A Supplementary Materials

A.1 Traditional List Scheduling Algorithm

Algorithm 2 Traditional list scheduling algorithm.

Input: Priority function $\mathcal{F}$, CDFG $G = (V, E)$, functional unit set U, and delay array d
Output: Generated schedule S'

1: $V_F \leftarrow \phi$, $V_R \leftarrow \phi$, $V_U \leftarrow V$, $S' \leftarrow \phi$, $Time \leftarrow 0$
2: **for** v_i in V **do**
3:     **if** $v_i.inDegree() = 0$ **then**
4:         $V_R \leftarrow V_R \cup v_i$, $V_U \leftarrow V_U \setminus v_i$
5: **while** $|V_F| < |V|$ **do**
6:     **for** u_j in U **do**
7:         $M \leftarrow MAX\_FLOAT$, $v_{chosen} \leftarrow \phi$
8:         **for** v_i in V_R **do**
9:             **if** $\mathcal{F}(v_i) < M$ **and** $v_i.canUse(u_j)$ **then**
10:                 $M \leftarrow \mathcal{F}(v_i)$, $v_{chosen} \leftarrow v_i$
11:         **if** $v_{chosen} = \phi$ **or** $u_j.unavailable(Time)$ **then**
12:             **continue**
13:         $S'.insert(v_{chosen}, u_j, Time)$
14:         $u_j.setUnavailable(Time, Time + d_{u_j} - 1)$
15:         $V_R \leftarrow V_R \setminus v_{chosen}$, $V_F \leftarrow V_F \cup v_{chosen}$
16:         **for** v_i in V_R **do**
17:             **if** $(v_{chosen}, v_i) \in E$ **then**
18:                 $E \leftarrow E \setminus (v_{chosen}, v_i)$
19:                 **if** $v_i \in V_U$ **and** $v_i.inDegree() = 0$ **then**
20:                     $V_R \leftarrow V_R \cup v_i$, $V_U \leftarrow V_U \setminus v_i$
21:     $Time \leftarrow Time + 1$

Algorithm 2 gives an overview of the traditional list scheduling algorithm. Our proposed NeuroSchedule method as shown in Algorithm 1 derives from Algorithm 2, while taking our trained neural network as the priority function. The loop (line 5) runs until each node in the CDFG is successfully scheduled. In each cycle, for every available functional unit (line 6), the algorithm schedules the operation v_{chosen} with the highest priority to the functional unit (line 13). Note that only operations of the corresponding type of the current functional unit can be chosen (line 9). After the chosen operation v_{chosen} is scheduled, all the out edges (v_{chosen}, v_i) of v_{chosen} are deleted (line 18). If the in-degree of v_i equals 0, v_i is added into the ready set V_R (line 20). Finally, the scheduling solution is obtained in S'.

In the traditional list scheduling algorithm, the most widely-used priority function is:

$$\mathcal{F}_t(i) = ALAP(v_i) - ASAP(v_i) \tag{6}$$

where $ASAP$ and $ALAP$ are defined as in Section 3. The priority function $\mathcal{F}_t$ describes the flexibility of each node in the CDFG. This priority function follows a simple but effective rule: nodes with higher flexibility are scheduled prior to nodes with lower flexibility. Recently, the force-directed priority function [12] and the entropy-directed priority function [13] are designed for better scheduling solutions. In the proposed NeuroSchedule method, our GNN-based priority function shows its superiority to other methods.

A.2 Dataset Preparation

It is necessary to acquire sufficient data for training a GNN model to predict the operations' priorities. We propose a CDFG generation algorithm, which generates CDFGs with a balanced structure and high diversity. Algorithm 3 presents the CDFG generation algorithm, where T denotes the number of operations in the generated CDFG, M denotes the minimum number of operations in a layer, N denotes the maximum number of operations in a layer, I denotes the rate that inter-layer operations are connected, U denotes the rate that an operation is not connected to a predecessor, and $G = (V, E)$ denotes the generated CDFG. As shown in the algorithm, operations in the generated CDFG are arranged into several layers, and operations in the current layer are randomly connected to operations in previous layers.

Algorithm 3 CDFG generation algorithm.

Input: T, M, N, I, U
Output: Generated CDFG $G = (V, E)$

$L_1 \leftarrow \phi, L_2 \leftarrow \phi, L_3 \leftarrow \phi, V \leftarrow \phi, E \leftarrow \phi$
while $T > 0$ **do**
  $C \leftarrow random.randint(M, N)$
  $T \leftarrow T - C$
  repeat
    $C \leftarrow C - 1$
    $O \leftarrow newOperation()$
    $O.optype \leftarrow randomOpType()$
    $V \leftarrow V \cup O$
    $L_3 \leftarrow L_3 \cup O$
    if $random.random() > U$ **then**
      if $random.random() < I$ **then**
        $O^{'} \leftarrow selectRandomElement(L_1)$
      else
        $O^{'} \leftarrow selectRandomElement(L_2)$
      $E \leftarrow E \cup (O, O^{'})$
  until $C < 0$
  $L_1 \leftarrow L_1 \cup L_2$
  $L_2 \leftarrow L_3$
  $L_3 \leftarrow \phi$
return (V, E)

A.3 Dataset Details

Two datasets are synthesized for training and evaluating the proposed model. Details of the synthesized datasets are presented as follows.

First, we synthesize a large dataset including 50000 CDFGs to train our model. The CDFGs are randomly generated using the algorithm shown in Algorithm 3, and the labels are generated using the ILP methods as introduced in Section 4. The numbers of functional units are randomly generated according to the size of CDFG, using the following function:

$$N_{u_i} = randInt(1, \max(1, \left\lfloor \frac{n_{CDFG}}{10} \right\rfloor) \tag{7}$$

Second, a small dataset including 1000 CDFGs are built to evaluate our training settings in Section 5.2. In the small dataset, the number of each functional unit is fixed to 1.

For better scalability on real applications, the dataset is extended to give more instructive information to our GNN model. The node features for each node v_i are a 11-dimensional vector F, and the details are given as follows:

- 0-1 : Operation's type (one-hot);
- 2 : Number of operation's cycles;
- 3 : The scheduling result using ASAP method (normalized);
- 4 : The scheduling result using ALAP method (normalized);
- 5 : $ASAP(v_i)$ as introduced in Section 3;
- 6 : $ALAP(v_i)$ as introduced in Section 3;
- 7 : The scheduling result of traditional list scheduling algorithm using $ALAP - ASAP$ as the priority function;

8 : $\max_{v_i \in V} ASAP(v_i)$;

9 : The maximum number of cycles obtained by traditional list scheduling algorithm;

10 : The number of corresponding functional units u_{v_i}.

Note that features $F[3,4]$ are different from $F[5,6]$, because $F[3,4]$ are the scheduling results using the idea of ASAP and ALAP, while considering the resource constraints. All the node features can be obtained efficiently by the traditional list scheduling algorithm using $ALAP - ASAP$ as the priority function.

A.4 Model details

In our network model, we use a 5-layer GIN [14] to generate the learned operation embeddings. The dimension of each GIN layer is 64. The learned operation embeddings are fed into a $(64, 64)$ dense layer, and then into a $(64, 1)$ dense layer. The activation function of each layer in the model is ReLU. There is a dropout layer between the 5-layer GIN model and the fully connected layers, while the dropout value is set to be 0.5.

A.5 Experiment Configuration

In our training process, we train our model in 2 stages, pre-training and fine-tuning. More specifically, we train our GNN model in the following way:

Stage 1 : **Pre-training:** We use our generated large dataset including 50000 CDFGs to train our model. The dataset is divided into the training set consisting of 49000 CDFGs, and the evaluation set consisting of 1000 CDFGs. The learning rate is set to be 0.0002 initially, which decays by $lr = 0.9 \times lr$ for each 10 epochs. We train the model for 2500 epochs.

Stage 2 : **Fine-tuning:** We use our generated small dataset including 1000 CDFGs to fine-tune our model. The learning rate is set to be 0.0002 initially, which decays by $lr = 0.9 \times lr$ for each 10 epochs. We train the model for 25 epochs.