Andy, Yu-Guang Chen

EE6094 CAD for VLSI Design
Programming Assignment 2: Scheduling
(Due: 23:59:59, 2023/04/16)

Introduction

In the front-end design phase, behavior synthesis converts high-level description of a design to the corresponding netlist. To meet performance requirement or reduce area/power overhead, the behavior synthesis process will try to either reduce computation cycles or minimize required computational unit. This process is called **Scheduling**. The goal of scheduling is to determine when each operation should be executed so that the above goals can be achieved. There are various scheduling algorithms. In this Programming Assignment, you are asked to implement "**List Scheduling**" algorithm [1][2] to solve a latency-constrained resources minimization problem.

Background

In integrated circuit design flow, high-level synthesis is a step in the standard design cycle. It is a process to convert the algorithm-level or behavior-level description of a circuit design specification into a circuit structure description under certain constraints. Specifically, scheduling and binding mainly determine the performance and required hardware resources of the generated RTL.

Scheduling plays a central role in the behavioral synthesis process, which automatically compiles high-level specifications into optimized hardware implementations. Scheduling, which exploits the parallelism in the behavior-level design and determines the time at which different computations and communications are performed, is commonly recognized as one of the most important problems in behavioral synthesis. However, finding an optimal schedule is intractable in general.

Problem definition

You are asked to implement a latency-constrained "**List**" scheduler to minimize the resource required. Assume there are only two types of operations: addition and multiplication. The operating time of addition takes 1 time unit, and multiplication takes 3 time units. Moreover, only two types of computing resources are available, which are adder and multiplier. To simplify the problem, we assume that the adder can only perform addition function and the multiplier can only perform multiplication function.

In this programming assignment, you're given an input file which contains the sequencing graph information of operations and the maximum allowed latency. Your program needs to find a scheduling result which leads to minimal resource (hardware) requirement while satisficing timing constrain. Then a corresponding output file should be generated with the information of resource requirement and scheduling result. The format of input and output file will be shown below.

Input file format

Fig. 1 gives an example of a sequencing graph, and the sample input file describes the sequencing graph. The line below **# Testcase** gives you the maximum latency. Starting from the line below **# Test data**, the information of the nodes is listed, each line contains a unique index of the node, followed by a symbol and a list of numbers. There are 4 kinds symbol: **i** means the node is an input node, **o** means output node, **+** means addition node, and ***** means multiplication node. The number list represents the successor(s) of the node. Takes the 6th line of the sample input file as an example, node 2 is an input node, and its successor nodes are 4, 5, 6 (data of node 2 will be passed to node 4, 5, 6). **Note that the order of node index is NOT guaranteed in the input file.**

Output file format

The first line gives the required number of adders of your solution. The second line gives the required number of multipliers of your solution. The (*i*+2)th line lists the node(s) executed during the *i*th time unit in the increasing order w.r.t the node number. **The output format is strict**. The output file of your program should be named as *name***.out** where *name* is the input file name.

Example

Below is a sample input file and corresponding output file with comments added. Fig. 1 is the corresponding DFG of input file and Fig. 2 is the result of scheduling.

Sample input file:

```
# Testcase 1
Latency constrain: 5       // latency constrain

# Test data
1 i 5              // node 1 is input, which has successor of node 5
2 i 5 4 6
3 i 4 7
4 + 6 7            // node 4 is Add operator, which has successors node 6, 7
5 + 8
6 * 8 9 10         // node 6 is Multiply operator, which has successors node 8, 9, 10
7 + 9 10
8 + 11
9 + 12
10 + 13
11 o               // node 11 is output, it doesn't have any successor
12 o
13 o
```

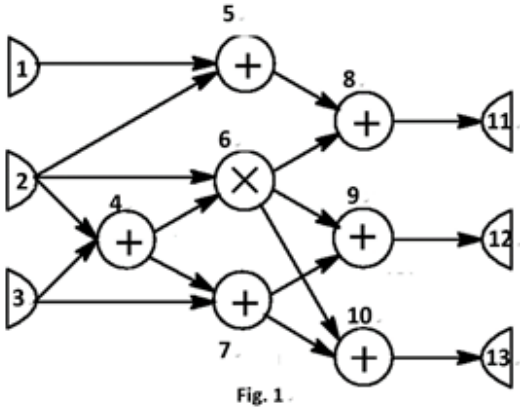

Fig. 1

Sample output:

```
3          // number of adders
1          // number of multipliers
4 5        // execute the operation of node 4 and 5 at t1
6 7        // execute the operation of node 6 and 7 at t2
6
6
8 9 10
```

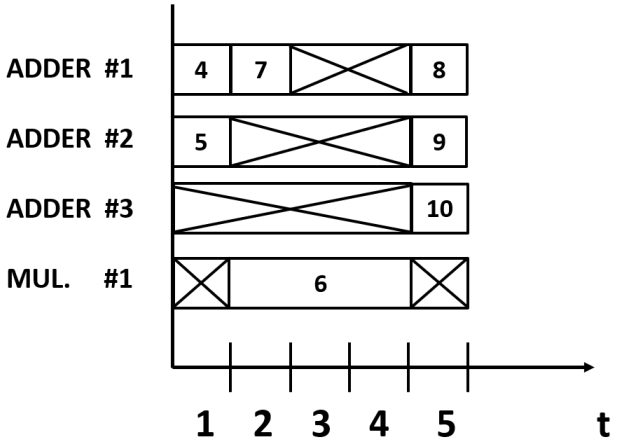

Fig. 2

Algorithm

You must use **List** algorithm to solve the ML-LCS problem. You are not allowed to use any other algorithm to solve this problem.

Bonus

In normal work, we only ask you to implement **List Scheduling,** however, the scheduling performance of this algorithm may be not good enough since it does not guarantee optimal solution. Therefore, in the bonus section, you are asked to implement **Force-Directed** [3] algorithm, trying

to find the globally best scheduling results. You are also encouraged to compare the effectiveness, efficiency, and resource cost (such as memory) of the two algorithms, and describe your observations.

Requirement

1. You must write this program in C or C++. No open source codes are allowed to use. (i.e., you MUST implement the tool by yourself). You can use any data structure to realize your program. We will verify your program on workstation. Therefore, **you have to make sure your program can be executed correctly and successfully on workstation, any other version of C++ or compilers are not allowed. The run time of your program is limited to at most 2 hours per testcase.**
   The workstation information is shown below:
   - System: CentOS 6.10
   - Compiler: gcc 4.8.2
   - C++ version: C++11
2. We will verify your program on a workstation with a **Makefile**. Therefore, you need to write a **Makefile** which can compile and execute your program directly. Your **Makefile** should at least contain these 3 commands, which are (1) **make all**, (2) **make run**, and (3) **make clean**. The descriptions of each command are shown below.
   (1) **make all**: This command will automatically compile your source codes and generate the corresponding objects and executable file.
   (2) **make run**: This command will execute your executable file and run your program.
   (3) **make clean**: This command will automatically remove all the objects and executable file generated by **make all**.
3. You should also write a report. We don't restrict the report format and length. In your report, you must at least include:
   (1) How to compile and execute your program; (You can use screenshot to explain)
   (2) The completion of the assignment (If you complete all requirements, just specify all)
   (3) Description of the data structure and algorithm you used.
   (4) The hardness of this assignment and how you overcome it
   (5) Any suggestion?
4. All files should be submitted through ee-class. You have to submit **a source code file** with its main file named as *StudID*_PA2.cpp (ex: 123456789_PA2.cpp), **a report** named *StudID_Name*_PA2_report.pdf (ex: 123456789_陳聿廣_PA2_report.pdf), and **a Makefile** to compile and execute your program. If your source code contains more than one file, only the "driver" file that contains main function should follow the naming rule mentioned above. If you implement a bonus version, please separate it from the original version. Name your

"driver" source code file of the bonus version as StudID_PA2_bonus.cpp (ex: 9862534_PA2_bonus.cpp) and upload the entire bonus package to ee-class. Note that **the only acceptable report file format is .pdf**, no .doc/.docx or other files are acceptable. **BE SURE to follow the naming rule mentioned above. Otherwise, your program will be not graded.**

Grading

The grading is as follows:

(1) Correctness of your code: 30%
(2) The quality of your solution (based on original version): 30%
(3) Readability of your code: 10%
(4) The report: 10%
(5) Demo session: 20%
(6) Bonus (at most): 10%

Please submit your assignment on time. Otherwise, the penalty rule will apply:

- Within 24hrs delay: 20% off
- Within 48hrs delay: 40% off
- More than 48hrs: 0 point

Contact

For all questions about PA2, please send E-mail to TA 蔡書儀 (noobyves@g.ncu.edu.tw)

References

[1] Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Syst. Techn. J. 45, 1563–1581 (1966)

[2] Epstein, L. (2008). List Scheduling. In: Kao, MY. (eds) Encyclopedia of Algorithms. Springer, Boston, MA.

[3] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661-679, June 1989, doi: 10.1109/43.31522.