests include several areas of computing and emerging technologies, especially in the design automation of microfluidic biochips. He was a recipient of the Best Paper Award at the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2015. Currently, he serves as the VP Conferences of IEEE CEDA, and the Executive Committee of ASP-DAC and ICCAD. He is a Distinguished Member of ACM and a Fellow of IEEE.