ABSTRACT

Coarse-Grained Reconfigurable Architectures (CGRAs) are a promising solution to domain-specific applications for their energy efficiency and flexibility. To improve performance on CGRA, modulo scheduling is commonly adopted on Data Dependence Graph (DDG) of loops by minimizing the Initiation Interval (II) between adjacent loop iterations. The mapping process usually consists of scheduling and placement-and-routing (P&R). As existing approaches don’t fully and globally explore the routing strategies of the long dependencies in a DDG at the scheduling stage, the following P&R is prone to failure leading to performance loss. To this end, this paper proposes a routability-enhanced scheduling for CGRA mapping using Integer Linear Programming (ILP) formulation, where a global optimized scheduling could be found to improve the success rate of P&R. Experimental results show that our approach achieves $1.12 \times$ and $1.22 \times$ performance speedup, 28.7% and 50.2% compilation time reduction, as compared to 2 state-of-the-art heuristics.

INDEX TERMS

Coarse-grained reconfigurable architectures (CGRAs), modulo scheduling, routing strategy, placement and routing (P&R).

I. INTRODUCTION

With both energy efficiency and program flexibility, Coarse-Grained Reconfigurable Architectures (CGRAs) are becoming attractive alternatives for embedded systems. By deploying the configurable and parallel hardware resources efficiently, CGRA enables pervasive use of high energy-efficient solutions for a wide range of applications [1]–[3].

As shown in Fig.1, a CGRA [4]–[6] usually consists of a host controller, a data memory, a configuration memory and a Processing Element Array (PEA). The host controller controls the whole system and exchanges data with the PEA by the data memory, which is composed of multiple banks form such that PE columns can access the data memory in parallel. The configuration memory stores configuration contexts that can dynamically change the data path of the PEA. The PEA is usually composed of dozens of PEs connected by 2-D connecting networks. Each PE includes an ALU-like Function Unit (FU), a Local Register File (LRF), and an output register. With these basic resources in a CGRA, a long dependence in a Data Dependency Graph (DDG) can be routed by PE, LRF, or data memory [7] and it makes routing strategy exploration viable in mapping.

In order to improve loop execution performance on CGRA, modulo scheduling [8], [9] is widely used. Modulo scheduling tries to map a DDG extracted from a loop to the target CGRA, where Initiation Interval (II) between adjacent loop iterations is the main concern to increase the
performance of a loop. Usually, DDG mapping involves 3 basic steps: i) scheduling, ii) operator placement, and iii) value routing. As the connection topology among PEs is sparse, the Placement and Routing (P&R) are often done simultaneously for a better solution. Once the P&R step gets failed, rescheduling is usually performed to generate another scheduled DDG for P&R.

To route the long dependencies (that traversing more than 1 time steps) in a DDG, 3 basic routing strategies could be explored, including PE routing [10], register routing [11] and memory routing [7]. Some works [8], [12] integrate routing strategy exploration into the P&R step. Due to the difficult search in the huge space, it is unusable for all but the small DDGs. For fast compilation, most works [9]–[11], [13]–[15] integrate routing strategy exploration into the scheduling step. In BufAware mapping [9], as the first row shown in Fig. 2, the authors argue that initial scheduling exploration should be paid more attention for the purpose of higher performance and robust compilation, and therefore they comprehensively analyze and reschedule the DDG to help for improving the success rate. However, the scheduling exploration in [9] only generate one modified DDG and it will directly increase II once the P&R of this modified DDG fails. That is to say, rescheduling routing exploration is neglected in this work.

Most works [10], [11], [15], as the second row shown in Fig. 2, pay more attention to rescheduling routing exploration, where routing strategy exploration is performed on those unmapped nodes from a P&R failure. If all rescheduling schemes are failed, it will increase II to relax resource constraints and try again. However, only learning from unmapped nodes would be too late and may easily get stuck in a local minimum.

From the analysis above, we note that as routing strategies are not fully (at both initial scheduling and rescheduling stage) and globally (on the whole DDG rather than unmapped nodes) explored, the following P&R is prone to failure leading to performance loss. To this end, as the third row shown in Fig. 2, this paper proposes a Routability-Enhanced Scheduling for Mapping (RESMap) applications on CGRAs, supporting global routing explorations in both initial scheduling and rescheduling stages. Our main contributions are summarized as follows:

- We unify traditional operator scheduling and graph modification in an Integer Linear Programming (ILP) formulation. Besides basic time step adjustment, it also supports graph modification, such as node insertion and edge cutting. Therefore, various mainstream routing strategies, such as PE routing, register routing, memory routing and path sharing, and their combinations could be globally explored.
- We make routing explorations on both scheduling and rescheduling steps. By solving the ILP problem iteratively, we can make full exploration of routing strategy and generate optimized DDG for the P&R attempts, such that it may get successful P&R at the very beginning, reduce the number of scheduling-P&R iterations, and finally save compilation time.

The remainder of the paper is organized as follows: Section II gives background of CGRA mapping. Then, section III states the proposed method. Next, Section IV shows the experimental results. Finally, we conclude in Section V.

II. BACKGROUND

In this section, we first introduce the routing strategies of long dependence. Then, we illustrate modulo scheduling on CGRA with an example.

A. ROUTING STRATEGIES OF LONG DEPENDENCE

As shown in Fig. 3, the three types of routing strategies are illustrated with a 4-node DDG mapping on a $1 \times 2$ PEA time extended to 4 time steps, where the yellow cell and blue cell indicate FU and LRF, respectively.

1) ROUTING DATA VIA LRF

LRF-routing supported mapping methods allow routing long dependencies via local registers [11], [16]. Although this method saves more FUs, it also implies a strict constraint that the father node and child node of a long dependence must be placed on the same PE. As shown in Fig. 3(a) and (b), the long dependence $1 \rightarrow 4$ indicates that it will be delivered by local registers in $PE_0$. To satisfy the strong constraint, operator 4 can only be placed on $PE_0$, too.
PE-routing supported mapping methods [10], [16], [17] allow routing long dependencies via extra PEs by inserting routing nodes into DDG before P&R. As shown in Fig. 3(e) and (f), the long dependence of the input DDG is firstly cut off. Then, two memory accessing nodes, storing node (S) and loading node (L), are inserted. Finally, the new modified DDG is mapped. Without placement constraints introduced by long dependencies, operator 4 can be placed on either PE₀ or PE₁.

From the different routing methods, we find that although LRF routing does not need graph modification, it brings strong placement constraint to P&R. PE routing and memory routing could be friendly to P&R, but it involves complicated graph modification.

### III. ROUTABILITY-ENHANCED MAPPING METHOD

In this section, we first extend traditional scheduling to support global routing strategy exploration by formulating it as an ILP problem. Then, we solve the ILP problem with a simple and effective metric. Finally, we present the overall flow of DDG mapping based on the enhanced scheduling.

#### A. ILP FORMULATION

We use ILP to formulate the enhanced scheduling for two reasons: 1) As our enhanced scheduling involves graph modification, e.g., routing node insertion and edge cutting, traditional scheduling, such as list scheduling [18], force-directed scheduling [19], [20] or even SDC [21], cannot work in this problem. 2) ILP approach can formulate most problems into clear representations in mathematics and it is possible to find the optimal solution [22], [23]. Before ILP formulation, we first give some parameters which are illustrated with the DDGs with II = 4 in Fig. 5:

- **Maximal DDG latency (Tᵢ):** The maximal latency of the loop DDG. It could be greater than or equal to the length of the critical path (Tᵢ) of the DDG. In the case of having greater latency, adjacent operators in the critical path may be scheduled apart such that a better mapping solution may be achieved. For the DDG in Fig. 5(b), Tᵢ is set to 7 equal to the length of the critical path.
- **Routable operator set (R):** The operator in this set has at least one output edge that could be traversing more than one time step in some cases (long edge candidates), such that a routing node could be inserted. For the DDG in Fig. 5(b), as edges 1 → 8, 8 → 5, 8 → 9 and 9 → 7 are long edge candidates, their father operators 1, 8 and 9 are in R. We also note that the back edge 4 → 2 (loop carried dependence) is also a long edge candidate in the case of II = 4, as the dashed arrow shown in Fig. 5(b). Thus, operator 4 is also in R. Therefore, the R set for the DDG in Fig. 5(b) is {1, 4, 8, 9}.
- **Earliest time step for scheduling (Sᵢ):** The earliest time step that operator n can be scheduled which could be easily obtained with As-Soon-As-Possible (ASAP) [19] considering a given Tᵢ. As shown in Fig. 5(b), the earliest time steps of operator 1, 4, 8 and 9 are 0, 3, 1 and 2, respectively.

The TEC is shown in Fig. 4(d), where a node PEᵢ indicates the instance of PE i at time step j and each arrow indicates a direct hardware link among these PE instances. As long dependence indicates register routing implicitly in P&R and the local register can only be accessed by its corresponding FU, the endpoint operator of a long dependence must be placed on the same PE at different time steps. As shown in Fig. 4(d), operator 5 and 4 of the long dependence 5 → 4 are both placed on PE₁ for register routing. As the target time steps for nodes in the modified DDG are previously determined, P&R just needs to place each node on a PE instance at the corresponding time step and make all those placed nodes be connected according to dependencies.
The last two binary variable sets define memory routing node scheduling and PE routing node insertion for an operator. ILP formulation. The first binary variable set defines operator we define 3 binary variable sets and 1 integer variable for\[ S. Mu et al.: Routability-Enhanced Scheduling for Application Mapping on CGRAs \]

**Latest time step for scheduling** ($L_n$): The latest time step that operator $n$ can be scheduled which could be easily obtained with As-Late-As-Possible (ALAP) [19] considering a given $T_j$. As shown in Fig. 5(b), the latest time steps of operator 1, 4, 8 and 9 are 0, 3 and 5, respectively.

**Latest time step for routing** ($L_m$): The latest time step that operator $n$’s routing node can be inserted which is equal to $\max_{n' \in OE_n} (L_{n'} - 1)$, where $OE_n$ indicates the set of sub-operator from operator $n$. In the case of $n \rightarrow n'$ is a back edge (recurrence dependence), $L_m$ is equal to $\max_{n' \in OE_n} (L_{n'} + 1)$ as its sub-operator is actually from the next iteration II steps latter. In Fig. 5(b), given operator 8 has two sub-operators 5 and 9 with $L_5 = 4$ and $L_9 = 5$, such that $L_8 = L_9 - 1 = 4$. Similarly, consider that $4 \rightarrow 2$ is back edge from operator 4, $L_2 = 1$ and $L_4 = 4$, such that $L_4 = 1 + 4 - 1 = 4$.

**Mobility** ($M_n$): The mobility range of operator $n$ that is equal to $[S_n, L_n]$. As shown in Fig. 5(b), $M_1$, $M_4$, $M_8$ and $M_9$ are $[0, 0]$, $[3, 3]$, $[1, 3]$ and $[2, 5]$, by white bars.

**Extended mobility** ($\tilde{M}_n$): This new defined mobility is extended from $M_n$ to include the time steps of operator’s routing node, which is equal to $[S_n, L_m]$. As shown in Fig. 5(b), $\tilde{M}_1$, $\tilde{M}_4$, $\tilde{M}_8$, $\tilde{M}_9$ and $\tilde{M}_5$ are $[0, 1]$, $[3, 4]$, $[1, 4]$ and $[2, 5]$, indicated by the white bars plus yellow bars.

**Edge set** ($E$): The dependence set in the original DDG.

**Memory accessing latency** ($T_m$): The minimal latency of memory routing, including data storing, data holding and data loading. In case that data storing and data loading can be performed in one time step, $T_m$ is 3.

**PEA size** ($N_p$): The number of PEs in the target PEA. We assume that the target PEA for the DDG in Fig. 5(a) is a $1 \times 2$ PEA and thus $N_p$ is 2 there.

**Variable Definition:** Based on the parameters above, we define 3 binary variable sets and 1 integer variable for ILP formulation. The first binary variable set defines operator scheduling and PE routing node insertion for an operator. The last two binary variable sets define memory routing node insertion. The integer variable defines the number of PE consumed. These variables are necessary for the formulation, and are elaborated as follows:

- $x_{n,i,j}, l \in [S_n, L_n], j \in [i, L_m]$; Operator $n$ is scheduled at time step $i$ and one of its PE routing node is inserted at time step $j$, where $j$ is not less than $i$. In the case of $j = i$, it means operator $n$ is scheduled at time step $i$. In the case of $j > i$, it indicates a PE routing node is inserted later than operator $n$’s scheduled time step $i$. As shown in Fig. 5(a), it presents all binary variables referring to node 8. As the $M_8$ and $\tilde{M}_8$ are 3 and 4, there are nine $x$-variables for node 8.

- $y_{n,i,j}, l \in [S_n, L_n], j \in [i + 1, L_m - T_m + 1]$; The value of operator $n$ at time step $i$ is stored to memory by adding a storing node at time step $j$. Following the condition, as shown in Fig. 5(a), only one $y$-variable ($y_{8,1,2}$) is generated for operator 8.

- $z_{n,i,j}, l \in [S_n, L_m], j \in [i + T_m, L_m]$; The value of operator $n$ at time step $i$, previously stored to memory, is loaded from memory by adding a loading node at time step $j$. Following the condition, as shown in Fig. 5(a), only one $z$-variable ($z_{8,1,4}$) is generated for operator 8.

- $n_{pe}$: The maximal number of PE used. Following this condition, the number of operators with the same modulo time steps should be less than or equal to $n_{pe}$.

According to the time step of scheduling, the ILP variables of operator $n$ can be classified into $M_n$ groups. As shown in Fig. 5(a), node 8 has 3 possible time steps and its variables are classified into 3 groups, $G_1$, $G_2$ and $G_3$. As these variables would build a huge solution space, we further reduce the space by the following constraints:

**C1:** Operator’s scheduled time step is unique: Similar to traditional scheduling, each operator should be only scheduled at one time step. This constraint can be represented as follow:

$$\sum_{i \in [S_n, L_n]} x_{n,i,i} = 1, \forall n \in R$$ (1)
**C2: Routing node exclusivity:** This constraint is a supplement for the first one. Once an operator is scheduled at a time step, routing nodes generated for the operator at other scheduled time steps should be eliminated. It can be represented as follows:

\[ x_{n_i,1} + x_{n_i,2} \leq 1, \quad \forall i_2 \neq i_1 \]
\[ x_{n_i,1} + x_{n_i,2} \leq 1, \quad \forall i_2 \neq i_1 \]
\[ x_{n_i,1} + x_{n_i,2} \leq 1, \quad \forall i_2 \neq i_1 \]

(2)

**C3: Storing and loading node concurrence:** As long dependence can be cut off and routed by data memory, the memory accessing nodes, storing and loading, should be of concurrence. It can be represented as follows:

\[ y_{n,i,j} - z_{n,i,j} = 0, \quad \forall j_1, j_2 \]

(3)

**C4: Functional Unit routing and memory routing exclusivity:** Once a pair of memory routing nodes are inserted for an operator, Functional Unit routing nodes should not be inserted during the latency of the memory routing. It can be represented as follows:

\[ y_{n,i,j} + x_{n,i,j} \leq 1, \quad \forall j_1, j_2 \]

(4)

**C5: Dependence constraint:** Each operator should be scheduled earlier than its sub-operator and the operator’s routing nodes should be inserted earlier than the child node that may have the latest scheduling time step. They can be represented as follows:

\[ i_1 \times x_{n_1,i_1,j_1} < \sum_{i_2 \in \{n_2 \mid n_2 \in E \}} (i_2 \times x_{n_2,i_2,j_2}), \quad \forall (n_1, n_2) \in E \]

(5)

\[ j_1 \times x_{n_1,i_1,j_1} < i_2 \times x_{n_2,i_2,j_2}, \quad n_2 = \{n(1,n) \in E \} S_n \]

(6)

As presented in Fig. 5(d), operator 8’s scheduling time step should be simultaneously earlier than its sub-operator 5 and 9. Meanwhile, operator 8’s routing nodes just needs satisfy only one constraint that earlier than sub-operator 9.

**C6: Functional Unit resource constraint:** At each time step, the total number of nodes, including operator and inserted routing nodes, should be no more than the number of Functional Units of the target CGRA. It can be represented as follows:

\[ \sum_{n} x_{n,i,j} + \sum_{n} y_{n,i,j} + \sum_{n} z_{n,i,j} \leq n_{pe}, \quad \forall (j \mod \Pi) = t, \quad \forall t \in [0, \Pi - 1] \]

(7)

**C7: Memory bank resource constraint:** At each time step, the total number of memory accessing nodes, including storing and loading nodes, should be no more than the number of banks (N_b) of the data memory. It can be represented as follows:

\[ \sum_{n} y_{n,i,j} + \sum_{n} z_{n,i,j} \leq N_b, \quad \forall (j \mod \Pi) = t, \quad \forall t \in [0, \Pi - 1] \]

(8)

**C8: Finding suboptimal solutions:** As one scheduling result cannot guarantee that the corresponding P&R would succeed, we need to generate multiple suboptimal solutions. As the work in [24], a suboptimal solution of ILP can be generated by adding extra constraints from obtained solutions. Then, the new ILP problem is solved again to find another solution. As our problem just contains 0-1 variables, we can use the binary cut method, which involves only one extra constraint and no additional variables. From an obtained solution S*, composed of variables set \{x_{n,i,j}\}, \{y_{n,i,j}\} and \{z_{n,i,j}\}, we can add an extra constraint as follows:

\[ \sum_{v \in B} v - \sum_{v \in A} v \leq k - 1 \]

(9)

where \( A = \{v|v = 0, \forall v \in S*\} \) and \( B = \{v|v = 1, \forall v \in S*\} \) are binary cut set from an obtained solution S*. By adding this constraint, we can find the k solutions with the same objective value.

**C9: Bound constraint:** As the suboptimal solution above will put more and more constraints into the ILP problem and make the ILP solver more and more complicated, the extra bound constraint is introduced to simplify the formulation once the objective for ILP is changed. In that cases, all the previously added constraints for getting suboptimal solutions could be removed. The bound constraint is presented as follows:

\[ O > O_{lb} \]

(10)

where O is the objective of ILP formulation that is introduced in the next subsection, and \( O_{lb} \) is the last value of objective once the objective is changed by ILP solver.

**B. MAPPING FLEXIBILITY EVALUATION**

As the maximal width of the folded DDG will affect the placement flexibility (how many PE positions a node could be placed) while the number of long dependencies will put strong constraint that source node and target node must be placed on the same PE, we will evaluate the enhanced scheduling from these two factors. The first factor can be easily presented by the ILP variable \( n_{pe} \). For the second factor, directly formulating the number of long dependence in scheduling, especially in graph modification supported scheduling, is of high cost and difficult to solve. Alternatively, we use the number of routing nodes inserted to indirectly evaluate the number of long dependencies. The more routing nodes are inserted, the fewer long dependencies it may have. Therefore, the second factor is presented as follows:

\[ n_{ins} = \sum_{n_e R, i \in [n_a, L_a], j \in [1, L_a]} \left( x_{n,i,j} + \alpha(y_{n,i,j} + z_{n,i,j}) \right) \]

(11)

where \( \alpha \) is the weight factor for memory routing nodes. As memory accessing nodes will cut off long dependencies and exclude more PE routing nodes inserted (Equation (4)), we give greater weights for memory accessing nodes. Empirically, we find that our method will generate good results when \( \alpha \) is set to 10.

**Objective Function:** Considering both factors above, we present the objective function of ILP in linear form as...
The overall mapping flow based on our enhanced scheduling follows:

$$\min_{n_{x_{ij,k}}, n_{x_{ijk}}, n_{r_{x_{ijk}}, r_{pe}}} O = \beta \cdot n_{pe} - n_{ins}$$  (12)

where $\beta$ is equal to the upper bound of $n_{ins}$ and it can be trivially calculated by traversing each long dependence in the original DDG. The first item of RHS indicates that minimizing $n_{pe}$ is of high priority as it could save more PEs and improve routability for every modulo time step of folded DDG. The second item of RHS indicates that maximizing $n_{ins}$, in the premise of keeping the maximal width of the folded DDG, is encouraged to reduce the number of long dependencies.

With the objective function and the 9 constraints, we can use sophisticated ILP solvers, such as PULP [25], to obtain one scheduling solution. Then, the new DDG could be easily generated by adding extra nodes or edges according to the values of the binary variables. By iteratively adding suboptimal constraint in Equation (9), a series of solution candidates could be generated. Therefore, our approach could make global routing strategy exploration at both scheduling and rescheduling stages.

### C. OVERALL FLOW

Fig. 6 presents the overall mapping flow based on our enhanced scheduling. 1) The mapping flow starts with an input DDG given its MII, where the interconnection problem is preprocessed with re-computation or routing [15]. 2) Then, we analyze the extended mobility ($\hat{M}$) for each operator in the DDG. 3) According to $\hat{M}$, we define all the variables referring to scheduling and routing. 4) Next, we construct all constraints listed above where constraint C8 and C9 are not included in the first round. 5) With the objective defined in Equation (12), we perform ILP solver [25] to find an optimal scheduling. If there is no solution found, we will increase II and goto step 4. Otherwise, we record the objective value $O_i$ in this $i^{th}$ iteration and further judge if it is changed or not. If yes, 6) we will update the extra bound constraint (C9) and 7) clear all previous suboptimal constraints (C8) and goto step 4. Otherwise, 8) we will generate the modified DDG according to determined decision variables and 9) perform P&R based on a clique searching algorithm [11]. If the P&R succeeds, the whole mapping is done and a valid mapping with II is obtained. Otherwise, 10) we will add a suboptimal constraint C8 based on the current solution and goto step 4.

From the flow, we note that the P&R is the most time-consuming stage. In case the P&R time is too long, an upper bound of running time is set which is usually several hours. Therefore, all things we have done in this work are trying to improve the capability of P&R from the perspective of DDG form involving operator scheduling and graph modification. With the routability enhanced scheduling, the number of scheduling-P&R iteration will be greatly reduced. As a result, it is helpful to find a mapping with less II and even less compilation time.

### IV. EXPERIMENTAL RESULTS

In order to evaluate the effectiveness of our approach, we conduct a series of experiments, including the performance in terms of II, the overall compilation time and its breakdown.

#### A. SETUP

1) **BENCHMARKS**

We conducted experiments by mapping 12 selected performance-critical loops from three benchmarks: 1) MiBench [26], 2) Spec2006 [27], and 3) PolyBench [28] suite for broad representation. In Table 1, we describe the DDGs converted from these loops in detail, where $N_{op}$, $N_{dep}$, $N_{M_{op}}$, $N_{M_{dep}}$, $D_{M_{max}}$, and $D_{M_{min}}$ indicate the number of operators, the number of dependencies, the number of operators with mobilities, the number of long dependencies, the maximal dependence length (i.e., the number of control steps traversed by a dependence) and the minimal dependence length, respectively.

2) **TARGET ARCHITECTURES**

The target architecture [16] is a $4 \times 4$ CGRA with mesh-plus routing style. In mesh-plus routing CGRA, FUs are connected in a mesh network, where each FU is connected to its...
immediate neighbors and its near neighbors with 1 hop. In our setup, each PE includes 1 FU and 2 local registers and the number of memory banks is 4.

3) BASELINES
We evaluated RESMap against other state-of-the-art approaches, including RAMP [15] and BufAware [9]. In our approach, the parameter $T_l$ is set to $T_c + 1$ (the length of the critical path plus 1) to make every operator in DDG have the opportunity of routing.

4) DEVICES AND SOFTWARE
Our experiments over three approaches are conducted on the same Linux workstation with an Intel Xeon 2.4 GHz CPU and 64GB memory. In order to compare the effectiveness of our approach with RAMP and BufAware, the upper bound of each scheduling-P&R iteration is set to 2 hours. If the running time of one iteration is beyond the upper bound, BufAware will directly increase II while RAMP and RESMap will perform rescheduling and try again.

B. RESMap GENERATES BETTER MAPPING
From Fig. 7, the first observation is that RESMap achieves lower II values than RAMP in 10 out of 12 kernels and BufAware in 6 out of 12 kernels. On average, our RESMap gets 17.8% and 10.5% less II than RAMP and BufAware. In another word, RESMap reaches up 1.12 × and 1.22 × performance speedup, as compared to BufAware and RAMP, respectively.

As compared to RAMP, the improvement is mainly from two aspects: 1) RAMP does not give the ideal initial scheduling before P&R. Consequently, the poor initial scheduling can hamper P&R flexibility and it cannot find a valid P&R with the given II. For example, without full routing exploration in initial scheduling, RAMP cannot find a valid P&R for kernel Susan, Lane, Inner, Csettle, Dummies, Jfadtlt, Adi, Couplong and R3000 at the MII for each. Whereas, given a modified DDG from global exploration of routing strategies at the initial scheduling stage, RESMap can easily find a valid P&R for each of these kernels at their MII. 2) During rescheduling, RAMP focus on unmapped nodes, without considering the whole DDG. Thus, this partial routing exploration at the rescheduling stage may not generate a good enough DDG for P&R. For example, RAMP cannot find a valid P&R with partial routing exploration at the rescheduling stage for kernel Susan with II = 4, Coupling with II = 7, Lane with II = 4, and Lulesh with II = 14. In those cases, BufAware directly increases II and starts the next iteration. Instead of sacrificing performance, RESMap generates a suboptimal DDG at the rescheduling stage and it finally gets valid mappings without increasing II. As a result, RESMap outperforms both RAMP and BufAware on performance.

C. REDUCED COMPILATION TIME
Fig. 8 presents the accumulated compilation time (in logarithmic form) and the number of scheduling-P&R iterations till a valid P&R is found with a tuned II in different approaches. If it is failed to find a valid P&R for a generated DDG, the upper bound of time will be added to the total compilation time. For kernel Newmdlt, as all the three approaches can get valid P&R at the first iteration, the compilation time of the three approaches is almost of the same order of magnitude. For other kernels, as RESMap generates P&R-friendly DDGs, it is more prone to find valid P&Rs with less compilation time. As a result, RESMap achieves 28.7% and 50.2% compilation time reduction, as compared BufAware and RAMP, respectively.

As compared to BufAware, the large number of iterations brings more compilation time, which mainly comes from the poor routing exploration of the initial scheduling. For
kernel Inner, Csettle, Dummies, Jfdctlt, Adi, Couplong and R3000, RAMP is failed to find a valid P&R for the first DDG generated from the initial scheduling. Therefore, much of the compilation time is spent on the trivial P&R of the first iteration. As a result, on average, RESMap outperforms RAMP on compilation time.

As compared to BufAware, although the comprehensive routing exploration at the scheduling stage could generate an optimized DDG form for P&R, it cannot ensure the success of the P&R. When it is failed to map the optimized DDG, it will wait for the timeout of the current iteration, increase II and restart another iteration. Consequently, it brings more compilation time. For kernel Jfdctlt and Adi, BufAware is failed to get valid P&R in the first iteration while RESMap succeeds in the first iteration. Therefore, the compilation time (the accumulated time till a valid P&R is found) of RESMap is within 2 hours while those of BufAware is beyond 2 hours. For kernel Susan, Coupling, Lane, and Lulesh, as both BufAware and RESMap cannot find valid P&Rs in the first iteration, BufAware starts the next iteration with increased II and RESMap starts the next iteration without changing II. Therefore, the compilation time in these kernels is compatible. As a result, on average, RESMap achieves compatible compilation time, as compared to that of BufAware.

In order to fully demonstrate the breakdown of the compilation time under the same II, Fig. 9 presents the scheduling time and P&R time of RAMP, BufAware and RESMap over DDGs with different mobility sums (the sum of mobility of all nodes in a DDG), where 5 kernels are selected and reordered according to the mobility sum from low to high. As the horizon axis is shown in Fig. 9, the underlined number indicates the mobility sum of the corresponding DDG while the other number indicates the total number of nodes in the original DDG. As the compilation time consists of DDG processing and P&R and they have a difference of orders of magnitudes, Fig. 9 presents the P&R time in logarithmic form (left vertical axis) and scheduling time (right vertical axis), respectively.

From Fig. 9, we first note that the scheduling time of RESMap increases sharply with increasing mobility sum while other approaches increase slightly with it. That is because the ILP formulation of RESMap not only considers the time assignment of original operators but also considers the graph modification referring to routing strategy exploration. However, compared to the P&R time, the scheduling time is negligible. As RESMap makes full routing exploration and generates a P&R-friendly DDG form, it can finally find 5 valid mappings while RAMP and BufAware can only find 1 and 3 valid mappings out of 5, respectively. From the perspective of P&R time, we can note that the complexity of P&R is mainly dependent on the size of DDG. As shown in Fig. 9, the P&R time of different approaches decreases greatly when the sizes of DDGs vary from 75 to 25. As a result, the enhanced scheduling in RESMap has negligible complexity growth in the whole mapping flow, but it offers a P&R friendly DDG form to improve the mapping success rate.

V. CONCLUSION

In modulo scheduling on CGRA, inadequate routing exploration of DDG leads to poor P&R solutions. To tackle the problem, a routability-enhanced scheduling method is proposed by formulating traditional scheduling and graph modification as an integer linear programming formulation. It explores routing strategies at both the initial scheduling stage and rescheduling stage and exposes a larger optimization space that is prone to find a global optimized DDG for P&R. Our experimental results show that RESMap generates mapping which leads to an average of 1.12× and 1.22× better performance, 28.7% and 50.2% compilation time reduction, as compared to 2 state-of-the-art heuristics.

REFERENCES

[1] Y. Wang, L. Liu, S. Yin, M. Zhu, P. Cao, J. Yang, and S. Wei, “On-chip memory hierarchy in one coarse-grained reconfigurable architecture to compress memory space and to reduce reconfiguration time and data-reference time,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 22, no. 5, pp. 983–994, May 2014.
[2] L. Liu, Z. Zhou, S. Wei, M. Zhu, S. Yin, and S. Mao, “DRMaSV: Enhanced capability against hardware trojans in coarse-grained reconfigurable architectures,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 4, pp. 782–795, Apr. 2018.
[3] X. Fan, D. Wu, W. Cao, W. Lu, and L. Wang, “Stream processing dual-track CGRA for object inference,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 26, no. 6, pp. 1098–1111, Jun. 2018.
[4] L. Liu, D. Wang, M. Zhu, Y. Wang, S. Yin, P. Cao, J. Yang, and S. Wei, “An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding,” IEEE Trans. Multimedia, vol. 17, no. 10, pp. 1706–1720, Oct. 2015.
[5] L. Liu, J. Zhu, Z. Li, Y. Lu, and S. Wei, “A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications,” ACM Comput. Surv., vol. 52, no. 6, pp. 1–39, 2019.
[6] J. Lee, Y. Jeong, and S. Seo, “Fast shared on-chip memory architecture for efficient hybrid computing with CGRAs,” in Proc. Design, Automat. Test Eur. Conf. Exhib., 2013, pp. 1575–1578.
[7] S. Yin, X. Yao, D. Liu, L. Liu, and S. Wei, “Memory-aware loop mapping on coarse-grained reconfigurable architectures,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 24, no. 5, pp. 1885–1908, May 2016.
[8] S. A. Chin and J. H. Anderson, “An architecture-agnostic integer linear programming approach to CGRA mapping,” in Proc. 55th ACM/EDAA/IEEE Design Automat. Conf. (DAC), Jan. 2018, pp. 1–6.
[9] Z. Zhao, W. Sheng, Q. Wang, W. Yin, P. Ye, J. Li, and Z. Mao, “Towards higher performance and robust compilation for CGRA modulo scheduling,” IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 9, pp. 2201–2219, Sep. 2020.

[10] M. Hamzeh, A. Shrivastava, and S. Vrudhula, “EPIMap: Using epimorphism to map applications on CGRAs,” in Proc. 49th Annu. Design Automat. Conf. (DAC), 2012, pp. 1284–1291.

[11] M. Hamzeh, A. Shrivastava, and S. Vrudhula, “REGIMap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs),” in Proc. 50th Annu. Design Automat. Conf. (DAC), 2013, pp. 1–10.

[12] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, “Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling,” IEE Proc.-Comput. Digit. Techn., vol. 150, no. 5, p. 255, 2003.

[13] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, “ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix,” in Proc. Int. Conf. Field Program. Log. Appl., 2003, pp. 61–70.

[14] S. Friedman, A. Carroll, B. Van Essen, B. Ylvisaker, C. Ebeling, and S. Hauck, “SPR: An architecture-adaptive CGRA mapping tool,” in Proc. ACM/IEEE Int. Symp. Field Program. Logic Appl. (FPGA), 2009, pp. 191–200.

[15] S. Dave, M. Balasubramanian, and A. Shrivastava, “RAMP: Resource-aware mapping for CGRAs,” in Proc. 55th Annu. Design Automat. Conf., 2018, pp. 1–6.

[16] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, “Architectural exploration of the ADRES coarse-grained reconfigurable array,” in Proc. Int. Workshop Appl. Reconfigurable Comput. Berlin, Germany: Springer, 2007, pp. 1–13.

[17] H. Park, K. Fan, S. Mahlke, T. Oh, H. Kim, and H. Kim, “Edge-centric modulo scheduling for coarse-grained reconfigurable architectures,” in Proc. Int. Conf. Parallel Archit. Compilation Techn. (PACT), 2008, pp. 166–176.

[18] S. Davidson, D. Landskov, B. D. Shriver, and P. W. Mallett, “Some experiments in local microcode compaction for horizontal machines,” IEEE Trans. Comput., vol. C-30, no. 7, pp. 460–477, Jul. 1981.

[19] A. Fell, Z. E. Rkossy, and A. Chattopadhyay, “Force-directed scheduling for data flow graph mapping on coarse-grained reconfigurable architectures,” in Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig), 2014, pp. 1–8.

[20] P. G. Paulin and J. P. Knight, “Force-directed scheduling for the behavioral synthesis of ASICs,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 8, no. 6, pp. 661–679, Jun. 1989.

[21] Z. Zhang and B. Liu, “SDC-based modulo scheduling for pipeline synthesis,” in Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD), Nov. 2013, pp. 211–218.

[22] T. Ruschke, L. J. Jung, and C. Hochberger, “A near optimal integrated solution for resource constrained scheduling, binding and routing on CGRAs,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW), Jun. 2017, pp. 213–218.

[23] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, “A general constraint-centric scheduling framework for spatial architectures,” in Proc. 34th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI), 2013, p. 495.

[24] J. Tsai, M. Lin, and Y. Hu, “Finding multiple solutions to general integer linear programs,” Eur. J. Oper. Res., vol. 184, no. 2, pp. 802–809, 2008.

[25] T. Sakunkonchak, S. Komatsu, and M. Fujita, “Synchronization verification in system-level design with ILP solvers,” in Proc. 2nd ACM IEEE Int. Conf. Formal Methods Models Co-Design (MEMOCODE), Jul. 2008, pp. 121–130.

[26] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, and R. B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” in Proc. IEEE Int. Workshop Workload Characterization, Dec. 2002, pp. 3–14.

[27] V. Packirisamy, A. Zhai, W.-C. Hsu, P.-C. Yew, and T.-F. Ngai, “Exploring speculative parallelism in SPEC2006,” in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw., Apr. 2009, pp. 77–88.

[28] J. Karimov, T. Rabl, and V. Markl, “PolyBench: The first benchmark for polystores,” in Proc. Technol. Conf. Perform. Eval. Benchmarking, Cham, Switzerland: Springer, 2018, pp. 24–41.

SONG MU was born in Chongqing, China, in 1995. He received the B.S. degree in information management and information system from the Xi’an University of Technology, China, in 2018. He is currently pursuing the master’s degree in computer science and technology with Chongqing University, Chongqing. His research interests include reconfigurable computing and EDA.

YI ZENG (Member, IEEE) was born in Wulanabu, Inner Mongolia, China, in 1961. He received the bachelor’s degree in computer software and theory from Wuhan University, China, in 1982. He is currently a Professor with the College of Computer Science, Chongqing University, Chongqing, China. He has authored or coauthored more than 70 refereed publications. His research interests include different aspects of software engineering, software testing, and management information systems.

BO WANG was born in Chongqing, China, in 1996. He received the bachelor’s degree in engineering from the Henan University of Technology, Jiaozuo, Henan, China, in 2019. He is currently pursuing the master’s degree in computer science with Chongqing University, Chongqing. His research interests include computer architecture, coarse-grained reconfigurable processors (CGRAs), new computing architectures based on stochastic computing, and low-power processor design.

* * *