Final Report: Automated ILP Scheduling

Kyle Jiun-Guei Wu

ENEE 759U – Fall 2024
University of Maryland College Park

Github repo:
https://github.com/kylejgwu/ENEE759-U---CAD-of-Digital-Circuits-Final-Project/tree/main

I. Introduction and Motivation

ILP scheduling can provide the optimal schedule of a given DFG, which has great potential in the EDA field. With a characteristic of NP property, ILP scheduling is especially useful for small graph processing.

There are some powerful ILP solvers already. This project constructs a front end CAD program to preprocess the latency-memory optimization problem and solve it by one of them, which is GLPK. The output of the program gives the minimum memory amount of the MRLC problem and the minimum latency of the MLRC problem. These numbers can be used in Pareto-optimal analysis and provides a concept of the trade-off between latency and memory consumption.

II. Approach

A. *Architecture*

The program will receive the constraints in memory and latency to construct ILP equations in the Mathprog file. The program will then call GLPK by os instructions to solve the problem and write the results into an output file. Finally, this file will be called by the program and parsed to give results on MRLC and MLRC problems.

B. *ILP equations*

The problem is similar to the general MRLC and MLRC problem, and the variables are also the same, which is shown below.

$$t = 1, \dots, l(control\ steps)$$
$$x_{it} = \begin{cases} 1, if\ v_i\ is\ executed\ in\ step\ t \\ 0,\ otherwise \end{cases}$$

For both problems, the first step is to determine the start time constraints, and slack is a necessary parameter. The slack can be obtained by applying ASAP and ALAP algorithms and the critical path can also be found during the process. Fig. 1 shows an example output, which can directly construct two corresponding arrays in Fig. 2. These two arrays are used to construct a systematic expression, representing the scalars and variables. When solving ILP, each row is an equation that adds the product of the scalar and the variable in columns.

```
[[1, 1], [1, 1], [1, 1], [2, 1], [3, 1], [4, 1], [5, 1], [5, 1], [6, 1], [7, 1]]
[[2, 1], [1, 1], [2, 1], [2, 1], [3, 1], [4, 1], [5, 1], [5, 1], [6, 1], [7, 1]]
```

Fig. 1 ASAP and ALAP output

```
[[1, 1, 0, 0, 0, 0, 0],  [['x0_1', 'x0_2', 'x0_3', 'x0_4', 'x0_5', 'x0_6', 'x0_7'],
 [1, 0, 0, 0, 0, 0, 0],   ['x1_1', 'x1_2', 'x1_3', 'x1_4', 'x1_5', 'x1_6', 'x1_7'],
 [1, 1, 0, 0, 0, 0, 0],   ['x2_1', 'x2_2', 'x2_3', 'x2_4', 'x2_5', 'x2_6', 'x2_7'],
 [0, 1, 0, 0, 0, 0, 0],   ['x3_1', 'x3_2', 'x3_3', 'x3_4', 'x3_5', 'x3_6', 'x3_7'],
 [0, 0, 1, 0, 0, 0, 0],   ['x4_1', 'x4_2', 'x4_3', 'x4_4', 'x4_5', 'x4_6', 'x4_7'],
 [0, 0, 0, 1, 0, 0, 0],   ['x5_1', 'x5_2', 'x5_3', 'x5_4', 'x5_5', 'x5_6', 'x5_7'],
 [0, 0, 0, 0, 1, 0, 0],   ['x6_1', 'x6_2', 'x6_3', 'x6_4', 'x6_5', 'x6_6', 'x6_7'],
 [0, 0, 0, 0, 1, 0, 0],   ['x7_1', 'x7_2', 'x7_3', 'x7_4', 'x7_5', 'x7_6', 'x7_7'],
 [0, 0, 0, 0, 0, 1, 0],   ['x8_1', 'x8_2', 'x8_3', 'x8_4', 'x8_5', 'x8_6', 'x8_7'],
 [0, 0, 0, 0, 0, 0, 1]]   ['x9_1', 'x9_2', 'x9_3', 'x9_4', 'x9_5', 'x9_6', 'x9_7']]
```

Fig. 2 Arrays constructed by Fig. 1

Similarly, the precedence constraints can be constructed in Fig. 3.

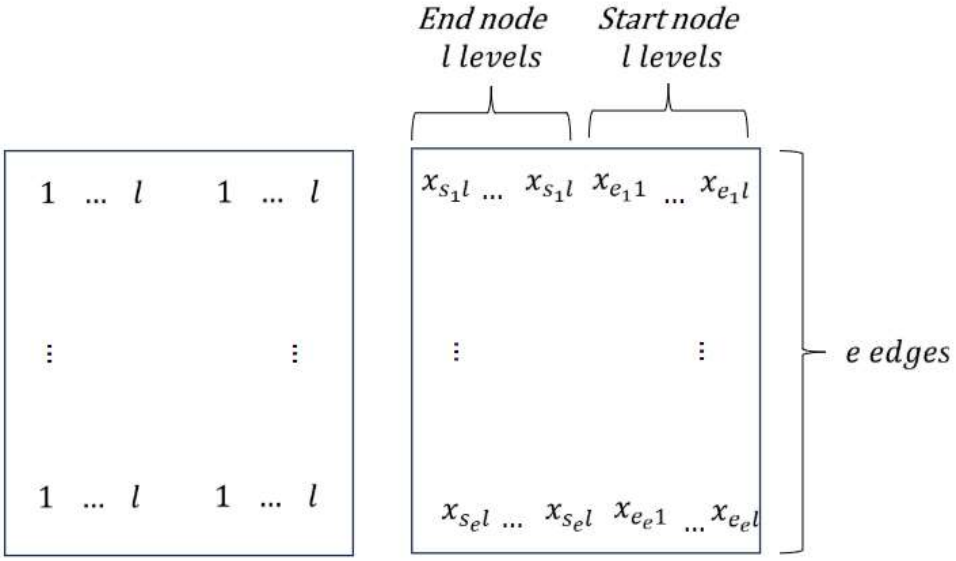

Fig. 3 Precedence constraints

The difference is that this project used memory constraints instead of traditional constraints like area, so when constructing the resource constraints, new formulas were applied.

$$w_t = \sum_{edges=0}^{total\ edges} w_{ij}(x_{i1} + \ldots + x_{it})(x_{j(t+1)} + \ldots + x_{il}),\ t = 1 \ldots l$$

The above formula shows that an edge will be counted in its memory in time step t only when the start node is prior to this time step and the end node is later than this time step. Notice that this formula is a quadratic equation, which requires further simplification to be solvable by GLPK. Since there will only exist a '1' for all x_it, t=0…l, the product of two (x+...) is actually an and function. Previous research has shown that this can be reduced into three formulas by introducing an extra variable.

$$y = x_1 \wedge x_2 \quad \begin{cases} y \geq x_1 + x_2 - 1 \\ y \leq x_1 \\ y \leq x_2 \end{cases}$$

According to these equations, memory constraints can be built.

C. Pruning

To accelerate the entire process, the number of variables should be reduced as much as possible. Firstly, by setting all the x, y variables Binay can improve the speed.

In most cases, there will not be so many variables due to the start time constraints. Some nodes have already been determined and will be constants, which is trivial. This optimization can be done by slack, which has been already calculated. For a sparse graph, this can significantly accelerate the flow. Fig. 4 shows that this can reduce the number of variables from 70 to 4 in an extreme case (slack=0).

```
[['x0_1', 'x0_2', 'x0_3', 'x0_4', 'x0_5', 'x0_6', 'x0_7'],
 ['x1_1', 'x1_2', 'x1_3', 'x1_4', 'x1_5', 'x1_6', 'x1_7'],
 ['x2_1', 'x2_2', 'x2_3', 'x2_4', 'x2_5', 'x2_6', 'x2_7'],
 ['x3_1', 'x3_2', 'x3_3', 'x3_4', 'x3_5', 'x3_6', 'x3_7'],
 ['x4_1', 'x4_2', 'x4_3', 'x4_4', 'x4_5', 'x4_6', 'x4_7'],
 ['x5_1', 'x5_2', 'x5_3', 'x5_4', 'x5_5', 'x5_6', 'x5_7'],
 ['x6_1', 'x6_2', 'x6_3', 'x6_4', 'x6_5', 'x6_6', 'x6_7'],
 ['x7_1', 'x7_2', 'x7_3', 'x7_4', 'x7_5', 'x7_6', 'x7_7'],
 ['x8_1', 'x8_2', 'x8_3', 'x8_4', 'x8_5', 'x8_6', 'x8_7'],
 ['x9_1', 'x9_2', 'x9_3', 'x9_4', 'x9_5', 'x9_6', 'x9_7']]
```

```
[['x0_1', 'x0_2', 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0],
 ['x2_1', 'x2_2', 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 1]]
```

Fig. 4 Pruning array by slack

Increasing the number of parameters significantly increases the runtime. When dealing with the memory constraints, lots of variables were introduced because each y is a new variable (although some of them were pruned based on the previous method). There is a simpler model that reduces variables but gives the wrong memory calculation, which is the mapping I presented in the final presentation. This can be a potential mapping if the nodes were further constrained in some special cases, but not useful in this project.

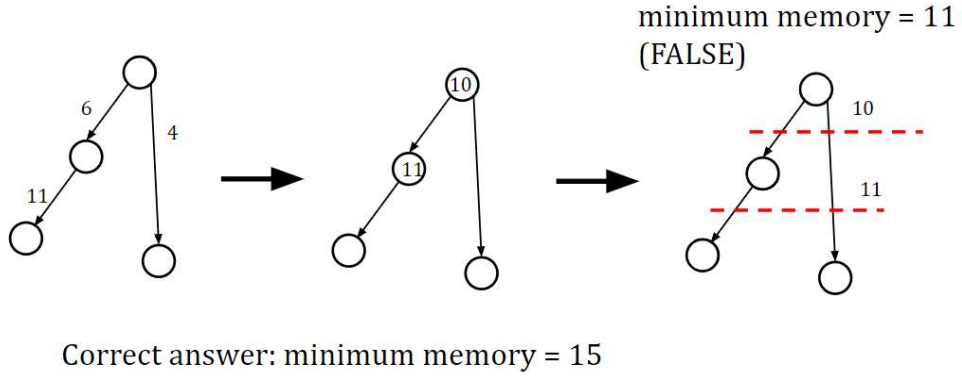

Fig. 5 Potential mapping method

III. RESULTS

For 50 node testbenches, it takes an extremely long runtime when L is somewhat larger than the length of the critical path. For example, a 34-level critical path graph, L=40 takes more than 8 hours and doesn't have the output yet. Therefore, the Pareto-optimal analysis was done around L=critical path.

The process of collecting points of the Pareto-optimal plot is shown below. The invalid inputs will be automatically detected in the program and return 'Failed'. When 'Failed' is obtained, increase the boundary of l and m.

1. Use ASAP to find L for the starting point
2. Use MRLC with an input L to find M for the starting point
3. Decrease L and increase M to find more points
4. Increase L and decrease M to find more points

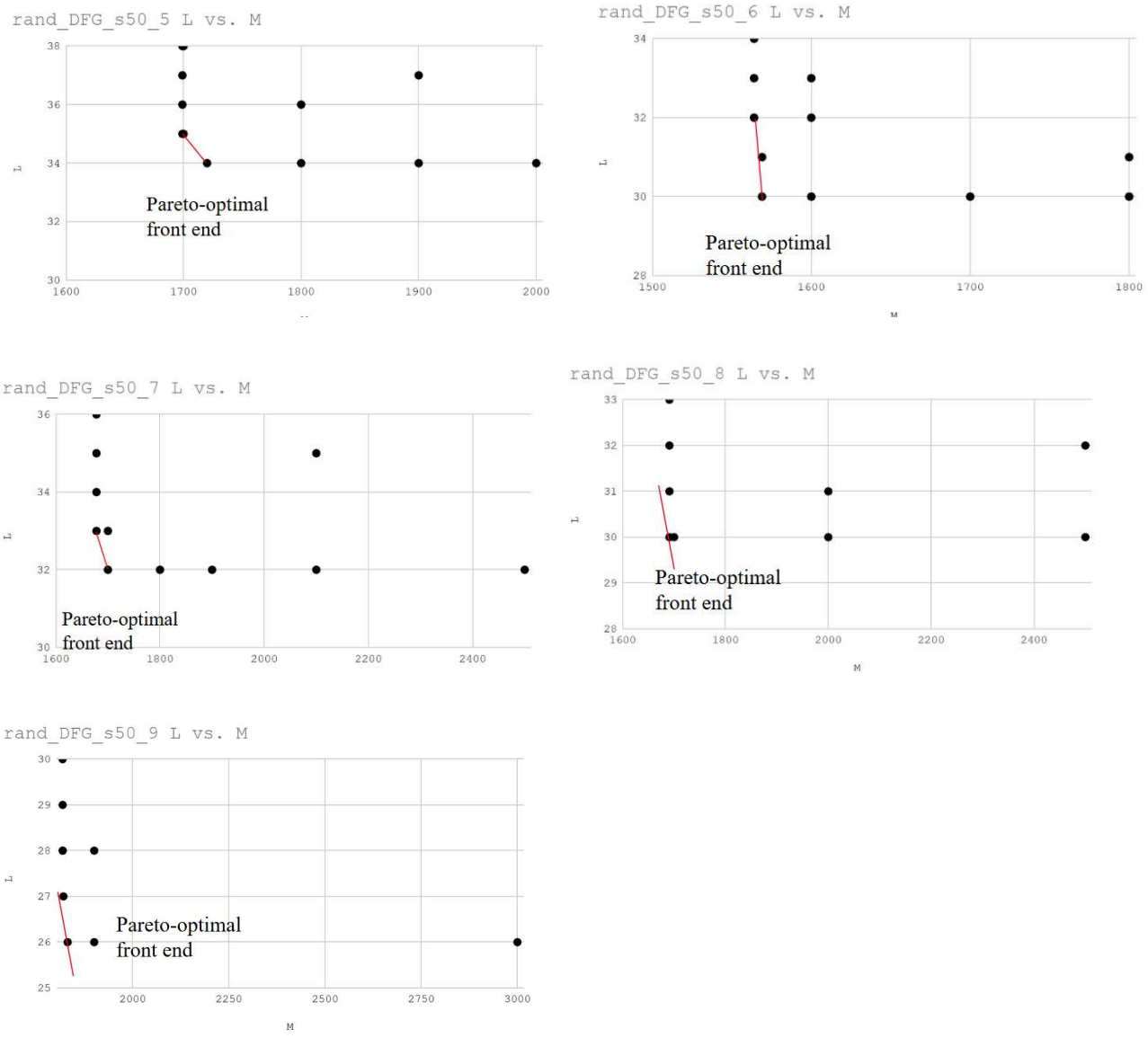

Fig. 6 Pareto-optimal analysis of 6 testbenches

In conclusion, the trade-off between can be seen in the figures above. However, due to the limitation of the data, the front end is not significant. In some cases, increasing latency constraints would not help decrease memory amounts since the nodes that can be scheduled can only be executed in certain timesteps which is not the critical timestep. For example, benchmark rand_DFG_s10_1have has a Pareto-optimal frontend with only a point because the free nodes do not dominate the memory consumption. Fig. 7 clearly shows the expected results.

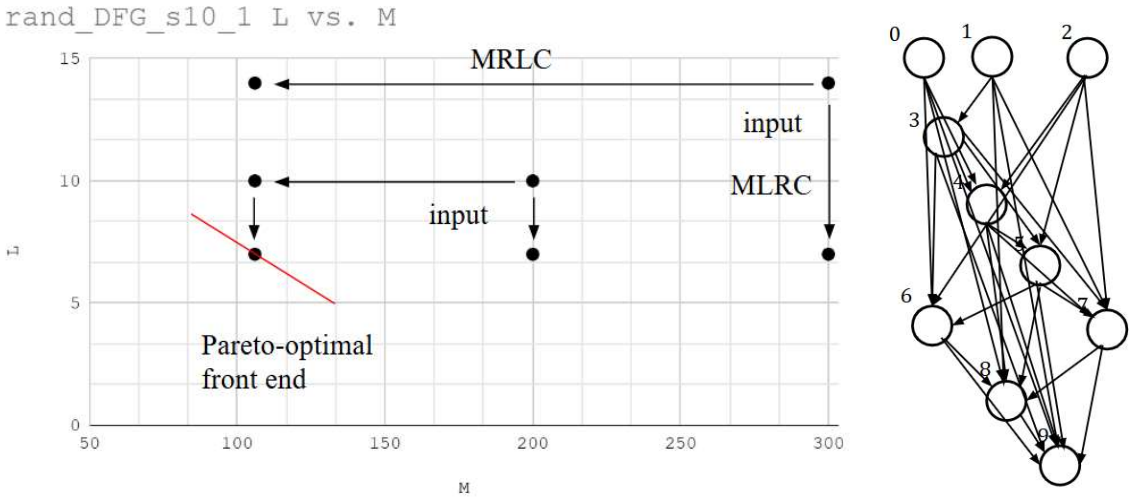

Fig. 7 Small input graph and Pareto-optimal analysis

On the other hand, increasing memory constraints can also have no contribution to minimizing latency because the latency has a hard limit of the critical path length. Therefore, this project can only provide limited Pareto-optimal analysis.

Reference

[1]Yin, J., Zhang, Z., & Yu, C. (2022, December). Exact memory-and communication-aware scheduling of dnns on pipelined edge tpus. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)* (pp. 203-215). IEEE