Received 10 August 2024, accepted 29 August 2024, date of publication 2 September 2024, date of current version 13 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3453503

RESEARCH ARTICLE

Integer Linear Programming-Based Simultaneous Scheduling and Binding for SiLago Framework

DHILLESWARARAO PUDI[1], SHIVAM MALVIYA[1], SRINIVAS BOPPU[1], (Member, IEEE), YU YANG[2], (Member, IEEE), AHMED HEMANI[2], (Senior Member, IEEE), AND LINGA REDDY CENKERAMADDI[3], (Senior Member, IEEE)

[1]School of Electrical Sciences, Indian Institute of Technology at Bhubaneswar, Bhubaneswar 752050, India
[2]Division of Electronics and Embedded Systems, KTH Royal Institute of Technology, 114 28 Stockholm, Sweden
[3]Department of ICT, University of Agder, 4630 Grimstad, Norway

Corresponding author: Linga Reddy Cenkeramaddi (linga.cenkeramaddi@uia.no)

This work was supported in part by the Indo-Norwegian Collaboration in Autonomous Cyber-Physical Systems (INCAPS) through the International Partnerships for Excellent Education, Research and Innovation (INTPART) Program from the Research Council of Norway under Project 287918.

ABSTRACT Coarse-Grained Reconfigurable Array (CGRA) architectures are potential high-performance and power-efficient platforms. However, mapping applications efficiently on CGRA, which includes scheduling and binding operations on functional units and variables on registers, is a daunting problem. SiLago is a recently developed VLSI design framework comprising two large-scale reconfigurable fabrics: Dynamically Reconfigurable Resource Array (DRRA) and Distributed Memory Architecture (DiMArch). It uses the Vesyla compiler to map applications on these fabrics. The present version of Vesyla executes binding and scheduling sequentially, with binding first, followed by scheduling. In this paper, we proposed an Integer Linear Programming (ILP)-based exact method to solve scheduling and binding simultaneously that delivers better solutions while mapping applications on these fabrics. The proposed ILP combines two objective functions, one for scheduling and one for binding, and both of these objective functions are coupled with weightage factors α and β so that the user can have the flexibility to prioritize either scheduling or binding or both based on the requirements. We determined the binding and execution time of image processing tasks and various routines of the Basic Linear Algebraic Subprogram (BLAS) using the proposed ILP for multiple combinations of weightage factors. Furthermore, a comparison analysis has been conducted to compare the latency and power dissipation of several benchmarks between the existing and proposed approaches. The experimental results demonstrate that the proposed method exhibits a substantial reduction in power consumption and latency compared to the existing method.

INDEX TERMS Coarse-grain reconfigurable architecture, dynamically reconfigurable resource array, distributed memory architecture, integer linear programming, high-level synthesis, scheduling, binding.

I. INTRODUCTION

In recent years, a new class of accelerators called *Coarse-Grained Reconfigurable Arrays (CGRA)* has emerged, with applications primarily in embedded systems [1]. The computationally intensive sections of an application can be executed on these scalable, highly parallel platforms. CGRAs are an attractive trade-off between custom ASICs (*Application Specific Integrated Circuits*) and FPGAs (*Field Programmable Gate Arrays*) since they provide power efficiency on par with non-programmable hardware accelerators while yet being programmable [2], [3], [4]. CGRAs are especially appealing when tailored to specific areas where computational requirements are known, yet programmability is required to meet evolving technical specifications. References [5] and [6] provide a comprehensive review of CGRAs.

A *High-Level Synthesis (HLS)* framework is required to map applications on CGRAs. The process of mapping

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

an application onto a CGRA involves determining how to schedule and bind the operations of the application to the resources of CGRA in a way that considers the control and data dependencies and any other architectural constraints. Scheduling aims to minimize the time or control steps needed for an application, considering hardware constraints. Binding assigns operations and variables to resources while minimizing their usage. Traditionally, scheduling and binding were handled one after the other.

Simultaneous scheduling and binding in HLS are critical due to their significant impact on the performance and efficiency of the hardware. The simultaneous approach ensures that these decisions are made cohesively, optimizing resource utilization, reducing latency, and improving overall performance. Efficient scheduling and binding can minimize execution cycles and balance the workload across available resources, leading to more efficient hardware designs. The complexity of simultaneous scheduling and binding arises from the intricate interdependencies between these two tasks. Scheduling an operation for a specific time slot affects which resources are available for binding, and vice versa. This creates a highly combinatorial problem space, making it challenging to find an optimal solution using straightforward methods. Both scheduling and binding are individually NP-hard problems [7], [8], and their combination further adds to the complexity. This necessitates the use of optimization techniques such as *Integer Linear Programming (ILP)* and heuristics to find feasible solutions within reasonable time frames. Timing and area constraints further aggravate the complexity of the problem. Effective simultaneous scheduling and binding must account for these constraints to ensure that the hardware meets performance specifications while minimizing area and power consumption. The use of ILP for simultaneous scheduling and binding effectively addresses these challenges by formulating the problem as a set of linear equations and inequalities, allowing for precise control over the optimization process and yielding high-quality solutions.

This paper focuses on *SiLago*, an end-to-end synthesis framework [9], [10]. The SiLago framework takes into account two CGRA fabrics: (a) *Dynamically Reconfigurable Resource Array (DRRA)* [11], (b) *Distributed Memory Architecture (DiMArch)* [12], and employs *Vesyla* [13], [14] compiler to map ML and DSP applications onto both DRRA and DiMArch. The current version of Vesyla performs binding first, then scheduling [13]. The sequential approach to binding and scheduling has led to suboptimal resource utilization because decisions made during binding do not account for the dynamic requirements of scheduling. Therefore, in this paper, we opted to simultaneously perform scheduling and binding to achieve better results.

The main contributions of this paper are:

1) We propose an Integer Linear Programming (ILP) approach for the Simultaneous Scheduling and Binding (SSB) problems.
2) The proposed ILP has multiple objectives and includes weightage factors for scheduling and binding.
3) We determine execution time and binding for image processing tasks and various BLAS functions implemented on DRRA and DiMArch fabrics.
4) We show through our experiments that the proposed ILP provides better results compared to the existing method [13].

We want to emphasize that, although the ILP technique is well-known, our contributions in this paper focus on adapting the ILP method to our specific architecture. This adaptation involves developing an appropriate objective function and formulating constraints tailored to our needs. Our primary goal is to perform binding and scheduling simultaneously within the SiLago framework, thereby improving the underlying hardware's performance and increasing designer productivity.

The remaining sections of the paper are structured in the following manner: A concise overview of the SiLago framework, encompassing the DRRA, DiMArch, and Vesyla compiler are presented in Section II. The related work is discussed in Section III, and the proposed ILP formulation for simultaneous scheduling and the binding problem is discussed in detail in Section IV. Results are discussed in Section V. In Section VI, conclusions and future works are presented.

II. BACKGROUND

A quick introduction to the Vesyla compiler, DRRA, and DiMArch fabrics is given in this section. DRRA is a coarse-grained fabric for spatial computing that consists of control, arithmetic/logic, distributed storage, and connection resources. The DRRA fabric consists of DRRA cells arranged in a grid fashion, with two rows and many columns. Each cell is connected to both a vertical and horizontal bus via switch boxes. The DRRA fabric is capable of being expanded in size and can be adjusted to specific dimensions. The DRRA cell consists of a Data-Path Unit (DPU), a Register File (RF), a Sequencer (SEQ), and Switch Boxes (SB), as depicted in Fig. 1. Each DRRA cell is distinguished by its coordinate data. For instance, the top-right DRRA cell in Fig. 1, located in the seventh column and second row, is designated as a DRRA cell [1, 6].

The DPU functions as the primary arithmetic unit of the DRRA fabric, with four 16-bit input ports and two 16-bit output ports [11]. The RF refers to the distributed local storage of the DRRA fabric. It contains two read/write ports that can transfer 16 bits at a time. Each port is equipped with its own Address Generation Unit (AGU) to generate addresses [11], [15]. The RF is equipped with a 256-bit bidirectional interface that is exclusively used for transferring data between the RF and the SRAM macros. The RF's depth and breadth are parameterized, with default settings of 64 and 16, respectively. The DPU, switchboxes, and AGUs of the register file are dynamically configured at run-time by the sequencer (SEQ), the control unit of each DRRA [11], [16]. The sequencer governs the transmission of data between DiMArch and the DRRA fabric through a circuit-switched data Network-on-Chip (NoC) and a packet-switched high bandwidth configuration and control NoC. The DRRA cells engage in communication using a sliding window mechanism. The sliding window enables a cell to establish communication with a limited group of neighboring cells, demonstrating localized connectivity [17]. The size of the sliding window is a parameter that is determined during the design phase. In this study, we have selected a sliding window with a size of two. This means that each DRRA cell can communicate with other DRRA cells within a range of two columns on both sides, resulting in a sliding range of five columns. The switch box establishes connections between the outputs of the RFile and DPU and the inputs of the RFile and DPU of the DRRA cells located within the sliding window [17]. DiMArch mostly consists of SRAM macros, specifically mBanks, which are typically 2KB in size. The SRAM macro's dimensions, both in terms of depth and breadth, are parameterized. The default settings for the depth and width are 64 and 256-bit, respectively. Every SRAM macro is provided with both a read and a write port, which additionally feature Address Generation Units (AGUs) identical to those mentioned above for Register Files (RFs).

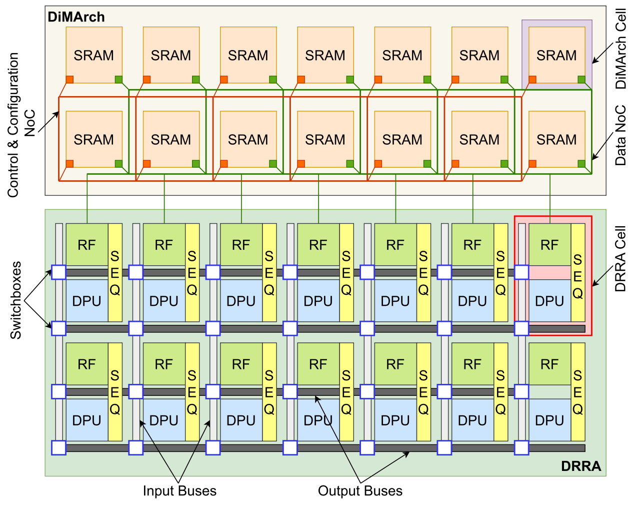


FIGURE 1. DRRA and DiMArch architecture, adopted from [18].

As previously indicated, Vesyla is utilized within the SiLago framework to facilitate the mapping of applications onto DRRA and DiMArch. Vesyla accepts input in the form of Matlab-style code with pragmas and produces the SiLago block layout, DRRA configware, and the cost of the synthesized algorithm [13]. Once an input source code is read into Vesyla, a Control Address and Data Flow Graph (CADFG), which is used as an intermediate representation, is used to capture all the information of the input program [19]. In this work, numerous parameters for the proposed ILP are derived from the CADFG. The layout generated by Vesyla includes the specified number of DRRA and DiMArch cells, known as SiLago blocks. These blocks are composed by abutment and do not require physical synthesis, as each block is hardened, effectively eliminating the need for logic and physical synthesis in the SiLago design flow. However, these SiLago blocks can be hardened using standard synthesis tools. A key limitation of Vesyla is that it specifically targets the DRRA+DiMArch architecture. However, because the SiLago CGRA platform shares similarities with other CGRA platforms, the techniques used in Vesyla could potentially be adapted for use with those platforms, although with some necessary modifications. Currently, there are no other tools available that can synthesize designs for the DRRA+DiMArch architecture. The SiLago framework is carefully designed to maintain composability across functional, electrical, and cost estimation models, making it a robust solution for large-scale designs.

The following sequence of steps occurs when a streaming application is mapped onto the DRRA and DiMArch: a) input streams are populated into DiMArch, which are subsequently copied to RFs by SEQ, b) DPUs retrieve the input streams from RFs and execute arithmetic operations, then store the results/output streams back into the RFs, and c) the output streams from the RF are transferred to the DiMArch by the SEQ. The streaming applications on DRRA and DiMArch utilize DiMArch SRAM macros to store their input and output streams, which are often known as SRAM variables. The operands used in the calculations are stored in register file variables, which consistently retrieve data from SRAM variables.

Over the past few years, several applications have been implemented on DRRA and DiMArch fabrics. The authors in [20] implemented a self-organizing map on DRRA for rapid identification of bacterial genomes. In [15], the Fast Fourier Transform (FFT) was mapped onto DRRA fabric. The implementation of the mixed radix FFT on DRRA fabric is discussed in [21]. In [22], [23], and [24], the authors explored image processing operations such as image averaging, 2D convolution, and Sobel edge detection on DRRA and DiMArch fabrics. DRRA and DiMArch fabrics have significant potential to accelerate machine learning, artificial intelligence, and digital signal processing algorithms, with many more implementations expected in the near future.

III. RELATED WORK

Scheduling and binding are essential for mapping applications on CGRA architectures. This process has two main approaches, depending on how the scheduling and binding are performed. The first approach involves solving binding and scheduling sequentially with heuristic [25], [26], [27] or exact methods [28], [29]. In [27], the edge-centric modulo scheduling heuristic is employed for scheduling. In [26], the authors have used Iterative Modulo Scheduling (IMS) [30] algorithm for scheduling and the binding problem is tackled by combining a routing heuristic from FPGA synthesis and a Simulated Annealing (SA) algorithm for placement. The method presented in [25] uses heuristic-based binding and scheduling to address a simplified problem, followed by a quantum-inspired evolutionary algorithm to refine the initial solution. This approach yields high-quality solutions but has a low convergence rate. In [28] and [29], the authors applied exact and heuristic methods to solve binding and scheduling problems sequentially. Scheduling is implicitly done by altering the DFG and integrating timing and architectural constraints. Levi's approach [31] is used to perform the binding, which identifies the common sub-graph between the temporally enlarged CGRA and the updated DFG.

The second solution addresses the binding and scheduling problems concurrently. The ILP-based approaches are employed in [32] and [33] to determine the optimal solutions for binding and scheduling. The ILP model is utilized in [34] to simultaneously perform the functional unit, register, and interconnect binding with the scheduling. In [35], the authors proposed a heuristic-enhanced scheduling algorithm to solve scheduling problems for distributed two-level control (D2LC) systems like DRRA fabric. The proposed algorithm is scalable and has been validated by a large number of experiments with varied problem sizes.

In [25], [26], [27], [28], and [29], the authors present methods that solve scheduling and binding in a sequentially. In contrast, our paper addresses scheduling and binding simultaneously. Although the authors in [32], [33], and [34] use ILP for simultaneous scheduling and binding, their approaches are not suited for the SiLago framework due to its unique characteristics. These methods derive ILP parameters from data flow graphs (DFG), while our work utilizes CADFG, which differs significantly from DFG. Furthermore, in [35], the authors focus solely on list scheduling without incorporating binding.

In recent years, numerous frameworks have emerged to facilitate the mapping of applications onto CGRA. In [36], the authors introduced HyCUBE which maps application kernels to CGRA through modulo scheduling. An ILP-based architecture-independent approach is presented in [37] for application mapping on CGRA. Chin et al. introduced CGRA-ME in [2], a unified framework for application mapping on CGRA, offering adaptable scheduling, mapping, placement, and routing tools. This framework is incorporated with the popular LLVM compiler [38] to facilitate the mapping of optimized C-language benchmarks onto specific CGRAs. The authors in [39] introduced HiMap, a fast and scalable CGRA mapping approach designed for multidimensional kernels on larger CGRAs. In [40], the authors introduced a compiler for CGRAs leveraging the OpenMP programming model. The REVAMP framework is presented in [41] for application mapping on heterogeneous CGRA architectures.

The existing CGRA frameworks are restricted to generic CGRA templates, they cannot accommodate the features of DRRA fabric, such as address generation units, sliding windows, and heterogeneous processing units. Consequently, these generic CGRA compilers cannot handle the DRRA fabric. On the other hand, the Vesyla compiler is specifically designed for DRRA and can effectively accommodate all these functionalities.

In our previous work [19], we introduced methods to automate binding in Vesyla, which was previously done manually using pragmas. We used a list scheduling-based approach for functional unit binding and an ILP-based method for register binding. Furthermore, scheduling and binding were performed sequentially, with binding preceding scheduling. Performing binding and scheduling sequentially has led to inefficient resource utilization, as binding fails to account for the dynamic requirements of scheduling. Therefore in this paper, we have implemented a simultaneous approach to scheduling and binding to achieve better results. Our objective is to address a key challenge: Given an application in Matlab-style code, efficiently schedule all operations to minimize overall execution time while determining optimal assignments of operations to DPUs, register file variables to register files, and SRAM variables to SRAM macros. This approach aims to minimize interconnect length between register files, SRAM macros, and DPUs. Notably, prior research has not tackled simultaneous scheduling and binding on this specific target architecture.

IV. METHODOLOGY

The Simultaneous Scheduling and Binding (SSB) problem is an NP-complete problem [42]. Integer linear programming (ILP) is a standard optimization method. This section presents an ILP-based approach for solving the SSB problem on a target architecture that includes DRRA and DiMArch fabrics.

A. INTEGER LINEAR PROGRAMMING (ILP) FORMULATION

The proposed ILP provides the scheduling information of various DPU operations, binding of the SRAM variables, register file variables, and DPU operations. To improve the understanding of our ILP formulation, we begin by introducing the notations employed in our formulation. The parameters utilized in the proposed ILP are enumerated in Table 1. Table 2 provides an overview of the variables employed in the proposed ILP. The objective function and constraints used in this ILP are presented below.

1) OBJECTIVE FUNCTION:

The ILP formulation encompasses two distinct objectives: the first aims at reducing the number of clock cycles required for processing a particular application, while the second objective is to minimize the data transfer interconnect length. In this scenario, the interconnect length refers to the distance between the DPU and RF, or between the RF and SRAM macro for RF binding and SRAM binding, respectively. Shorter interconnect lengths result in lower dynamic power dissipation. The interconnect length is determined by calculating the number of hops, which is the difference between the coordinates of the register file

TABLE 1. Parameters and their definition.

Parameter	Definition
N_s	No. of SRAM variables in CADFG
N_r	No. of RF variables in CADFG
N_{op}	No. of DPU operations in CADFG
N_{dpu}	No. of DPUs in DRRA
N_{dc}	No. of DRRA columns
N_{dic}	No. of DiMArch columns
N_{dr}	No. of DRRA rows
N_{dir}	No. of DiMArch rows
N_{rfile}	No. of RFs in DRRA
N_{sram}	No. of SRAM macros in DiMArch
D_{rfile}	Depth of the RF
D_{sram}	Depth of the SRAM macro
W_{rfile}	Width of the RF
W_{sram}	Width of the SRAM macro
N_{cyc}	A good upper bound for the number of clock cycles needed to schedule all operations
N_{wr}	No. of write ports in the RF and SRAM macro
N_{rd}	No. of read ports in the RF and SRAM macro
L_{sw}	Length of the sliding window
L^r	Length of the RF variable
L^s	Length of the SRAM variable
d	Execution time of DPU operations
A	Represents register file variables associated with each DPU operation
	A_{ij}=1 if register file variable j is associated with operation i
D	Represents the dependencies between DPU operations
	D_{ij}=1 if operation j depends on operation i
DR	Represents the number of data transfers between the DPU operations and register file variables
	$DR_{ij} = n$ if n data elements are transferred between register file variable j and DPU operation i
P	Represents the presence of register file variables across various DPU operations
	P_{ij}=1 if register file variable j is alive during the execution of operation i
R	Represents the register file variables to be used in each DPU operation
	R_{ij}=1 if register file variable j is used in operation i
W	Represents the register file variables to be defined in each DPU operation
	W_{ij}=1 if register file variable j defined at operation i
S	Represents the association of register file variables with the SRAM variables
	S_{ij}=1 if there is a data transfer between register file variable j and SRAM variable i
SR	Represents the number of data transfers between the SRAM and register file variables
	$SR_{ij} = n$ if n data elements are transferred between register file variable j and SRAM variable i
α, β	Weightage factors of scheduling and binding, respectively

variable, SRAM variables, and DPU operations, as illustrated in Eq. (1), as shown at the bottom of the page. In Eq. (1), α and β represent the weightage factors for scheduling and binding, respectively.

$$\min \Bigg[\alpha \times \left(T + \sum_{i=1}^{N_{op}} T_i^{st} \right) + \beta \times \left(\sum_{i=1}^{N_{op}} \sum_{j=1}^{N_r} DR_{ij} \times \left(\mid B_{i1}^{dpu} - B_{j1}^{rfile} \mid + \mid B_{i2}^{dpu} - B_{j2}^{rfile} \mid \right) + \sum_{i=1}^{N_s} \sum_{j=1}^{N_r} SR_{ij} \times \left(\mid (B_{i1}^{sram} + N_{dr} - B_{j1}^{rfile}) \mid + \mid B_{i2}^{sram} - B_{j2}^{rfile} \mid \right) \right) \Bigg] \quad (1)$$

TABLE 2. Variables and their definitions.

Variable	Definition
T^{st}	Start time of DPU operations in CADFG
T^{end}	End time of DPU operations in CADFG
T	Total execution time
o^{st}	$o^{st}_{ijk} = 1$ represents operation i at time step j starts execution in DPU k
o^{end}	$o^{end}_{ijk} = 1$ represents operation i at time step j ends execution in DPU k
o^{P}	$o^{P}_{ij} = 1$ represents operation i present at time step j
r^{P}	$r^{P}_{ij} = 1$ represents register file variable i present at time step j
x	$x_{ijk} = 1$ if at time step j RF variable i is mapped to RF k
y	$y_{ijk} = 1$ if at time step j SRAM variable i is mapped to SRAM macro k
B^{rfile}	Register file binding
B^{dpu}	DPU binding
B^{sram}	SRAM binding

2) CONSTRAINTS

Eqs. (2), and (3) ensures that each DPU operation has to be executed exactly once and mapped on exactly one DPU. Eqs. (4), and (5) define the start and end times of the DPU operations. Eq. (6) guarantees that the end time of each DPU operation depends on its start time and execution time. Eq. (7) describes precedence relations and assures that each operation's start time is bounded by its predecessors' completion times. Eq. (8) ensures that the total execution time is bounded by the execution time of each operation. Eq. (9) assures that each DPU operation starts and ends at the same DPU. Eq. (10) ensures that each DPU performs at most one operation at a time. Eq. (11) represents the existence of DPU operations over time, capturing the temporal aspect of these operations. This temporal representation is vital for minimizing latency in the scheduling and binding process. Eqs. (12), and (13) represent the DPU binding for various DPU operations. Specifically, these equations provide the coordinate information (i.e., row and column) of the DPU to which each operation is mapped. Eq. (12) denotes the row information, while Eq. (13) specifies the column information of the assigned DPU. The DPU binding relies on the temporal distribution of DPU operations and the availability of DPUs, as determined by N_{dr}. Each DPU operation is allocated to an available DPU within DRRA.

$$\forall i \in [1, N_{op}] : \sum_{j=1}^{N_{cyc}} \sum_{k=1}^{N_{dpu}} o^{st}_{ijk} = 1 \tag{2}$$

$$\forall i \in [1, N_{op}] : \sum_{j=1}^{N_{cyc}} \sum_{k=1}^{N_{dpu}} o^{end}_{ijk} = 1 \tag{3}$$

$$\forall i \in [1, N_{op}] : T^{st}_i = \sum_{j=1}^{N_{cyc}} (j \times \sum_{k=1}^{N_{dpu}} o^{st}_{ijk}) \tag{4}$$

$$\forall i \in [1, N_{op}] : T^{end}_i = \sum_{j=1}^{N_{cyc}} (j \times \sum_{k=1}^{N_{dpu}} o^{end}_{ijk}) \tag{5}$$

$$\forall i \in [1, N_{op}] : T^{end}_i = d_i + T^{st}_i - 1 \tag{6}$$

$$\forall i, j \in [1, N_{op}] : T^{st}_j \geq D_{ij} \times (1 + T^{end}_i) \tag{7}$$

$$\forall i \in [1, N_{op}] : T^{end}_i \leq T \tag{8}$$

$$\forall k \in [1, N_{dpu}] : \forall i \in [1, N_{op}] : \sum_{j=1}^{N_{cyc}} (o^{st}_{ijk} - o^{end}_{ijk}) = 0 \tag{9}$$

$$\forall k \in [1, N_{dpu}] : \forall j \in [1, N_{cyc}] : \sum_{i=1}^{N_{op}} (o^{st}_{ijk} - o^{end}_{ijk}) \leq 0 \tag{10}$$

$$\forall i \in [1, N_{op}] : \forall j \in [1, N_{cyc}] : o^{P}_{ij} = \sum_{k=1}^{N_{dpu}} \sum_{l=1}^{j} (o^{st}_{ilk} - o^{end}_{ilk}) + \sum_{k=1}^{N_{dpu}} o^{end}_{ijk} \tag{11}$$

$$\forall i \in [1, N_{op}] : \forall j \in [1, N_{cyc}] : B^{dpu}_{i1} = \left(\sum_{k=1}^{N_{dpu}} o^{st}_{ijk} \times (k-1) \right) \% N_{dr} \tag{12}$$

$$\forall i \in [1, N_{op}] : \forall j \in [1, N_{cyc}] : B^{dpu}_{i2} = \frac{\left(\sum_{k=1}^{N_{dpu}} o^{st}_{ijk} \times (k-1) \right)}{N_{dr}} \tag{13}$$

Eq. (14) assures that every RF variable is associated with exactly one register file in DRRA. Eq. (15) represents the existence of the register file variables over time. Due to the limited depth of RF, the total size/length of multiple variables stored in it cannot go beyond its depth, which is ensured by Eq. (16). The limited number of read and write ports in RF limits the number of variables that can be simultaneously written to or read from it. Eqs. (17), and (18) ensure that the number of variables read from or written to the RF corresponds to the available number of read and write ports, respectively. An RF can seamlessly communicate with other DPUs and RFs in other DRRA cells that are present within a sliding window spanning five columns. Eqs. (19), and (20) ensure this local connectivity of RF. Eqs. (21), and (22) represent the RF binding. Eq. (21) denotes the row information, while Eq. (22) specifies the column information of the assigned RF.

$$\forall i \in [1, N_r] : \forall j \in [1, N_{cyc}] : \sum_{k=1}^{N_{rfile}} x_{ijk} = 1 \tag{14}$$

$$\forall i \in [1, N_r] : \forall j \in [1, N_{cyc}] : r^P_{ij} = \sum_{k=1}^{N_{op}} (P_{ki} \times o^P_{kj}) \tag{15}$$

$$\forall k \in [1, N_{rfile}] : \forall j \in [1, N_{cyc}] : \sum_{i=1}^{N_r} x_{ijk} \times L^r_i \times r^P_{ij} \leq D_{rfile} \tag{16}$$

$$\forall k \in [1, N_{rfile}] : \forall j \in [1, N_{cyc}] : \sum_{i=1}^{N_r} (x_{ijk} \times \sum_{m=1}^{N_{op}} (R_{mi} \times o^P_{mj})) \leq N_{rd} \tag{17}$$

$$\forall k \in [1, N_{rfile}] : \forall j \in [1, N_{cyc}] : \sum_{i=1}^{N_r} (x_{ijk} \times \sum_{m=1}^{N_{op}} (W_{mi} \times o^P_{mj})) \leq N_{wr} \tag{18}$$

$$\forall i \in [1, N_r] : \forall j \in [1, N_{op}] : A_{ji} \times \mid B^{dpu}_{j1} - B^{rfile}_{i1} \mid \leq L_{sw} \tag{19}$$

$$\forall i \in [1, N_r] : \forall j \in [1, N_{op}] : A_{ji} \times \mid B^{dpu}_{j2} - B^{rfile}_{i2} \mid \leq L_{sw} \tag{20}$$

$$\forall \in [1, N_r] : \forall j \in [1, N_{cyc}] : B^{rfile}_{i1} = \left(\sum_{k=1}^{N_{rfile}} x_{ijk} \times (k-1) \right) \% N_{dr} \tag{21}$$

$$\forall \in [1, N_r] : \forall j \in [1, N_{cyc}] : B^{rfile}_{i2} = \frac{\left(\sum_{k=1}^{N_{rfile}} x_{ijk} \times (k-1) \right)}{N_{dr}} \tag{22}$$

Eq. (23) guarantees that every SRAM variable in DiMArch is associated with exactly one SRAM macro. SRAM, which can also hold multiple input/output variables, is limited in size, similar to RF. Therefore, Eq. (24) ensures that the total combined length of variables stored in the SRAM cannot go beyond its depth. Eqs. (25), and (26) guarantees that every SRAM macro will establish communication with RFs located within the sliding window, which spans five columns. Eqs. (27), and (28) represent the SRAM binding. Eq. (27) denotes the row information, while Eq. (28) specifies the column information of the assigned SRAM macro in DiMArch.

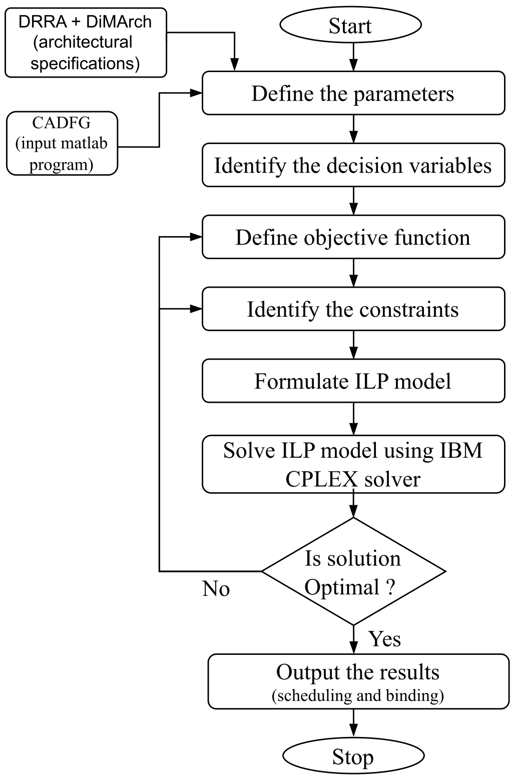

DRRA + DiMArch (architectural specifications)
CADFG (input matlab program)
Start
Define the parameters
Identify the decision variables
Define objective function
Identify the constraints
Formulate ILP model
Solve ILP model using IBM CPLEX solver
Is solution Optimal ?
No
Yes
Output the results (scheduling and binding)
Stop

FIGURE 2. Flow chart of proposed method.

$$\forall i \in [1, N_s] : \forall j \in [1, N_{cyc}] : \sum_{k=1}^{N_{sram}} y_{ijk} = 1 \tag{23}$$

$$\forall k \in [1, N_{sram}] : \forall j \in [1, N_{cyc}] : \sum_{i=1}^{N_s} y_{ijk} \times L^s_i \leq \frac{D_{sram} \times W_{sram}}{W_{rfile}} \tag{24}$$

$$\forall i \in [1, N_s] : \forall j \in [1, N_r] : S_{ij} \times \mid B^{sram}_{i1} - B^{rfile}_{j1} \mid \leq L_{sw} \tag{25}$$

$$\forall i \in [1, N_s] : \forall j \in [1, N_r] : S_{ij} \times \mid B^{sram}_{i2} - B^{rfile}_{j2} \mid \leq L_{sw} \tag{26}$$

$$\forall \in [1, N_s] : \forall j \in [1, N_{cyc}] : B^{sram}_{i1} = \left(\sum_{k=1}^{N_{sram}} y_{ijk} \times (k-1) \right) \% N_{dr} \tag{27}$$

$$\forall \in [1, N_s] : \forall j \in [1, N_{cyc}] : B^{sram}_{i2} = \frac{\left(\sum_{k=1}^{N_{sram}} y_{ijk} \times (k-1) \right)}{N_{dr}} \tag{28}$$

We implemented the ILP formulations in Vesyla using IBM CPLEX solver [43]. Although the number of variables and inequalities in an ILP formulation grows exponentially as the complexity of the problem grows, the motivation of this paper is to achieve an improved solution.

Fig. 2 depicts the flowchart for the proposed method. The process begins with defining the parameters used in the ILP formulation. These parameters, detailed in Table 1, are derived from the DRRA+DiMArch architectural specifications and CADFG. The next step involves identifying and defining the decision variables as integers, as listed in Table 2. The objective function is formulated once the variables are defined, as shown in Eq. (1). Constraints are then formulated and categorized into four types: scheduling constraints (Eq. (2)-Eq. (8)), DPU binding constraints (Eq. (9)-Eq. (13)), register file binding constraints (Eq. (14)-Eq. (22)), and SRAM binding constraints (Eq. (23)-Eq. (28)). After defining the parameters, decision variables, objective function, and constraints, the ILP model is formulated and solved using the IBM CPLEX solver. The final step is to verify if the ILP provides an optimal solution. If not, the objective function and constraints are revised. If the solution is optimal, the scheduling and binding results are used further to generate the configware for DRRA and DiMArch cells.

The proposed ILP formulation incorporates weightage factors α and β for scheduling and binding, respectively. By defining these factors, users can optimize execution time and data transfer interconnect length. In contrast, the sequential execution of scheduling and binding in the existing SiLago framework results in performance degradation and limits effective resource utilization. This limitation is due to the underlying architecture's interdependence between scheduling and binding processes during application mapping. Moreover, the current version of Vesyla requires manual binding, whereas the proposed approach automates this process. This automation allows users with limited architectural knowledge to effectively utilize the framework, significantly reducing the design space and saving time compared to manual binding. By allowing users to select appropriate values for α and β based on the specific application, the proposed method significantly improves the performance of the underlying architecture. Another notable advantage of the proposed ILP approach is its adaptability to architectural changes. For instance, if the depth and width of RF and SRAM macros increase in the future, it is sufficient to adjust the corresponding ILP parameters (i.e., D_{rfile}, W_{rfile}, D_{sram}, and W_{sram}) to reflect these changes. This flexibility ensures that the proposed approach remains effective and relevant despite architectural modifications.

V. RESULTS AND DISCUSSIONS

We conducted the experiments on a set of Basic Linear Algebra Subprograms (BLAS), namely, matrix-vector multiplication (*blas3gemv*), matrix-matrix multiplication (*blas3gemm, blas3axbxc*), and matrix-matrix multiplication and addition (*blas3mmadd*), listed in Table 3. In addition, we considered the BLAS function using matrices and vectors of dimension 4. Specifically, all the benchmarks presented in Table 3 consist of matrices of size 4 × 4 and vectors of size 1 × 4. Table 3 also includes the total number of DPU operations, RF variables, and SRAM variables in the BLAS routines. Table 4 lists the utilized parameters in all the experiments.

TABLE 3. Benchmarks.

BLAS function	Operation	# DPU Operations	# Register file variables	# SRAM variables
blas2gemv	y = aAx + by A : matrix x, y : vectors a, b : scalar coefficients	2	4	4
blas3gemm	Y = aAX + bY A, X, Y : matrices a, b : scalar coefficients	2	4	4
blas3axbxc	C = aXYC C, X, Y : matrices a : scalar coefficient	2	4	4
blas3mmadd	C = aXY + bYX + C C, X, Y : matrices a, b : scalar coefficients	3	4	4

TABLE 4. Parameters and their values.

Parameter	D_{rfile}	W_{rfile}	D_{sram}	W_{sram}	N_{rd}	N_{wr}	L_{sw}
Value	64	16	64	256	2	2	2

The proposed ILP is very flexible since it permits user-defined weightage factors α and β to determine the relative importance of the two objectives, execution time, and data transfer connection length. Here, minimization in data transfer interconnect length determines the dynamic power dissipation. Shorter interconnect lengths lead to less dynamic power dissipation. Therefore, the execution time, binding, and power value will vary depending on the values α and β. In this work, we considered three combinations of α and β, i.e., $\alpha = 1, \beta = 0$ (*Case 1*), $\alpha = 0, \beta = 1$ (*Case 2*), $\alpha = \alpha_{opt}, and \beta = \beta_{opt}$ (*Case 3*). The values of α and β depend on operations and variables present in input Matlab code, execution times of operations, and data movement between DPUs, register files, and SRAM macros. The generalized values of α and β for which the proposed ILP yields better results are shown in Eq. (29) and Eq. (30). Here, x_1 and x_2 are objective function values for *Case 1* and *Case 2*, respectively. By using Eq. (29) and Eq. (30), we have calculated the values of α and β that yield better results for BLAS functions as listed in Table 5.

$$\alpha_{opt} = \frac{x_2}{x_1 + x_2} \tag{29}$$

$$\beta_{opt} = \frac{x_1}{x_1 + x_2} \tag{30}$$

In *Case 1*, weightage is given to the scheduling objective, which reduces the total execution time, but no weightage is assigned to the binding objective. Table 6 shows the number of clock cycles required to execute all the operations, DPU, register file, and SRAM binding of various BLAS functions for $\alpha = 1$ and $\beta = 0$. From Table 6, it is obvious that the *blas3gemv* function containing two DPU operations is mapped to DPU in the DRRA cell [0, 0] in the DRRA

TABLE 5. The optimal values of α and β for BLAS functions.

BLAS function	α_{opt}	β_{opt}
blas2gemv	0.546	0.454
blas3gemm	0.472	0.518
blas3axbxc	0.402	0.598
blas3madd	0.474	0.526

TABLE 6. Scheduling and binding of BLAS functions for $\alpha = 1$, $\beta = 0$.

BLAS Function	# cycles	DPU Binding	Register file Binding	SRAM Binding
blas2gemv	28	[0, 0] [0, 0]	[1, 0] [1, 1] [1, 0] [1, 1]	[0, 0] [0, 1] [0, 0] [0, 1]
blas3gemm	84	[0, 0] [0, 0]	[1, 1] [1, 1] [1, 0] [1, 0]	[0, 1] [0, 1] [0, 0] [0, 0]
blas3axbxc	82	[1, 1] [0, 0]	[1, 1] [1, 1] [1, 0] [1, 0]	[0, 1] [0, 1] [0, 0] [0, 0]
blas3madd	101	[0, 0] [0, 0] [0, 0]	[1, 0] [1, 1] [1, 0] [1, 0]	[0, 0] [0, 1] [0, 1] [0, 0]

TABLE 7. Scheduling and binding of BLAS functions for $\alpha = 0$, $\beta = 1$.

BLAS Function	# cycles	DPU Binding	Register file Binding	SRAM Binding
blas2gemv	40	[0, 0] [0, 0]	[0, 0] [0, 0] [1, 0] [0, 0]	[0, 0] [0, 0] [0, 0] [0, 0]
blas3gemm	100	[0, 0] [0, 0]	[0, 0] [0, 0] [1, 0] [1, 0]	[0, 0] [0, 0] [0, 0] [0, 0]
blas3axbxc	100	[0, 0] [0, 0]	[0, 0] [0, 0] [0, 0] [1, 0]	[0, 0] [0, 0] [0, 0] [0, 0]
blas3madd	120	[0, 0] [0, 0] [0, 0]	[1, 0] [0, 0] [0, 0] [1, 0]	[0, 0] [0, 0] [0, 0] [0, 0]

TABLE 8. Scheduling and binding of BLAS functions for $\alpha = \alpha_{opt}$, $\beta = \beta_{opt}$.

BLAS Function	# cycles	DPU Binding	Register file Binding	SRAM Binding
blas2gemv	28	[0, 0] [0, 0]	[0, 0] [0, 0] [1, 0] [0, 0]	[0, 0] [0, 0] [0, 0] [0, 0]
blas3gemm	84	[0, 0] [0, 0]	[0, 0] [0, 0] [1, 0] [1, 0]	[0, 0] [0, 0] [0, 0] [0, 0]
blas3axbxc	82	[0, 0] [0, 0]	[0, 0] [0, 0] [0, 0] [1, 0]	[0, 0] [0, 0] [0, 0] [0, 0]
blas3madd	101	[0, 0] [0, 0] [0, 0]	[1, 0] [0, 0] [0, 0] [1, 0]	[0, 0] [0, 0] [0, 0] [0, 0]

fabric. The register file variables mapped over two columns of DRRA lead to an increase in interconnect length. All the SRAM variables are mapped to the SRAM macro [0, 0].

In *Case 2*, a weighted binding objective is considered, which minimizes data transfer interconnect length, but no weightage is given to the scheduling objective. In this case, the proposed ILP yields better binding results, i.e., it provides the appropriate mappings for the operations and variables. Table 7 shows the number of clock cycles required to execute all the operations, DPU, register file, and SRAM binding of various BLAS functions for $\alpha = 0$ and $\beta = 1$. Table 7 clearly shows that the blas3gemv function, which includes two DPU operations, maps to the DPU in the DRRA cell [0, 0] within the DRRA fabric. The register file variables mapped over the first column of DRRA lead to a minimization in interconnect length compared to *Case 1*. All the SRAM variables are mapped to the SRAM macro [0, 0]. Since the scheduling objective is not weighted, the execution time increases compared to Case 1.

In *Case 3*, equal weightage is given to both the scheduling and binding objectives. In this case, the proposed ILP yields better scheduling and binding results, with the minimum number of clock cycles needed to process a given application, and provides appropriate mappings for the operations and variables. This results in less execution time and low power dissipation. Table 8 shows the number of clock cycles required to execute all the operations, DPU, RF, and SRAM binding of aforementioned BLAS functions for $\alpha = \alpha_{opt}$ and $\beta = \beta_{opt}$.

Fig. 3 shows the comparison of execution times of the BLAS functions for different combinations of weightage factors α and β. From Fig. 3, it is observed that the proposed ILP yields an optimum number of clock cycles for Case 1 and Case 3 compared to Case 2. Fig. 4 shows the comparison of power values of various BLAS functions for different combinations of weightage factors α and β. From Fig. 4, it is observed that the proposed ILP yields low power values for Case 2 and Case 3 compared to Case 1. From Fig. 3 and Fig. 4, it is evident that the proposed ILP provides better scheduling and binding results for *Case 3*, where equal weightage is assigned to both scheduling and binding objectives. We employed Synopsys Design Compiler and Cadence Innovus for logic synthesis and physical synthesis, respectively. Mentor's Questasim was utilized for simulating

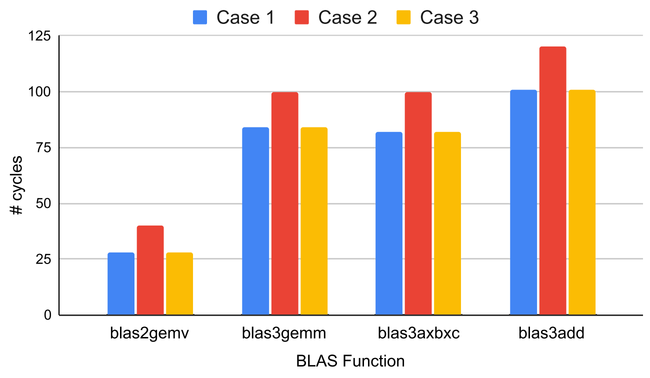


FIGURE 3. The execution time of BLAS functions for different combinations of weightage factors α and β.

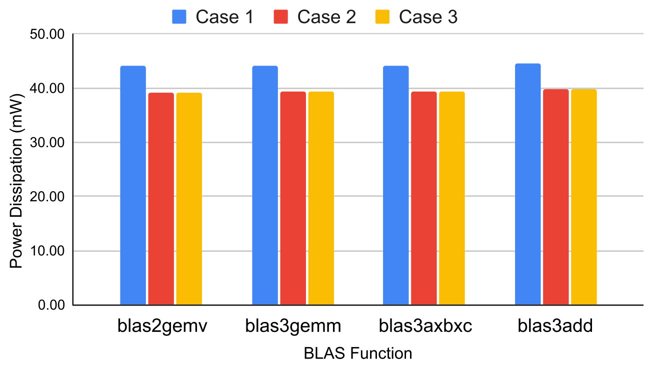


FIGURE 4. The power dissipation of BLAS functions for different combinations of weightage factors α and β.

the switching activity, which is subsequently stored in Value Change Dump (VCD) format. This VCD file was used later in power calculations using Innovus. Furthermore, the DRRA fabric was realized using a 28 *nm* technology node and limited by a clock of 200 MHz.

In addition, we performed experiments that involved 2D convolution, a range of image-processing tasks, imagine averaging, and Sobel edge detection. The outcomes of scheduling and binding for each application and the compilation time for each function are listed in Table 9 including the optimized values of α and β. In Table 9, "*2Dconv_m*" represents the implementation of 2D convolution on the "m"-column DRRA. Similarly, "*imgavg_n*" and "*Sobel_n*" denote the image averaging and Sobel edge detection for a window of size $n \times n$. For 2D convolution, we have considered the kernel of size 3×3. For all experiments, we have considered an input matrix of size 16×16, stride of one, and zero padding. Table 9 shows that the compilation time increases with larger window sizes for Sobel edge detection and image averaging. This is due to the increased computational complexity associated with larger window sizes, leading to increased decision variables and constraints in the ILP formulation. Similarly, for 2D convolution implemented on a two-column DRRA, the ILP solver requires more time than a one-column DRRA. This trend emerges due to the enlarged DRRA fabric size, which in turn increases the number of decision variables and constraints in the ILP formulation, resulting in a longer optimization process.

Power dissipation and latency for both the proposed and existing method are compared, see Fig. 5. The existing method [13] performs binding first, then scheduling, with binding manually provided through pragmas [13]. In the existing method, the list scheduling algorithm [35] is used to schedule the DPU operations. The proposed approach not only performs scheduling and binding simultaneously but also automates the binding in Vesyla. Based on Fig. 5, it can be noted that the proposed method yields significantly lower power values compared to the existing method. Moreover, the proposed method demonstrates superior latency performance compared to the existing method.

We calculated the time the ILP solver took to solve the proposed ILP for the aforementioned BLAS routines and image processing tasks, as shown in Fig. 6. We can conclude from the observations in Fig. 6 that the proposed approach yields solutions within a reasonable time. The proposed approach can easily accommodate architectural modifications and necessitates only minor adjustments to input parameters to work with larger DRRA fabrics.

Fig. 7 shows the compilation times for the aforementioned tasks of image processing implemented on various sizes of DRRA fabric. It is to be noted that due to coupling issues with interconnect wires and power consumption, the number of rows is fixed at two in DRRA fabric. As the DRRA fabric size increases, the compilation times also increase as shown and evident from Fig. 7. The reason for this increase is that a bigger fabric size creates additional constraints and inequalities in the ILP formulation, which leads to the solver requiring more time to find the optimal solution. Our proposed methods offer efficient solutions for 16×16 matrices, even when dealing with huge DRRA fabrics.

The compilation times of various benchmark applications subject to different input matrix/vector dimensions is shown in Fig. 8. In this case, we have considered the DRRA size of 2×2. According to Fig. 8, there is a significant exponential growth in compilation time as the size of the matrix increases. This occurs because increasing the dimensions of the matrix leads to a greater number of constraints, inequalities, and design possibilities. When the dimensions of the matrix are smaller, we have noticed only a slight increase in the time it takes to compile. However, for bigger matrix sizes, this increase is significant. As an example, the amount of time it takes to compile the *blas2gemv* BLAS subroutine for a 16×16 input matrix is 39 seconds, while it increases to 1541 seconds for a 64×64 input matrix. Hence, for smaller and medium-sized matrix dimensions, our proposed approach effectively delivers solutions within a reasonable time.

The proposed ILP approach is tailored to the SiLago framework to solve scheduling and binding problems. Consequently, we have not validated it on current state-of-the-art CGRA architectures. Nonetheless, our method can be adapted to other CGRA architectures with minimal modifica-

TABLE 9. Scheduling, binding, and compilation time for image processing operations.

Function	Operation	α_{opt}	β_{opt}	# DPU operations	# Register file variables	# SRAM variables	DPU binding	Register file binding	SRAM binding	# cycles	Compilation time (sec)
2DConv_1	2D convolution	0.2076	0.7924	2	5	3	DPU [0,0] DPU [1,0]	RFILE [0,0] RFILE [1,0] RFILE [0,0] RFILE [1,0] RFILE [1,0]	MEM [0,0] MEM [0,0] MEM [0,0]	1078	97
2DConv_2	2D convolution	0.2974	0.7026	4	7	3	DPU [0,0] DPU [1,0] DPU [0,1] DPU [1,1]	RFILE [0,0] RFILE [1,0] RFILE [1,0] RFILE [0,1] RFILE [0,1] RFILE [1,1] RFILE [1,1]	MEM [0,0] MEM [0,0] MEM [0,0]	539	139
Imgavg_3	Image averaging	0.4128	0.5872	1	2	1	DPU [0,0]	RFILE [0,0] RFILE [0,0]	MEM [0,0]	882	124
Imgavg_5	Image averaging	0.4824	0.5176	1	3	1	DPU [0,0]	RFILE [0,0] RFILE [1,0] RFILE [1,0]	MEM [0,0]	1800	145
Imgavg_7	Image averaging	0.5291	0.4709	1	3	1	DPU [0,0]	RFILE [0,0] RFILE [1,0] RFILE [0,1]	MEM [0,0]	2450	161
Sobel_3	Sobel edge detection	0.6915	0.3085	4	7	4	DPU [0,0] DPU [1,0] DPU [0,0] DPU [1,0]	RFILE [0,0] RFILE [1,0] RFILE [0,0] RFILE [1,0] RFILE [0,1] RFILE [0,1] RFILE [1,1]	MEM [0,0] MEM [0,0] MEM [0,0] MEM [0,0]	1176	202
Sobel_5	Sobel edge detection	0.5826	0.4174	4	9	4	DPU [0,0] DPU [1,0] DPU [0,0] DPU [1,0]	RFILE [1,0] RFILE [0,1] RFILE [0,1] RFILE [1,1] RFILE [0,0] RFILE [0,0] RFILE [0,2] RFILE [0,2] RFILE [1,2]	MEM [0,0] MEM [0,0] MEM [0,0] MEM [0,0]	2016	286

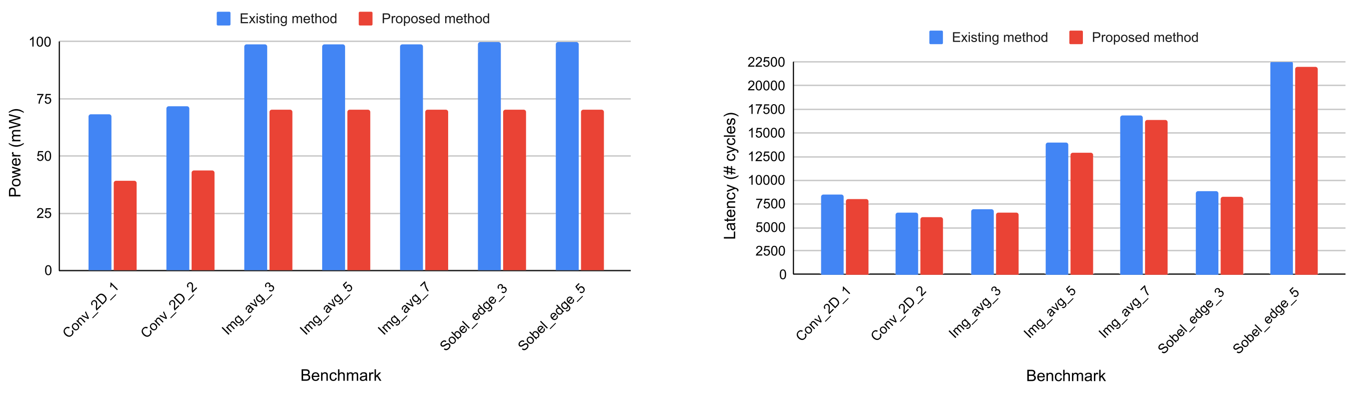


FIGURE 5. Power dissipation and latency comparison for both existing [13] and proposed methods.

tions, enabling customization to the unique architectural constraints of each target system. For example, DRRA features a register file with a capacity of 64 bytes, while other CGRA architectures might have a register file capacity of 128 bytes. To apply our approach to these architectures, the ILP model can be easily modified by adjusting the register file width and depth parameters to match the particular architectural constraints. Another key aspect of DRRA is its sliding window communication, which allows unrestricted access to data within a sliding window. Other CGRA architectures

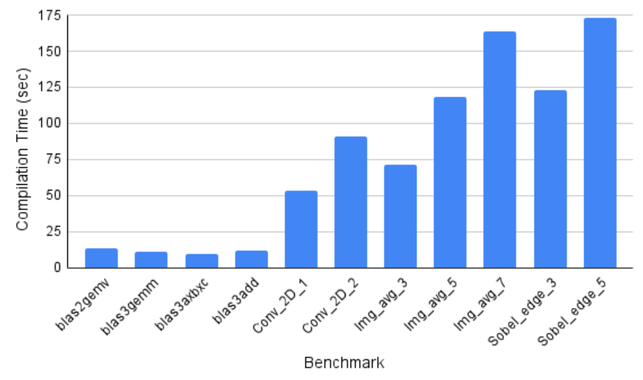


FIGURE 6. Compilation times for various benchmarks.

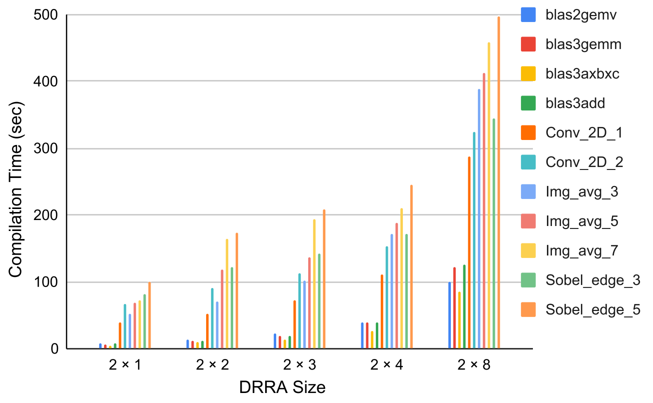


FIGURE 7. Comparison of compilation times for various DRRA sizes.

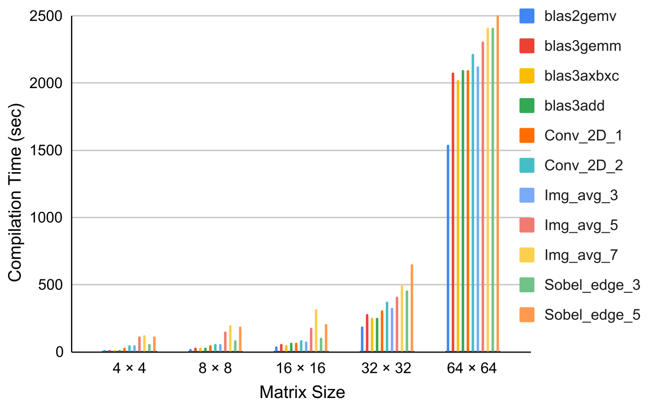


FIGURE 8. Comparison of compilation times for different matrix dimensions.

may have different sliding window lengths or may lack this feature entirely, necessitating further adjustments to the ILP model. Apart from register file capacity and sliding window communication, other parameters, such as memory storage capacity and the number of read/write ports, may differ across various CGRA architectures. The proposed method can effectively adapt by recognizing and incorporating these differences. This flexibility ensures that the proposed method can be utilized to solve scheduling and binding problems across a wide range of CGRA architectures.

VI. CONCLUSION

In this paper, we have proposed an Integer Linear Programming (ILP) based exact method to solve the simultaneous scheduling and binding problems for the target architecture, comprising DRRA and DiMArch fabrics. The proposed ILP considers two distinct objective functions: the first objective aims to reduce the number of clock cycles required to run a particular application, while the second objective strives to minimize data transfer interconnect length. We have calculated the execution times and binding of various BLAS routines and image processing tasks using the proposed ILP for different scheduling and binding weightage factor combinations. The proposed ILP formulation yields better results for both scheduling and binding while providing equal weightage to both the scheduling and binding objectives. The proposed approach is observed to have significantly less power dissipation and latency than the existing method. We demonstrate that the ILP solver can solve the proposed ILP in a reasonable time for benchmarks. Only the SiLago framework, consisting of DRRA and DiMArch fabrics, is meant to use the Proposed ILP. For this reason, we have not validated the proposed ILP on state-of-the-art CGRA architectures.

REFERENCES

[1] Y. Kim and R. N. Mahapatra, "Hierarchical reconfigurable computing arrays for efficient CGRA-based embedded systems," in *Proc. 46th ACM/IEEE Design Autom. Conf.*, Jul. 2009, pp. 826–831.

[2] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *Proc. IEEE 28th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2017, pp. 184–189.

[3] Y.-H. Lai, H. Rong, S. Zheng, W. Zhang, X. Cui, Y. Jia, J. Wang, B. Sullivan, Z. Zhang, Y. Liang, Y. Zhang, J. Cong, N. George, J. Alvarez, C. Hughes, and P. Dubey, "SuSy: A programming model for productive construction of high-performance systolic arrays on FPGAs," in *Proc. 39th IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[4] J. Wang, L. Guo, and J. Cong, "AutoSA: A polyhedral compiler for high-performance systolic arrays on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2021, pp. 93–104.

[5] M. Baron, "Trends in the use of re-configurable platforms," in *Proc. 41st Annu. Design Autom. Conf.* New York, NY, USA: Association for Computing Machinery, Jun. 2004, p. 415, doi: 10.1145/996566.996685.

[6] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in *Proc. Design, Autom. Test Europe. Conf. Exhib.*, 2001, pp. 642–649.

[7] A. Zhang, Y. Chen, L. Chen, and G. Chen, "On the NP-hardness of scheduling with time restrictions," *Discrete Optim.*, vol. 28, pp. 54–62, May 2018.

[8] N. Min-Allah, M. B. Qureshi, S. Alrashed, and O. F. Rana, "Cost efficient resource allocation for real-time tasks in embedded systems," *Sustain. Cities Soc.*, vol. 48, Jul. 2019, Art. no. 101523. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670718300635

[9] S. M. A. H. Jafri, N. Farahini, and A. Hemani, "SiLago-CoG: Coarse-grained grid-based design for near tape-out power estimation accuracy at high level," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2017, pp. 25–31.

[10] A. Hemani, N. Farahini, S. M. A. H. Jafri, H. Sohofi, S. Li, and K. Paul, "The SiLago solution: Architecture and design methods for a heterogeneous dark silicon aware coarse grain reconfigurable fabric," in *The Dark Side of Silicon*. Cham, Switzerland: Springer, 2017, pp. 47–94, doi: 10.1007/978-3-319-31596-6_3.

[11] M. A. Shami, "Dynamically reconfigurable resource array," Ph.D. dissertation, KTH Roy. Inst. Technol., Stockholm, Sweden, 2012.

[12] M. A. Tajammul, S. M. A. H. Jafri, A. Hemani, and P. Ellervee, "TransMem: A memory architecture to support dynamic remapping and parallelism in low power high performance CGRAs," in *Proc. 26th Int. Workshop Power Timing Modeling, Optim. Simul. (PATMOS)*, Sep. 2016, pp. 92–99.

[13] Y. Yang and A. Hemani, "Vesyla-II: An algorithm library development tool for synchoros VLSI design style," 2022, *arXiv:2206.07984*.

[14] O. Malik, A. Hemani, and M. A. Shami, "A library development framework for a coarse grain reconfigurable architecture," in *Proc. 24th Int. Conf. VLSI Design*, Jan. 2011, pp. 153–158.

[15] M. A. Shami, M. A. Tajammul, and A. Hemani, "Configurable FFT processor using dynamically reconfigurable resource arrays," *J. Signal Process. Syst.*, vol. 91, no. 5, pp. 459–473, May 2019.

[16] M. A. Shami and A. Hemani, "An improved self-reconfigurable interconnection scheme for a coarse grain reconfigurable architecture," in *Proc. NORCHIP*, Nov. 2010, pp. 1–6.

[17] M. A. Shami and A. Hemani, "Partially reconfigurable interconnection network for dynamically reprogrammable resource array," in *Proc. IEEE 8th Int. Conf. ASIC*, Oct. 2009, pp. 122–125.

[18] Y. Yang, D. Stathis, P. Sharma, K. Paul, A. Hemani, M. Grabherr, and R. Ahmad, "RiBoSOM: Rapid bacterial genome identification using self-organizing map implemented on the synchoros SiLago platform," in *Proc. 18th Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simul.*, Jul. 2018, pp. 105–114.

[19] D. Pudi, U. Tiwari, S. Boppu, Y. Yang, and A. Hemani, "Automating functional unit and register binding for synchoros CGRA platform," *Des. Automat. Embedded Syst.*, pp. 1–32, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s10617-024-09286-y

[20] D. Stathis, Y. Yang, S. Tewari, A. Hemani, K. Paul, M. Grabherr, and R. Ahmad, "Approximate computing applied to bacterial genome identification using self-organizing maps," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 560–567.

[21] R. Kallapu, D. Stathis, S. Boppu, and A. Hemani, "DRRA-based reconfigurable architecture for mixed-radix FFT," in *Proc. 36th Int. Conf. VLSI Design 22nd Int. Conf. Embedded Syst. (VLSID)*, Jan. 2023, pp. 25–30.

[22] P. Dhilleswararao, R. Ryansh, S. Boppu, Y. Yang, and A. Hemani, "Efficient implementation of 2-D convolution on DRRA and DiMArch architectures," in *Proc. 13th Int. Symp. Highly Efficient Accel. Reconfigurable Technol.*, Jun. 2023, pp. 86–92.

[23] D. Pudi, V. Goudu, S. Boppu, R. Ratnu, and A. Hemani, "Implementation of image averaging on DRRA and DiMArch architectures," in *Proc. 36th SBC/SBMicro/IEEE/ACM Symp. Integr. Circuits Syst. Design (SBCCI)*, Aug. 2023, pp. 1–6.

[24] D. Pudi, R. Ryansh, V. Goudu, S. Boppu, and A. Hemani, "Implementation of Sobel edge detection on DRRA and DiMArch architectures," in *Proc. 26th Euromicro Conf. Digit. Syst. Design (DSD)*, Sep. 2023, pp. 16–23.

[25] G. Lee, K. Choi, and N. D. Dutt, "Mapping multi-domain applications onto coarse-grained reconfigurable architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 5, pp. 637–650, May 2011.

[26] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, "SPR: An architecture-adaptive CGRA mapping tool," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Feb. 2009, pp. 191–200.

[27] H. Park, K. Fan, S. Mahlke, T. Oh, H. Kim, and H.-S. Kim, "Edge-centric modulo scheduling for coarse-grained reconfigurable architectures," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2008, pp. 166–176.

[28] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *Proc. 49th Annu. Design Autom. Conf.*, Jun. 2012, pp. 1280–1287.

[29] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "REGIMap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf.*, May 2013, pp. 1–10.

[30] B. R. Rau, "Iterative module scheduling: An algorithm for software pipelining loops," in *Proc. 27th Annu. IEEE/ACM Int. Symp. Microarchit.*, Nov. 1994, pp. 63–74.

[31] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcolo*, vol. 9, no. 4, pp. 341–352, Dec. 1973.

[32] J. Brenner, J. Van Der Veen, S. Fekete, J. Filho, and W. Rosenstiel, "Optimal simultaneous scheduling, binding and routing for processor-like reconfigurable architectures," in *Proc. Int. Conf. Field Program. Log. Appl.*, 2006, pp. 1–6.

[33] E. Raffin, C. Wolinski, F. Charot, K. Kuchcinski, S. Guyetant, S. Chevobbe, and E. Casseau, "Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, Oct. 2010, pp. 168–175.

[34] C. H. Gebotys and M. I. Elmasry, "Optimal synthesis of high-performance architectures," *IEEE J. Solid-State Circuits*, vol. 27, no. 3, pp. 389–397, Mar. 1992.

[35] Y. Yang, A. Hemani, and K. Paul, "Scheduling persistent and fully cooperative instructions," in *Proc. 24th Euromicro Conf. Digit. Syst. Design (DSD)*, 2021, pp. 229–237.

[36] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[37] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to CGRA mapping," in *Proc. Annu. 55th Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[38] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, 2004, pp. 75–86.

[39] D. Wijerathne, Z. Li, A. Pathania, T. Mitra, and L. Thiele, "HiMap: Fast and scalable high-quality mapping on CGRA via hierarchical abstraction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3290–3303, Oct. 2022.

[40] T. Kojima, B. Adhi, C. Cortes, Y. Tan, and K. Sano, "An architecture- independent CGRA compiler enabling OpenMP applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2022, pp. 631–638.

[41] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "REVAMP: A systematic framework for heterogeneous CGRA realization," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Feb. 2022, pp. 918–932.

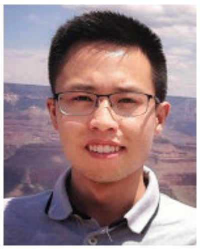
[42] D. S. Johnson and M. R. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.

[43] S. Nickel, C. Steinhardt, H. Schlenker, W. Burkart, and M. Reuter-Oppermann, *IBM ILOG CPLEX Optimization Studio*. Berlin, Germany: Springer, 2021, pp. 9–23, doi: 10.1007/978-3-662-62185-1_2.

DHILLESWARARAO PUDI received the M.Tech. degree in microelectronics and VLSI from the National Institute of Technology, Durgapur, India, in 2014, and the Ph.D. degree from Indian Institute of Technology Bhubaneswar, India, in 2024. He is currently an Assistant Professor with KL University, Hyderabad, India. His current research interests include programmable hardware accelerators, hardware/software co-design, compilers, and high-level synthesis.

SHIVAM MALVIYA received the B.Tech. degree in ECE from Indian Institute of Technology Bhubaneswar, India, in 2022. He is currently a Compiler Software Engineer with MathWorks, Hyderabad, India. His research interests include compilers and software application development.

SRINIVAS BOPPU (Member, IEEE) received the dual International M.Sc. degree in IC design from Nanyang Technological University (NTU), Singapore, and the Technical University of Munich (TUM), Germany, and the Ph.D. degree in the chair for hardware/software co-design from the Department of Computer Science, University of Erlangen–Nuremberg, Germany, in 2015. He has been an Assistant Professor with the School of Electrical Sciences, IIT Bhubaneswar, since October 2017. Before moving to India, he was a Senior Consultant with Infineon Technologies, Munich, Germany. He was with Freescale Semiconductors, India, and STMicroelectronics, as a Physical Design Engineer, before pursuing the Ph.D. degree. He has more than 15 years of experience in both academia and industry in the field of the VLSI domain. His research interests include high-level synthesis, programmable hardware accelerators, compilers, scheduling and mapping approaches, low-power VLSI design, SoC design, and design automation of integrated circuits. He was awarded the full scholarship by Infineon Technologies Asia Pacific Pte., Ltd., for the M.Sc. degree in IC design jointly offered by NTU and TUM.


YU YANG (Member, IEEE) received the bachelor's degree from Sichuan University, China, the master's degree from the Politecnico di Torino, Italy, and the Ph.D. degree from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2022. He is currently a Postdoctoral Researcher with the KTH Royal Institute of Technology. His research interest includes high-level and system-level synthesis flow for synchoros VLSI design.

AHMED HEMANI (Senior Member, IEEE) has been a Professor with the Electronics Department, School of EECS, KTH Royal Institute of Technology, Stockholm, Sweden, since 2006. His doctoral thesis on HLS was the basis for one of CADENCE's first commercial HLS products. Later, he contributed to latency-insensitive design style with research on GALS and GRLS. He also pioneered the concept of NOCs. His current research interests include massively parallel architecture and design methods to achieve comparable ASIC performance. He has introduced the concept of synchoricity and driven the development of the SiLago as an experimental synchoros VLSI design platform. He is applying synchoros VLSI design to artificial neural networks, biologically plausible models of the brain and complex multimedia, and telecom applications.

LINGA REDDY CENKERAMADDI (Senior Member, IEEE) received the master's degree in electrical engineering from Indian Institute of Technology Delhi (IIT Delhi), New Delhi, India, in 2004, and the Ph.D. degree in electrical engineering from Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2011.

He was with Texas Instruments, on mixed-signal circuit design before joining the Ph.D. degree with NTNU. After finishing the Ph.D. degree, he was involved in radiation imaging for an atmosphere–space interaction monitor (ASIM mission to the International Space Station) with the University of Bergen, Bergen, Norway, from 2010 to 2012. He is currently the Leader of the Autonomous and Cyber-Physical Systems (ACPS) Research Group and a Professor with the University of Agder, Grimstad, Norway. He has co-authored over 210 research publications that have been published in prestigious international journals and standard conferences in the research areas of the Internet of Things (IoT), cyber-physical systems, autonomous systems, robotics and automation involving advanced sensor systems, computer vision, thermal imaging, LiDAR imaging, radar imaging, wireless sensor networks, smart electronic systems, advanced machine learning techniques, and connected autonomous systems, including drones/unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs), unmanned underwater systems (UUSs), 5G- (and beyond) enabled autonomous vehicles, and socio-technical systems, such as urban transportation systems, smart agriculture, and smart cities. He is also quite active in medical imaging. He is a member of ACM and a member of the editorial boards of various international journals and the technical program committees of several IEEE conferences. Several of his master's students won the Best Master Thesis Award in information and communication technology (ICT). He serves as a reviewer for several reputed international conferences and IEEE journals. He is the Principal Investigator and the Co-Principal Investigator of many research grants from the Norwegian Research Council.