EE6094
CAD for VLSI Design
Programming Assignment 2 Report

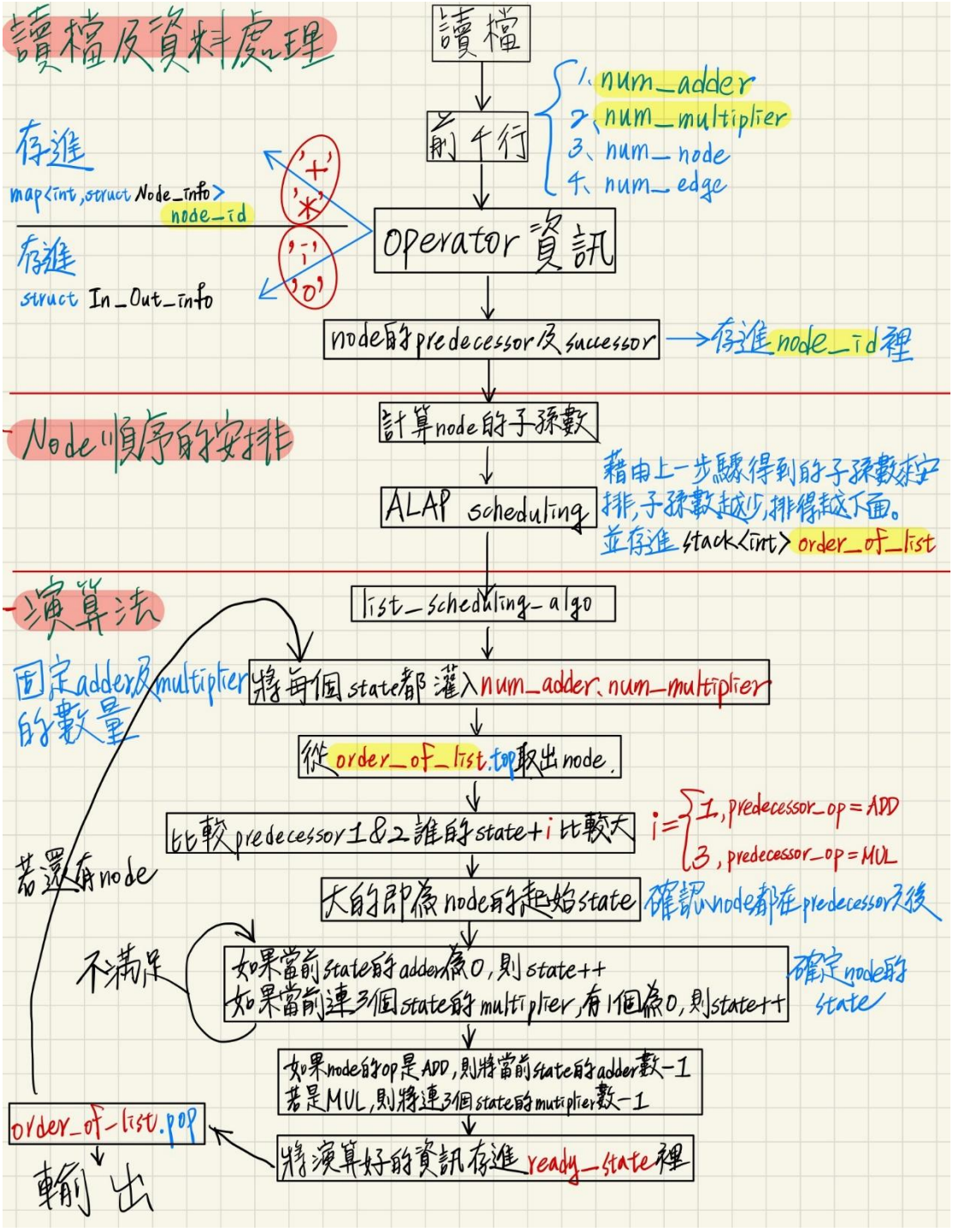
Student Name:107501019

Student ID:魏子翔

I. Problem Description

因為硬體的限制，所以編排 graph 時，必須去考慮每個 state 中硬體的使用量，並且同時能達到 minimize latency，因此本次作業就是些出一套演算法去將 DFG 做 resource-constrained 上的編排。

II. Program Structure



III. Data Structures / Algorithms Used

Data Structures：

```
struct Node_info     //各個 node 的資訊
{
    int node;    //node
    int predecessor1;    //node 的兩個 input
    int predecessor2;
    int num_descendant; //node 的子孫數量
    int status; //確認 node 是不是該 graph 最底下的 node(除去 output  node)
    vector<int> successor;   //node 輸出給到的位置
    char operator_type; //node 的運算子類型
};
map<int, struct Node_info> node_id;
```

利用 map 包 struct 的形式來存 node 的各種資訊，因為只要給予 key 值就能馬上得到 value，這樣就不用去用 for 迴圈搜尋。

老師上課有介紹 linked list，但是我上網查了一下，linked list 主要用在有增加或減少 node 的程式裡，但是本次作業沒有加減 node 的需求，所以我就把 struct 建成直接存前後 node 的數值，因為同樣都是 4 個 bytes。

```
struct In_Out_info   //儲存 input  node 及 output  node
{
    vector<int>input;
    vector<int>output;
}i_o_node;
```

建立 struct，裡面用 vector 去儲存 input node 及 output node。

```
stack<int> order_of_list;    //給演算法去取的順序
```

利用 stack 將要給進演算法的 node 排列好，然後 last in 的就是要最優先丟進演算法的 node。

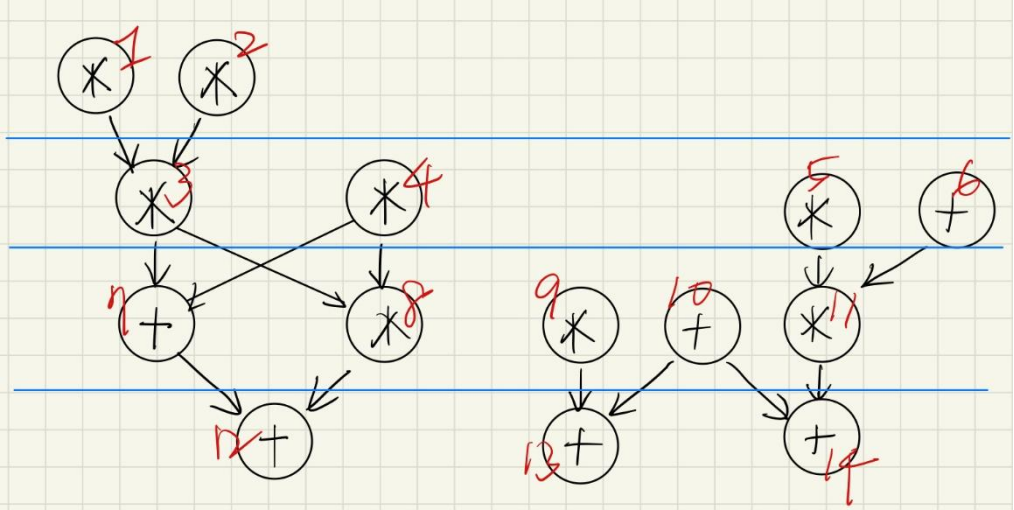
```
struct Ready_state   //給 list scheduling algotithm 用
{
    int state;
    int node;
    char operator_type;
};
map<int, struct Ready_state> ready_state;
```

一樣使用 map 包 struct 的形式將演算好的 node 的資訊丟進 ready_state，到時候要輸出時也不用一直用 for 迴圈去找資料，只要給定 key 值，value 就出來了。

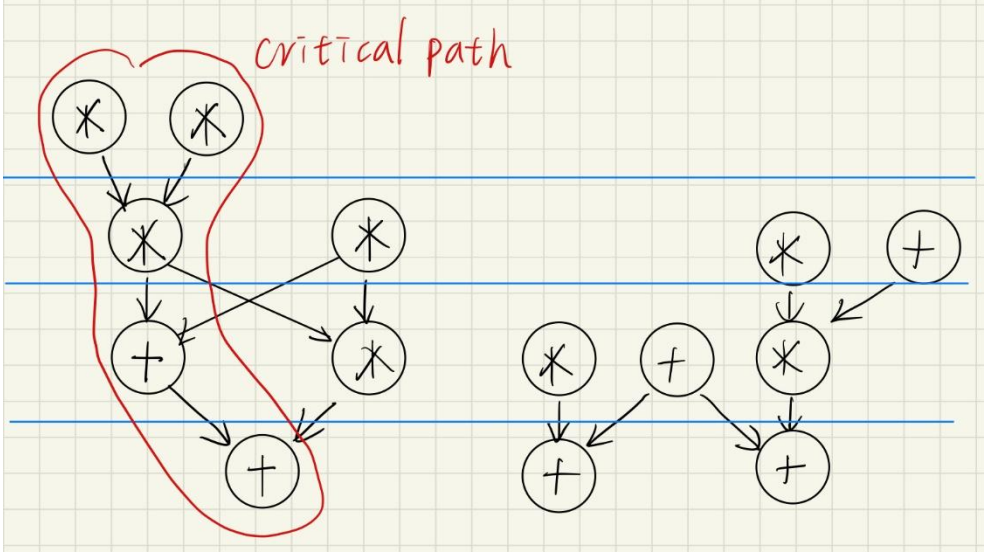
```
struct State_op_num //各 state 中，adder 與 multiplier 的個數
{
    int multiplier;
    int adder;
}state_op_num[SIZE];
```

各 state 中，adder 及 mulplier 的個數

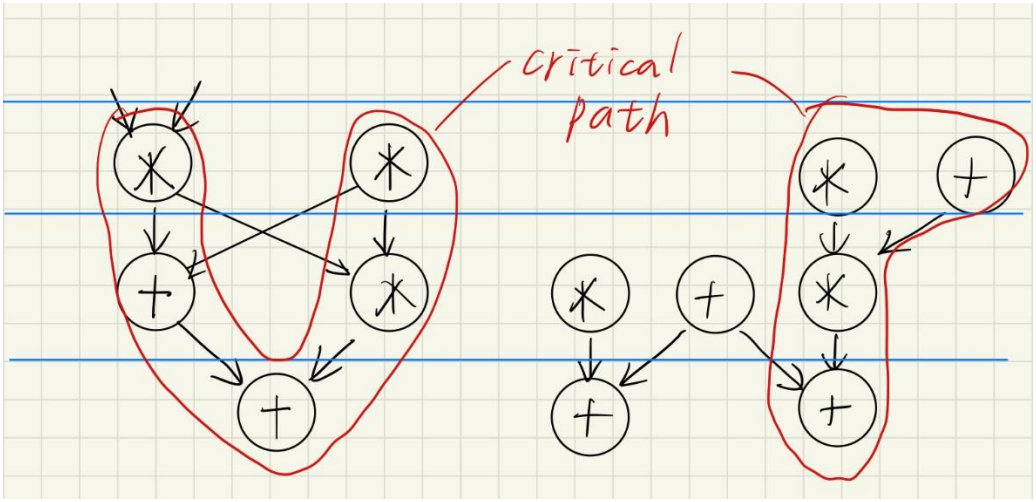
Algotithm：



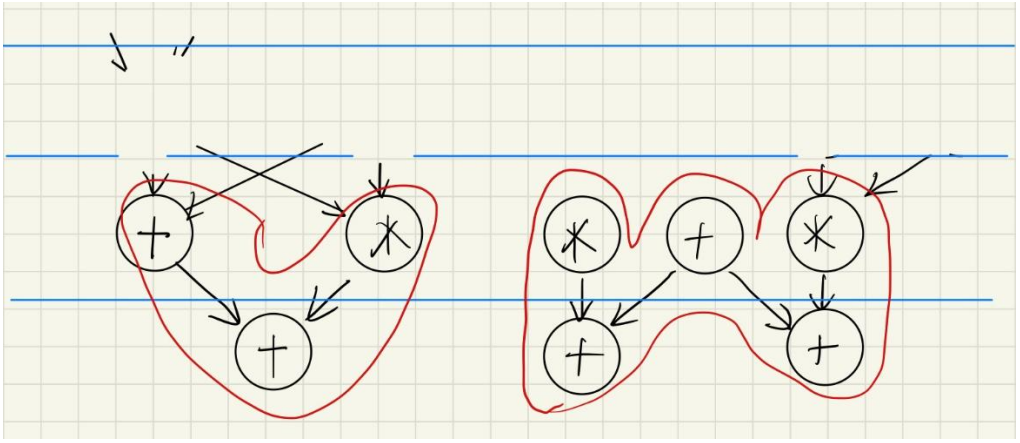
首先利用 ALAP 的形式將 graph 排列好。



找出 critical path，並將頭找出來然後進演算法。



當原本的頭被丟進演算法後，critical path 就有可改變，再次找出 critical path 並丟進演算法。



以上述方法不斷的進行。

本次作業的演算法先找出每個 node 的子孫數，再利用 ALAP scheduling 將圖形的優先順序排好，子孫數越少排的下面，並同時找出不同階段的 critical path 的頭，並依照越後面被使用的 node 越先被丟進 stack 中，因此越先被取出的 critical path 的頭，就會越晚放進 stack，之後才會越線被拿出。

將 node 一個一個拿出丟進演算法，首先判斷 ready_state 中 predecessor 誰的 state+i 比較大(predecessor 的 op 為 ADD，i=1；為 MUL，i=3)，並將大的給到 node 的 state 中，然後以此為基準開始判斷當前 state 的 adder 或 multiplier(multiplier 需判斷 3 個 state)是否還有位置，如果沒有，則 state++；確定好 state 之後，依照 node 的 operator 型態，將 state 中的 adder-1 或 mutiplier-1；最後將演算好的

資訊丟進 ready_state，不斷重複直到 stack 中的 node 被取完。

IV. How to Execute

首先在工作站或是 command line 中輸入

./PA2_107501019.exe　testcase　testcase.out

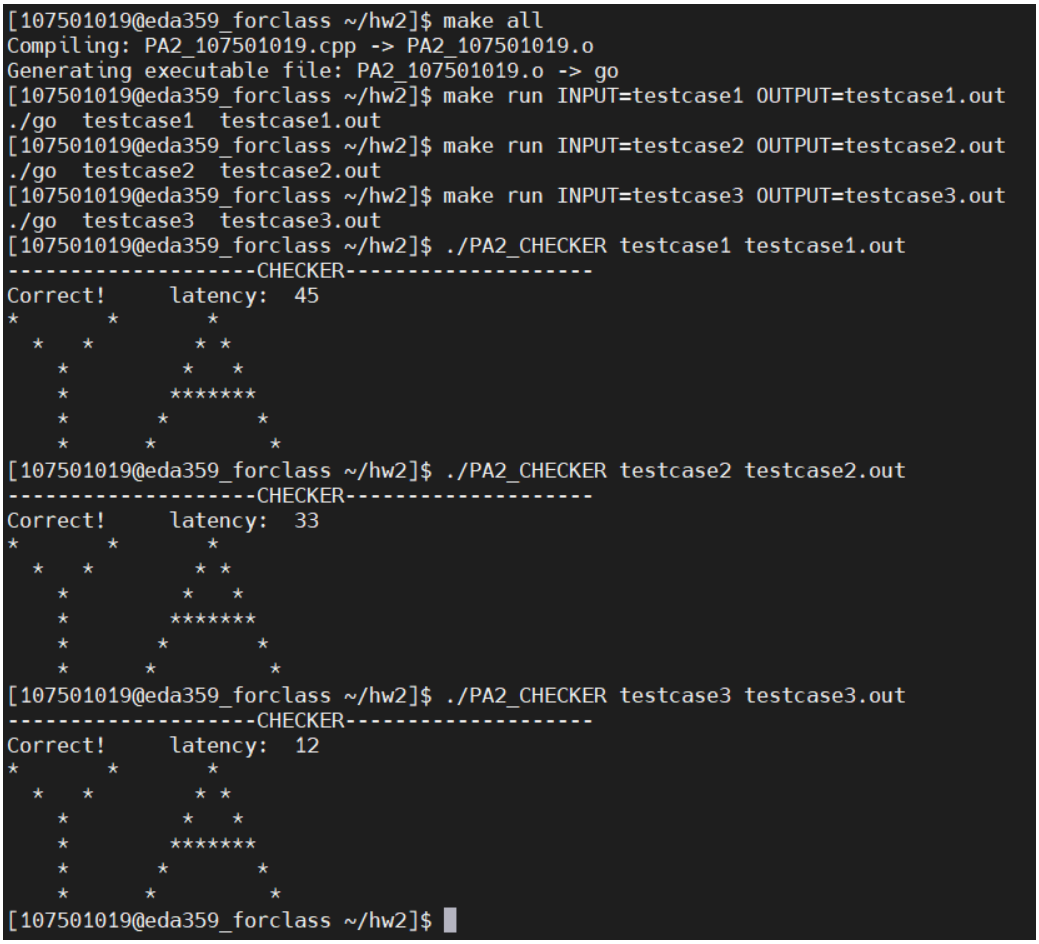
先輸入執行檔，在輸入要測試的 testcase 名稱，最後再輸入 output file 的名稱。

執行檔步驟如同 II.Program Structure 的流程圖。

V. Difficulty Encountered

一開始再想演算法時，不知道該怎麼去下手，面臨到該怎麼去取 node 的困擾，上網查了許多 pseudo code，仍然沒有甚麼想法，但是過幾天後聽到有人用 distance 去判斷 node 的優先順序，因此我想又想了一段時間，就想到利用子孫數去安排 ALAP，再藉由 ALAP 中越上面的越先取的方式去寫出本次作業的演算法。

VI. Experimental Results



VII. Reference

- https://dangerlover9403.pixnet.net/blog/post/216833943
- https://github.com/sally12guy/List-Scheduling-Algorithm/blob/master/list_scheduling.cpp
- https://officeguide.cc/c-cpp-stack-tutorial/
- https://mropengate.blogspot.com/2015/12/cc-map-stl.html
- https://ithelp.ithome.com.tw/articles/10231350