Operation Dependency Graph-Based Scheduling for High-Level Synthesis

Aoxiang Qin, Minghua Shen, Nong Xiao
School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China
Email: qinaox@mail2.sysu.edu.cn and shenmh6@mail.sysu.edu.cn

***Abstract*—Scheduling determines the execution order and time of operations in program. The order is related to operation dependencies, including data and resource dependencies. Data dependencies are intrinsic in programs, while resource dependencies are determined by scheduling methods. Existing scheduling methods lack an accurate and complete operation dependency graph (ODG), leading to poor performance. In this paper, we propose an ODG-based scheduling method for HLS with GNN and RL. We adopt GNN to perceive accurate relations between operations. We use the relations to guide an RL agent in building a complete ODG. We perform feedback-guided iterative scheduling with the graph to converge to a high-quality solution. Experiments show that our method reduces 23.8% and 16.4% latency on average, compared with the latest GNN-based and RL-based methods, respectively.**

I. Introduction

High-level synthesis (HLS) enables hardware design at the software level for high productivity [1]. Scheduling determines the execution order and time of operations in data flow graph (DFG), impacting the performance of HLS. The execution order and time are determined by operation dependencies. Operation dependencies consist of data and resource dependencies. Data dependencies are intrinsic in the program and are represented as edges in the DFG to show data flows. Resource dependencies are determined by scheduling method to resolve resource contentions. Parallel-executed operations contend the same function unit (FU) due to resource constraints. These contentions are resolved by inserting resource dependencies in DFG to build operation dependency graph (ODG). Building a complete and accurate ODG satisfies all constraints and optimizes scheduling quality. However, it is non-trivial as dependencies increase rapidly with the number of operations [2].

Existing heuristic methods adopt handcrafted policies to sequentially build the ODG following topological order. As soon/late as possible (ASAP/ALAP) method [3] sequentially schedules operations to their earliest/latest available clock cycles (CLC) regardless of resource dependencies. List scheduling (LS) [4] considers resource dependencies. Resource dependencies are inserted when the number of scheduled operations in the current CLC reaches the resource constraints. These dependencies are from the currently scheduled operation to the following operations in the topological order. Inspired by LS, entropy-directed scheduling [5] sequentially inserts resource dependencies and schedules operations to obtain maximum entropies. These methods simplify the insertion of resource dependencies, resulting in an incomplete ODG.

Existing linear programming (LP)-based methods formalize the scheduling problem as LP problem to build a complete ODG. Integer LP (ILP) method [6] formalizes the LP problem with integer constraints. The method adopts a branch-and-bound method to search for complete and accurate resource dependencies within the entire feasible space. The search method follows a trial-and-error strategy, without the help of dependency information from the program. Therefore, ILP method suffers from weak scalability when facing diverse structured DFGs and many branches. System of difference constraints (SDC) method [7] improves scalability by a heuristic transformation scheme. The transformation introduces sub-optimal resource dependencies, resulting in inaccurate ODG.

Recently, GNN-based method [8] and RL-based method [9] are designed for scheduling. GNN-based method improves the LS method by customizing scheduling priority for DFG, reducing latency. It still uses a heuristic policy to insert resource dependency, resulting in an incomplete ODG. RL-based method iteratively improves scheduling solution guided by the perceived resource requirements among CLCs. This method can hardly learn operation relations, degrading ODG accuracy. Observing these disadvantages, we integrate GNN with RL to build a complete and accurate ODG for efficient scheduling. Our basic idea is leveraging GNN to perceive resource requirements among operations. This scheme can express accurate relations among operations. The accurate relations can guide an RL agent to build a complete ODG.

In this paper, we exploit GNN and RL to build and exploit ODG for scheduling in HLS. To enhance the accuracy of the built ODG, we propose a bi-directional perception diffusion scheme and a perception fusion scheme. The former captures operation relations from forward and backward directions, and with different scopes. The latter fuses these various-scoped operation relations to improve the accuracy of operation relations and subsequent ODG. To enhance the completeness of the built ODG, we design a symmetric action space to enable dependency insertion from forward and backward directions. This bi-directional dependency insertion enables rollback for

This work was supported by the National Key R&D Program of China under Grant No. 2024YFB4505205, the Natural Science Foundation of China under Grant No. 62072479 and 62332021, the Guangdong Natural Science Foundation under Grant No. 2024B1515020011, the PCL-CMCC Foundation for Science and Innovation under Grant No. 2024ZY2B0060, the Open Research Fund from Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ) under Grant No. GML-KF-24-12, and Xiaomi Young Talents Program. Minghua Shen is the corresponding author of this paper.

978-3-9826741-0-0/DATE25/© 2025 EDAA

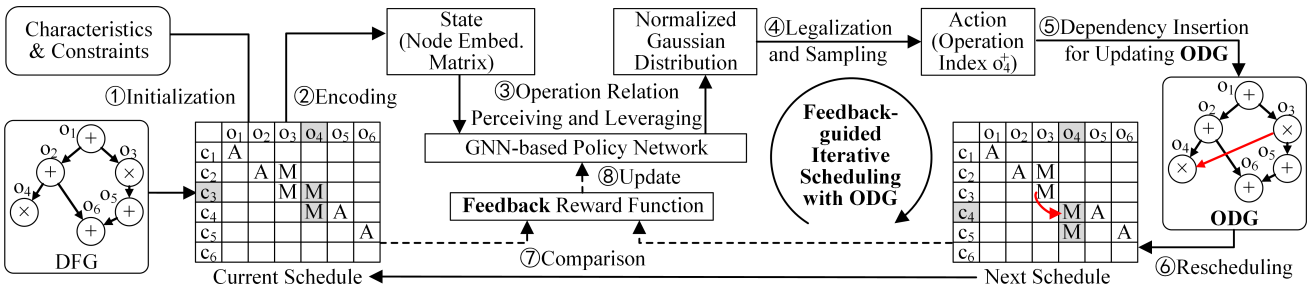

Fig. 1. The overview of our proposed feedback guided iterative scheduling with ODG. Applying the example action o_4^+ inserts resource dependency $o_3 \rightarrow o_4$ for updating ODG, and reschedules o_4 to its next CLC c_4.

those inappropriate insertions that make the optimal insertion illegal. With the ODG, we perform feedback-guided iterative scheduling (FGIS) to converge to a high-quality solution. In addition, our method can perform loop unroll and pipeline in design space exploration. The contributions are as follows.

- We propose an ODG-based scheduling method for HLS. The method perceives operation relations to build ODG, and performs FGIS for high-quality scheduling solutions.
- We propose bi-directional perception diffusion and fusion to improve ODG accuracy. We propose symmetric action space with FGIS to improve ODG completeness.
- Experiments show that our method reduces 23.8% and 16.4% latency on average, compared with the latest GNN-based and RL-based methods, respectively.

Our method is available at: https://tinyurl.com/5mhn4x5h.

II. Background

A. Scheduling

Scheduling determines the execution order and time of operations in DFG. It simultaneously minimizes resource usage and latency, while satisfying latency, resource, and dependency constraints. If we add a latency constraint of five CLCs, operations are restricted to execute within CLC $[c_1, c_5]$. If we add a resource constraint of two ADDs, no more than two operations can execute on ADD in each CLC. If we add a dependency constraint $o_i \rightarrow o_j$, o_j is scheduled after o_i finishes execution to avoid dependency violation.

B. GNN and RL

Our method adopts GNN to capture accurate operation relations, facilitating the updating of ODG. GNN [10] is a specialized neural network for graph-structured data, where edge shows relations, such as dependency, between nodes. In GNN, the nodes and edges exchange and aggregate information from neighbors to update their embeddings. Later, these updated embeddings are leveraged to perform downstream tasks, such as link prediction.

Our method adopts RL to update ODG and explore high-quality scheduling solutions iteratively. RL [11] solves Markov decision process (MDP) problems [12] through the interaction between agent and environment. Consider an MDP episode with T timesteps, its trajectory $\tau = \{(s_1, a_1, r_1), ..., (s_T, a_T, r_T)\}$. At timestep t, the agent observes a state s_t, and selects an action a_t based on a specific policy. The a_t modifies the environment, generating a new state s_{t+1}. The agent receives feedback reward r_t, which evaluates the effectiveness of a_t for s_t. By maximizing the cumulative reward $r = \sum_{t=1}^{T} r_t$, RL learns a parameterized policy network. The network estimates the probability distribution of action space, guiding the agent in making decisions.

III. Methodologies

A. Overview

Fig. 1 shows the overview of our method. We adopt GNN to perceive operation relations. We use these operation relations to guide RL agent in inserting resource dependencies for updating ODG. We use feedback rewards to train the GNN and RL, enhancing the dependency insertion policy. The dependency insertion modifies the execution order and time for operations, generating different scheduling solutions. After inserting all resource dependencies, the ODG is built, and the schedule converges to a high-quality solution.

Specifically, our method has eight steps. ① The initialization is conducted by ASAP/ALAP [3] method with given input DFG, device characteristics, and constraints. The ASAP schedule is adopted as the initial schedule. ② The input, initial, and current schedules are analyzed to conduct characteristic-based node encoding, generating a state. ③ The state is fed into a GNN-based policy network to conduct operation relation perception, generating a policy with normalized mean and variance of a Gaussian distribution. ④ The normalized Gaussian distribution is scaled and mapped into a legal action space through legalization. The legal action space comprises candidate actions satisfying pre-defined constraints. Afterwards, the legalized Gaussian distribution is sampled to generate action. ⑤ RL agent takes the action to insert resource dependencies for the chosen operation, updating ODG. ⑥ The dependency insertion modifies the execution order and time for operations, rescheduling the chosen operation to its previous/next CLC and generating the next schedule.

In specific RL episode, ②–⑥ are iterated to converge to a high-quality scheduling solution. The iteration stops when the maximum timesteps are reached, or the schedule does not improve for several timesteps (early stop). ⑦ During the training process, the improved schedule is compared with its previous one to calculate the feedback reward. The feedback reward evaluates resource usage and latency reductions of the improved schedule, as well as constraint violation of the

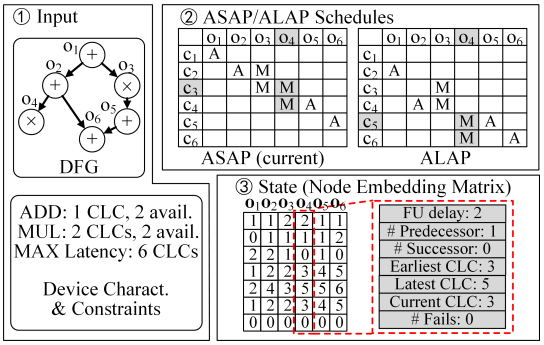

Fig. 2. The characteristic-based node encoding. The red box illustrates the seven characteristics of operation o_4.

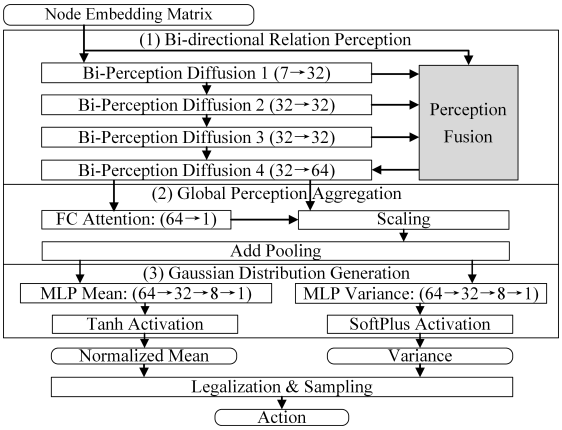

Fig. 3. The operation relation perception with GNN. The values in the braces show the transformation of hiddensizes for single embedding. Multi-layer perceptron (MLP) consists of multiple fully connected (FC) layers [14].

action. ⑧The feedback reward is recorded and accumulated in each episode. After several episodes, the cumulative feedback rewards are used to update the parameters of our GNN-based policy network through gradient ascent [13]. This update guides the training of GNN and RL for complete and accurate ODGs and high-quality scheduling solutions.

B. Characteristic-based Node Encoding

We encode each operation using its seven characteristics, generating a node embedding matrix, i.e., state. The 1st characteristic is the delay of the FU, on which the operation is executed. The 2nd/3rd characteristics are the numbers of predecessors/successors for the operation. The 4th–6th characteristics are the starting CLCs of the operation in its earliest possible, latest possible, and current schedules, respectively. The 7th characteristic is the number of failed times for rescheduling the operation, due to constraint violations. This characteristic helps the agent to recognize invalid actions and make progress. To obtain these characteristics, we schedule the DFG using the ASAP/ALAP method [3], and then analyze the DFG and the initial/current schedules.

For example, Fig. 2 illustrates the process of generating the initial state for a DFG with six operations. The targeted device has two ADDs and two MULs available. The delays of ADD and MUL are one and two CLCs, respectively. Considering operation o_4, it is executed on FU MUL with a delay of two CLCs. o_4 has one predecessor o_2, and no successor. The earliest/latest/current starting CLCs of o_4 are c_3, c_5, and c_3, respectively. Besides, o_4 has not violated any constraint before. Overall, the embedding for o_4 is represented as $\boldsymbol{e}_4 = [2, 1, 0, 3, 5, 3, 0]$. The embeddings of each node are concatenated to generate the state, $\boldsymbol{s} = [\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_6]$. Afterwards, the generated state is fed into the policy network to perceive operation relations.

C. Operation Relation Perception with GNN

Our GNN-based policy network perceives operation relations to generate a Gaussian distribution. We generate action with the distribution for dependency insertion and ODG updating. Fig. 3 shows the three phases of our policy network.

(1) The bi-directional relation perception phase comprises four bi-directional perception diffusion modules. The last diffusion module (4th module) is connected with others through a perception fusion scheme. Fig. 4 shows that each diffusion module comprises a forward and a backward aggregation layer, as well as an FC layer [14]. Rather than the vanilla module without backward aggregation, the adopted bi-directional aggregation enhances the expression of operation relations in both directions. These aggregation layers update each node embedding vector by aggregating the embeddings of its predecessors/successors, respectively. For instance, in forward aggregation, the embedding of o_2 aggregates the embedding of o_1 to generate its forward aggregated embedding. Alternatively, the backward one is generated by aggregating the embedding of o_4 and o_6 in backward aggregation. After aggregations, the two versions of aggregated embedding matrices are concatenated, and transformed to the desired output size by the FC layer. The transformed embedding matrix is activated by the LeakyReLU function [14] to generate the updated node embedding matrix for this diffusion module.

By serially applying these diffusion modules, node embedding iteratively passes to their higher-order adjacencies. Moreover, this serial application increases the node embedding dimension from 7 to 32 to 64. This iterative embedding passing and dimension increase enables the perception of operation relations with higher scope ranges. Afterwards, these perceptions of different scopes are fused by a perception fusion scheme in the last diffusion module (4th module). The scheme concatenates the state, updated embeddings in modules 1–3, and the aggregated embedding of both directions in module 4. In subsequent embedding update, these perceptions are fused, enhancing the updating of the ODG. Moreover, the scheme alleviates the vanishing gradient and over-smoothing phenomena, facilitating the model training [15].

(2) In the global perception aggregation phase, we aggregate all node embeddings to generate a graph embedding. This aggregation enables a comprehensive expression of operation relations. We first use FC Attention to generate the attention value vector [16] for updated node embedding matrix 4. These attention values weight their corresponding node embeddings with importance. By conducting Add Pooling, these weighted embeddings are summed to generate graph embedding [14].

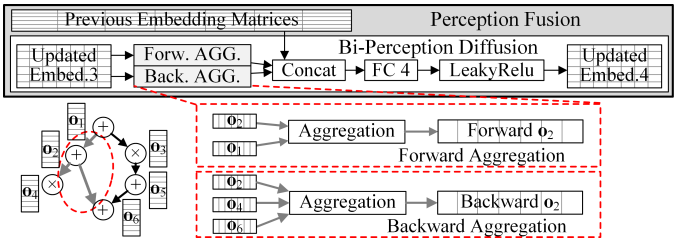

Fig. 4. The bi-directional perception diffusion and fusion schemes. The red circle and boxes illustrate the forward and backward aggregations for o_2.

(3) In the Gaussian distribution generating phase, we use the graph embedding to generate normalized mean μ_{n} and variance σ^2 of a Gaussian distribution [17]. Notably, we adopt the Gaussian distribution to construct continuous action space for adapting to DFGs with variable operation quantities. We first transform the 64-dimensional graph embedding into two real values by MLP [14] Mean and MLP Variance, respectively. Then, we activate these values using Tanh and Softplus functions [14], respectively. The Tanh function contributes to a symmetrical normalized mean value space of $[-1, 1]$, which is used to construct the action distributions in two opposite directions. The Softplus function contributes to non-negative variance for introducing randomness. Afterwards, the generated Gaussian distribution is used in the ODG update and iterative schedule.

D. ODG Update and Iterative Schedule with RL

To update ODG and conduct iterative scheduling, we use RL to generate a Gaussian distribution. ① The Gaussian distribution corresponds to a symmetric action space, enabling dependency insertion from both forward and backward directions. This symmetric action space enhances the completeness of ODG, improving scheduling quality. The sampled action corresponds to the index of the chosen operation. Applying the action inserts resource dependency for the chosen operation. If the currently inserted dependency $o_i \rightarrow o_j$ has the reverse direction with an existing dependency $o_j \rightarrow o_i$, they are both eliminated. The dependency insertion modifies the operation execution order and times, rescheduling the chosen operation to its previous/next CLC.

② To approach optimal schedules, we further reduce constraint violations by constructing a legal action space $\mathcal{A}_{\mathrm{legal}}$. An operation is added to $\mathcal{A}_{\mathrm{legal}}$ when either rescheduling it to the next/previous CLC satisfies all constraints. Considering both rescheduling directions to conduct legal operation selection facilitates the mapping from Gaussian distribution to $\mathcal{A}_{\mathrm{legal}}$. Although there are still illegal actions, they are checked and eliminated later.

③ With constructed $\mathcal{A}_{\mathrm{legal}}$, we scale the normalized mean of the generated Gaussian distribution μ_{n} with the legal action space size $|\mathcal{A}_{\mathrm{legal}}|$. The scaling constructs a legalized Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ for specific DFG, where $\mu = |\mathcal{A}_{\mathrm{legal}}|\mu_{\mathrm{n}}$. ④ Sampling $\mathcal{N}(\mu, \sigma^2)$ generates a real value i_t. i_t is used to calculate an index $i_{t,r} = round(|i_t - \mu|) + 1$ of $\mathcal{A}_{\mathrm{legal}}$, where $round()$ returns the nearest integer to a given number. The $\min(|\mathcal{A}_{\mathrm{legal}}|, i_{t,r})$th operation in $\mathcal{A}_{\mathrm{legal}}$ is chosen as the operation o_t to be rescheduled. As the action is a mapping from the sampled value i_t, its conditional transition probability is calculated using the probability density function of the Gaussian distribution [17]. The probability is used to calculate gradient later for policy network training. If $i_t \geq \mu$, $a_t = o_t^+$.

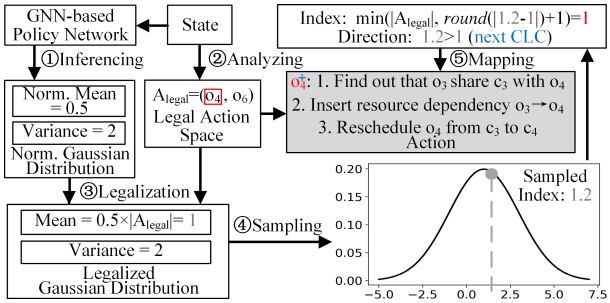

Fig. 5. The resource dependency insertion and rescheduling with RL.

⑤ Let the starting CLC of o_t be c_t. o_t^+ inserts resource dependencies from the operations that share c_t with o_t, to o_t. If there is no dependency from o_t to the operation scheduled in its next CLC, o_t is rescheduled to its next CLC. Otherwise, this action is illegal, and the 7th element (# Fails) in the state vector of o_t adds one to record this failure. This modification helps to mitigate the repetitive generation of o_t, which causes successive failure. Additionally, the environment returns a feedback penalty to the agent for learning this failure. Similarly, when the sampled value $i_t < \mu$, the action $a_t = o_t^-$. In this case, the directions of inserted resource dependencies are reversed. If there is no dependency from the operation scheduled in the previous CLC of o_t to o_t, o_t is rescheduled to its previous CLC. When o_t is successfully rescheduled, the 7th element (# Fails) in its state vector is reset as 0.

Fig. 5 gives an example of resource dependency insertion and rescheduling with RL in ODG update. ① The state in Fig. 2 is inferred by the policy network, generating a normalized Gaussian distribution $\mathcal{N}(0.5, 2)$. ② The agent analyzes data dependencies and the current schedule, determining $\mathcal{A}_{\mathrm{legal}} = \{o_4, o_6\}$. Specifically, o_1 cannot be rescheduled to the previous CLC as it is already in the first CLC c_1. Moreover, o_1 cannot be rescheduled to the next CLC c_2, which is already occupied by its successors o_2 and o_3. Similarly, o_2, o_3, and o_5 are also illegal due to their dependencies. Therefore, only operations o_4 and o_6 can be rescheduled to their next CLCs, respectively. ③ The normalized Gaussian distribution $\mathcal{N}(0.5, 2)$ is legalized to $\mathcal{N}(1, 2)$ by $|\mathcal{A}_{\mathrm{legal}}| = 2$. ④ A real value $i_t = 1.2$ is sampled from $\mathcal{N}(1, 2)$. ⑤ The index $i_{t,r}$ of $\mathcal{A}_{\mathrm{legal}}$ is calculated as $\min(2, round(|1.2 - 1|) + 1) = 1$. Therefore, the 1st operation o_4 in $\mathcal{A}_{\mathrm{legal}}$ is chosen. Moreover, as $1.2 > 1$ (i.e., $i_t > \mu$) , the action $a_t = o_4^+$. As depicted in Fig. 2, the starting CLC of o_4 is c_3, and o_3 shares c_3 with o_4. Therefore, a resource dependency $o_3 \rightarrow o_4$ is inserted by applying o_4^+. Consequently, o_4 is scheduled to its next CLC, i.e., c_5, since only o_6 is scheduled in c_5, and there is no dependency from o_6 to o_4.

After applying action a_t, the state is modified accordingly. Recall that a_t reschedules o_t to the next CLC if $a_t = o_t^+$, or

to the previous CLC if $a_t = o_t^-$. The 6th element (Current CLC) in the state vector of o_t is added by one if $a_t = o_t^+$, or decreased by one if $a_t = o_t^-$. Moreover, for the predecessor of o_t, the 5th element (Latest CLC) in its state vector is added by one if $a_t = o_t^+$ and o_t is its earliest successor. The element is decreased by one if $a_t = o_t^-$ and o_t is its earliest successor. Similarly, for the successor of o_t, the 4th element (Earliest CLC) in its state vector is added by one if $a_t = o_t^+$ and o_t is its latest predecessor. The element is decreased by one if $a_t = o_t^-$ and o_t is its latest predecessor. After modification, the new state is fed into the policy network again to start the next iteration. This iteration stops when the maximum timestep is reached, or the schedule does not improve for several timesteps (early stop). During the training process, a feedback reward for the modification is returned to the RL agent.

E. Feedback-Guided Training for Scheduling

Feedback rewards guide the training of GNN and RL for complete and accurate ODGs and high-quality scheduling solutions. The feedback reward r compares the qualities before and after applying the action, and is defined as $r = \sum_f R_{\text{reduction},f} + L_{\text{reduction}} + \delta$, where $R_{\text{reduction},f}$ and $L_{\text{reduction}}$ are the reduction of FU f and latency in this timestep, respectively. δ is the penalty factor that guides the agent to avoid invalid actions due to constraint violations, and is set as -0.1 in this paper. For example, given that the current action reduces a MUL and a CLC with no constraint violation, $r = 1 + 1 = 2$.

For trajectory τ, the cumulative reward $R(\tau)$ is calculated using feedback rewards in each timestep with a discount factor $\gamma \in (0, 1]$. Given that the final timestep in this episode is e, $R(\tau) = r_e + \gamma^1 r_{e-1} + \cdots + \gamma^{e-1} r_1$. The discounting of feedback rewards expresses the preference for immediate feedback rewards over future ones [11]. These feedback rewards are used to update parameter vector $\boldsymbol{\theta}$ of GNN and RL networks for complete and accurate ODGs and high-quality scheduling solutions. To do so, we set the objective of this update as maximizing the averaged reward expectation over the training DFG set. The expactation is approximated using the Monte Carlo method [18]. This average enhances the learning of domain knowledge among various structured DFGs. To maximize the objective, $\boldsymbol{\theta}$ is updated through the gradient ascent method [13] with a learning rate η.

IV. Experiments

A. Implementation and Settings

We implement our scheduling method based on Pytorch [19]. GNN and RL are trained by Adam optimizer [20]. We set hyperparameters according to reference [15] and hyperparameter study experiments. Specifically, the GNN type is set as GAT, learning rate $\eta = 0.001$, and discount factor $\gamma = 0.99$. Moreover, the batchsize, trained episodes, early stop timestep, and max timestep per trajectory are set as 1000, 50, $15n$, and n^2, respectively. The training is conducted on a Linux server using a Titan V GPU, a 24-core Intel Xeon CPU (2.30GHz), and 64G memory, with 2000 synthetic DFGs. They are of 100–1000 nodes, and each node pair is linked with probability 0.5. Moreover, we adopt the Cbc [21] LP solver.

TABLE I
Statistical characteristics of each benchmark, regarding number of nodes and edges, as well as critical path delay.

Benchmark	n_{node}	n_{edge}	L_{cp}
B1: backtracking-pipeline-adderchain_main_BB4	114	112	13
B2: fdtd-apml_main_BB1	133	144	7
B3: fdtd-apml_kernel-fdtd-apml_BB7	136	131	10
B4: heat-3d_kernel-heat-3d_BB9	143	141	15
B5: idct_main_BB3	323	321	5
B6: susan_susan-corners_BB24	429	489	57
B7: susan_susan-edges_BB17	435	495	62
B8: susan_susan-edges_BB43	492	572	54
B9: susan_susan-corners-quick_BB100	885	872	35
B10: idct_ChenIDct_BB1	1505	1920	88

The benchmark DFGs are generated by an LLVM pass [22] and open-sourced benchmark suites [23]–[25]. The statistical characteristics of the benchmarks are listed in Table I. The names of the benchmarks are composed of the program name, function name, and index of BB, separated by underlines. We schedule those benchmarks through LP-based [6], [7], and machine learning (ML)-based [8], [9] methods. Their performances are compared with our method regarding latency and resource usage. Besides, we conduct ablation and hyperparameter studies to verify the effectiveness of our proposed bi-directional perception diffusion and fusion schemes, as well as our hyperparameter configurations. All experiments are averaged over ten trials.

The default latency constraint is set as $L = l_f L_{\text{cp}}$, where the scaling factor l_f is set as 1.5. For the SDC method [7], l_f is set as 3 to address infeasibility caused by the heuristic transformation. The setting does not degrade latency as the SDC method minimizes latency. The default resource constraint $N_f^{\max}$ for FU f is set as the resource usage of the ASAP schedule [3], which is the maximum among all schedules. This loose constraint setting gives the maximum freedom for latency optimization but sacrifices resource usage.

For the SDC method [7] and the GNN-based method [8], the resource constraint for FU f is manually tuned to conduct fair comparisons with ours, and is set as $0.29N_f^{\max}$. The unfairness arises because our method enables resource optimization while these methods do not. As they focus on latency optimization, their resource usages are excessively high without tuning.

B. Comparison with ML-based Methods

Fig. 6 shows that our method provides an average 23.8% latency reduction than the latest GNN-based method [8]. The lower latency of our method demonstrates that our feedback-guided training contributes to a better dependency insertion policy than the handcrafted policy in the GNN-based method. Moreover, our method provides an average 0.5% resource usage reduction than the GNN-based method. The comparable resource usages are adopted for fair comparison by manually tuning the resource constraint for the GNN-based method. Our method automates the tuning process through feedback-

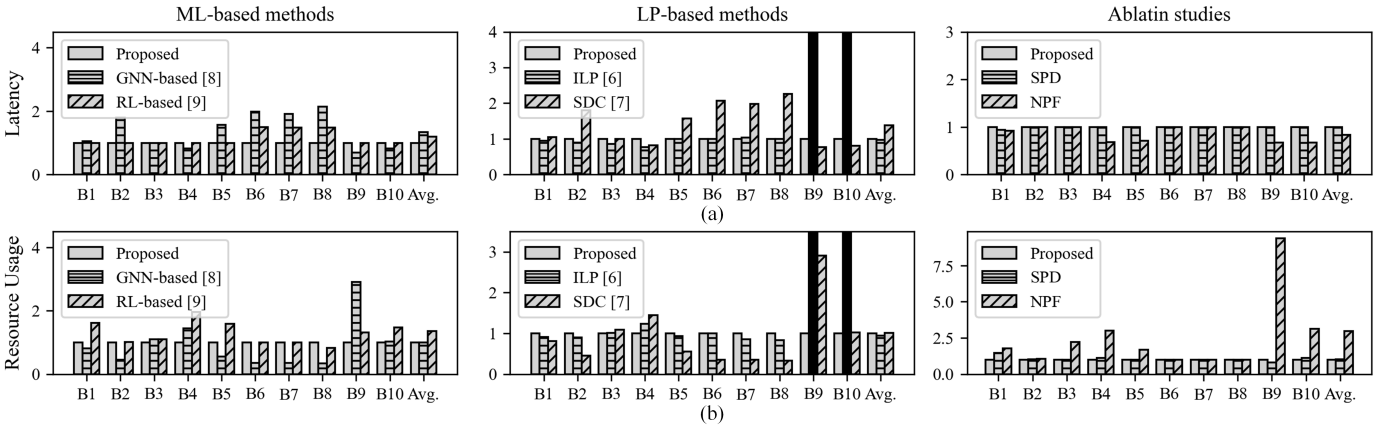

Fig. 6. Performance comparisons between our method, ML-based methods, LP-based methods, and two ablation studies. (a) Normalized Latency. (b) Normalized resource usage. The SPD and NPF represent single-directional perception diffusion [16] and no perception fusion scheme, respectively. The benchmarks with black boxes mean that they are unsolvable within a week.

guided learning. Notably, the GNN-based method consumes an average 237.6% more resource usage than our method without resource constraint tuning.

For the RL-based method [9], our method provides 16.4% latency reduction. Our method has lower latency because our feedback-guided learning balances both resource usage and latency, while the RL-based method focuses on resource optimization. Moreover, our method provides 26.5% resource usage reduction, compared with the RL-based method [9]. The lower resource usage of our method demonstrates that perceiving resource requirements among operations contributes to higher ODG accuracy than perceiving among CLCs, facilitating resource optimization.

C. Comparison with LP-based Methods

Fig. 6 shows that our method requires an average 2.9% higher latency and 6.1% higher resource usage than the optimal ILP method [6], for smaller benchmarks B1–B8. For large-scale benchmarks B9 and B10, the ILP method fails to obtain the optimal schedule within a week. The weak scalability of the ILP method is due to its uniform searching policy in the brach-and-bound scheme becoming inefficient when facing diverse structured DFGs and many branches. Rather, our method adopts RL to intelligently explore schedules using operation relations perceived by GNN. The feedback-guided training enables our method to explore less low-quality scheduling solutions, enhancing scalability.

Compared with the commonly used SDC method [7], our method provides an average 0.5% resource usage reduction and 27.9% latency reduction. The SDC method also requires manual resource constraint tuning to conduct this comparable comparison. Otherwise, the SDC method consumes an average 237.6% more resource usage than ours. The lower latency of our method demonstrates that our feedback-guided training contributes to a better dependency insertion policy than the heuristic transformation policy in the SDC method.

D. Ablation and Hyperparameter Study

Fig. 6 shows the results of two ablation studies. The first ablation study SPD adopts the single-directional perception diffusion [16] to learn scheduling policies. Our method provides an average 2.5% resource usage reduction at the cost of 0.2% latency increase than SPD. The lower resource usage of our method demonstrates that our proposed bi-directional perception diffusion scheme has higher expressive power for operation relations than the single-directional one.

Another ablation study NPF abandons the perception fusion scheme. Our method provides an average 66.4% resource usage reduction at the cost of 19.7% latency increase, compared with the NPF. The lower resource usage of our method demonstrates that fusing the dependency perception of different scopes helps in capturing resource requirements among operations, facilitating resource usage optimization. The lower latency of the NPF is due to its sacrifice of resource usage to achieve an extremely low latency.

Besides, experiments show learning rate 1e-3 outperforms 5e-4 and 1.5e-3 in resource usage with an average 5.2% and 7.9% reduction, respectively. This indicates that learning rate 1e-3 enables a better gradient optimization during limited episodes for this task. Discount factor 0.99 outperforms 0.9 and 0.95 in resource usage with an average 7.6% and 5.4% reduction, respectively. This indicates that scheduling prefers immediate rewards over future ones. GNN layer of 4 outperforms 2 and 6 in resource usage with average 3.4% and 9.3% reduction, respectively. This indicates that 4-layer GNN better balances the expressive power and the over-smoothing loss.

V. Conclusion

This paper proposes feedback-guided iterative scheduling with ODG based on GNN and RL. GNN perceives accurate operation relations, while RL uses these relations to iteratively insert resource dependencies, guided by feedback rewards. The insertion modifies execution order and time for operations, updates ODG, and generates different scheduling solutions. We enhance the accuracy of ODG with a bi-directional perception diffusion scheme and a perception fusion scheme. Moreover, we enhance the completeness of ODG with a symmetric action space setting. Experiments show our method reduces 23.8% and 16.4% latency on average, compared with the latest GNN-based and RL-based methods, respectively.

REFERENCES

[1] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. FPGA HLS today: Successes, challenges, and opportunities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(4):1–42, 2022.

[2] Bernhard H. Korte, Jens Vygen, B. Korte, and J. Vygen. *Combinatorial optimization (6th. ed.)*. Springer, New York, USA, 2011.

[3] Thomas Kailath. *Modern signal processing*. Hemisphere Pub. Corp., Washington, USA, 1986.

[4] Alice C Parker, Jorge T Pizarro, and Mitch Mlinar. MAHA: A program for datapath synthesis. In *ACM/IEEE Design Automation Conference (DAC)*, pages 461–466, 1986.

[5] Minghua Shen, Hongzheng Chen, and Nong Xiao. Entropy-directed scheduling for FPGA high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 39(10):2588–2601, 2019.

[6] C. T. Hwang, J. H. Lee, and Yu Chin Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 10(4):464–475, 1991.

[7] Jason Cong and Zhiru Zhang. An efficient and versatile scheduling algorithm based on SDC formulation. In *ACM/IEEE Design Automation Conference (DAC)*, pages 433–438, 2006.

[8] Jun Zeng, Mingyang Kou, and Hailong Yao. NeuroSchedule: A novel effective GNN-based scheduling method for high-level synthesis. In *Advances in Neural Information Processing Systems (NIPS)*, volume 35, pages 23792–23804, 2022.

[9] Hongzheng Chen and Minghua Shen. A deep-reinforcement-learning-based scheduler for FPGA HLS. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[10] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. Graph neural networks: Foundation, frontiers and applications. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 4840–4841, 2022.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction (2nd. ed.)*. MIT Press, Massachusetts, USA, 2018.

[12] Eitan Altman. *Constrained Markov decision processes*. Routledge, New York, USA, 2021.

[13] Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. In *Advances in Neural Information Processing Systems (NIPS)*, volume 35, pages 8214–8225, 2022.

[14] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, Massachusetts, USA, 2018.

[15] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning (ICML)*, volume 80, pages 5453–5462, 2018.

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *arXiv:1710.10903*, 2017.

[17] Radha G. Laha and Vijay K. Rohatgi. *Probability theory*. Courier Dover Publications, New York, USA, 2020.

[18] William L. Dunn and J. Kenneth Shultis. *Exploring Monte Carlo methods*. Elsevier, Amsterdam, Netherlands, 2022.

[19] Facebook. Pytorch v2.2, 2024. https://pytorch.org.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *arXiv:1412.6980*, 2014.

[21] COIN-OR Foundation. COIN-OR Branch-and-Cut solver, 2024. https://github.com/coin-or/Cbc.

[22] bin2415. LLVM-DFGPass: LLVM data flow graph dump, 2021. https://github.com/bin2415/llvm_DFGPass.

[23] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H. Anderson, Stephen Brown, and Tomasz Czajkowski. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *ACM/SIGDA international symposium on Field programmable gate arrays (FPGA)*, pages 33–36, Monterey, CA, USA, 2011.

[24] BoneLee. HLS-bench: A Benchmark Suit for FPGA High-Level Synthesis, 2014. https://github.com/BoneLee/HLS-bench.

[25] Louis-Noel Pouchet. PolyBench/C: the Polyhedral Benchmark suite, 2024. http://web.cse.ohio-state.edu/~pouchet.2/software/polybench.