Scheduling

Use list scheduling too schedule all the operations

實驗日期：2024.04.15

學號姓名：B092040016 陳昱逢

實驗內容及過程

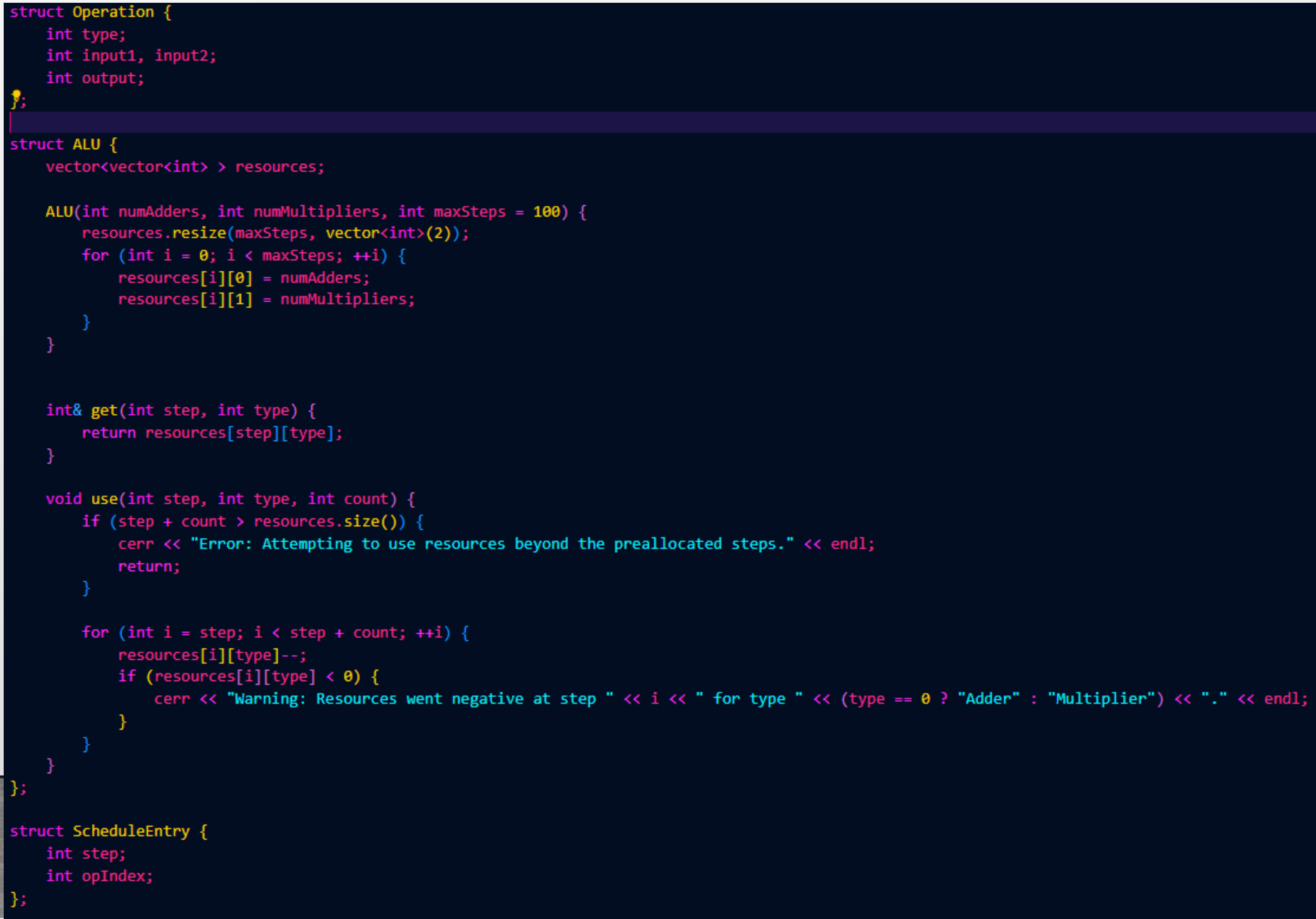
- **主要內容：**根據實驗給定的 Data Flow Graph，使用 List Scheduling 去找出在有加法器或乘法器的限制下如何才能排程完所有的運算。

- **主要過程：**參考教授提供的 Scheduling 程式架構，利用該架構寫出可以計算不同數量的加法器或乘法器下，需要幾個 step (clock) 才可以完成所有運算，接著再撰寫 RGB to YUV 的文字檔送入程式跑。

實驗過程-程式碼說明

首先定義基本架構，Operation 結構代表一個指令，存放指令形態、兩個輸入值以及一個輸出；ALU 結構裡存放一個二維陣列，row 代表有幾個 step，程式中預設本次實驗最大的情況為 100 個 step（可調整），column 代表硬件的形態（加法器與乘法器），裡面存放的值為該 step 下所剩餘的加法器或乘法器數量，而 ScheduleEntry 結構存放該在第幾個 step 執行（也就是排程後的結果）以及他在原本指令中是第幾個指令。

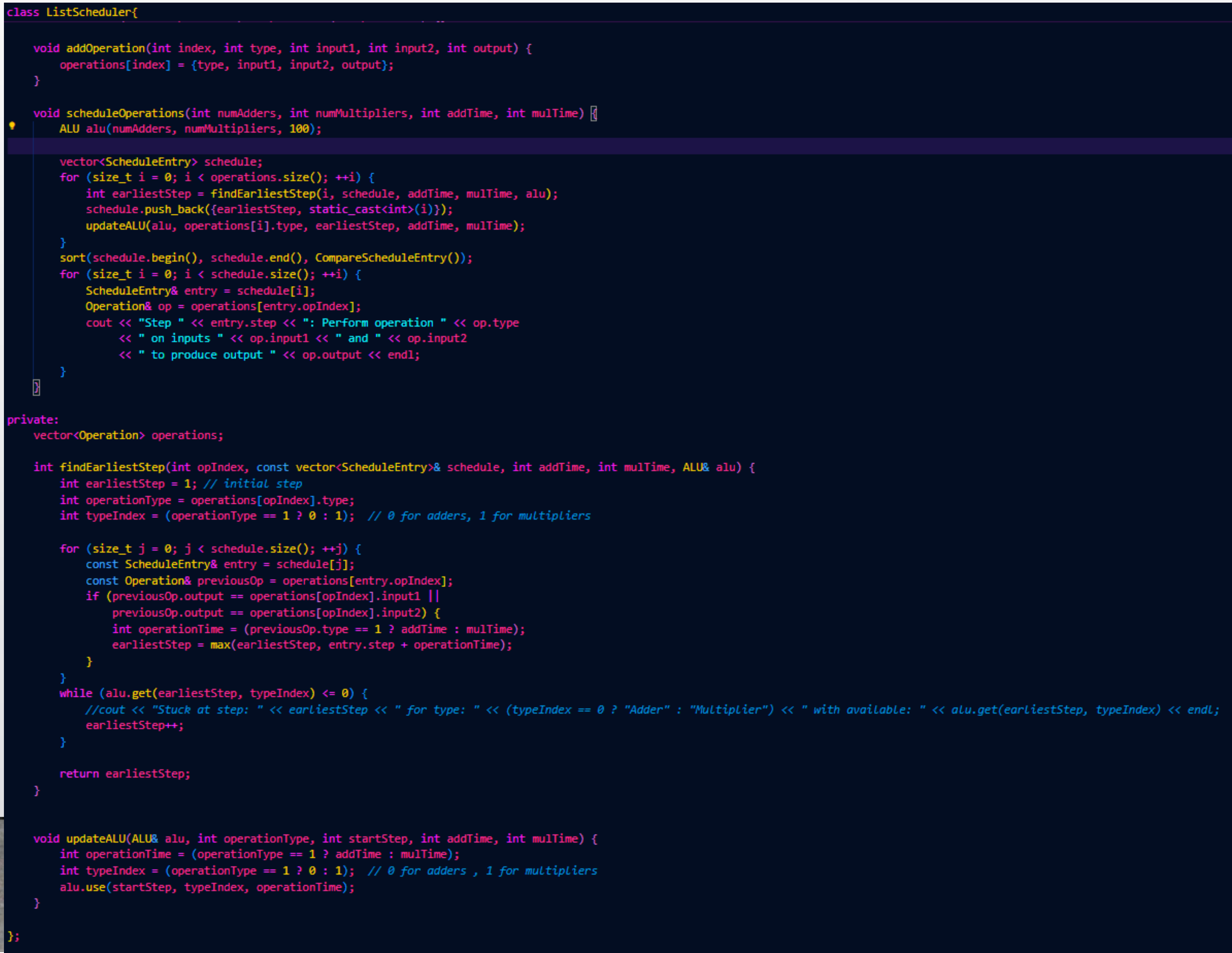
架構圖程式碼:



```
struct Operation {
    int type;
    int input1, input2;
    int output;
};

struct ALU {
    vector<vector<int> > resources;

    ALU(int numAdders, int numMultipliers, int maxSteps = 100) {
        resources.resize(maxSteps, vector<int>(2));
        for (int i = 0; i < maxSteps; ++i) {
            resources[i][0] = numAdders;
            resources[i][1] = numMultipliers;
        }
    }


    int& get(int step, int type) {
        return resources[step][type];
    }

    void use(int step, int type, int count) {
        if (step + count > resources.size()) {
            cerr << "Error: Attempting to use resources beyond the preallocated steps." << endl;
            return;
        }

        for (int i = step; i < step + count; ++i) {
            resources[i][type]--;
            if (resources[i][type] < 0) {
                cerr << "Warning: Resources went negative at step " << i << " for type " << (type == 0 ? "Adder" : "Multiplier") << "." << endl;
            }
        }
    }
};

struct ScheduleEntry {
    int step;
    int opIndex;
};
```

實驗過程-程式碼說明

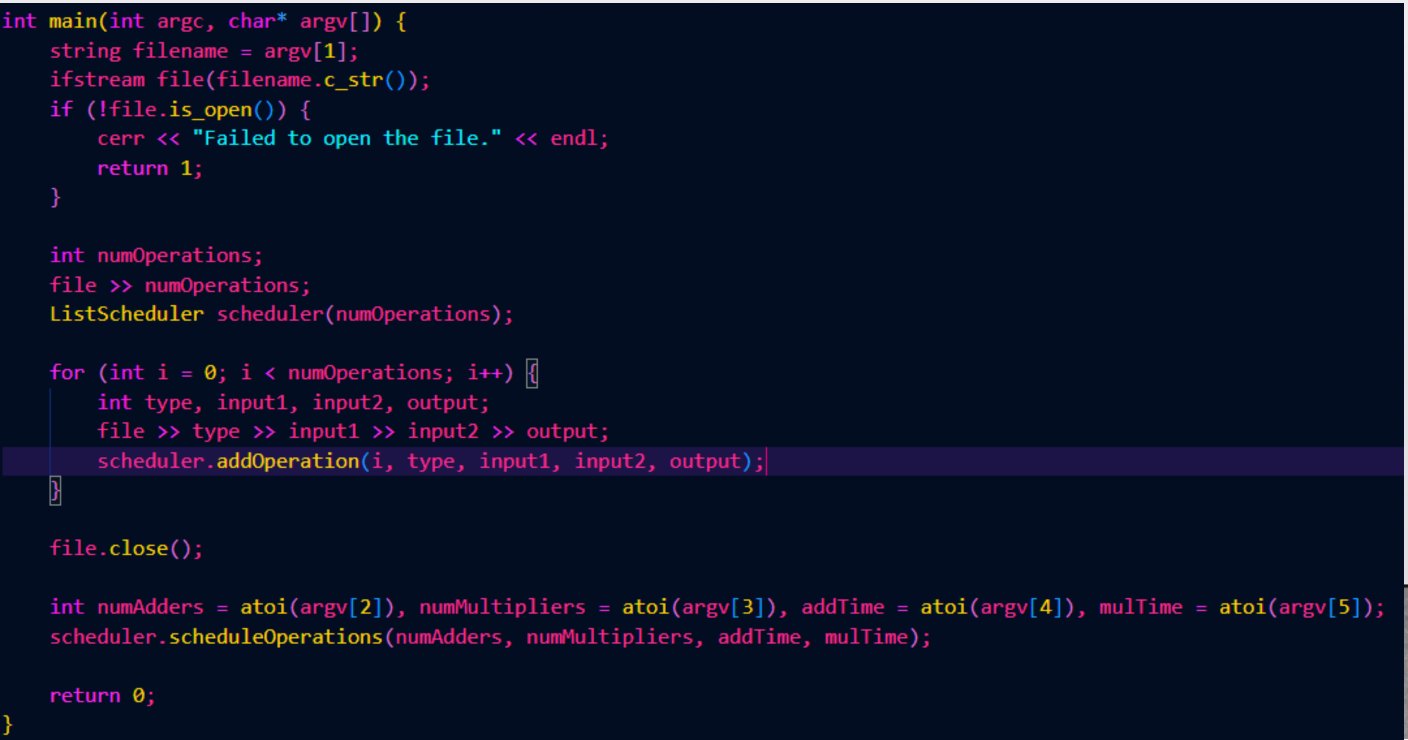
接下來我去定義 class 來完成此次實驗的主要排程接下來，主要利用 scheduleOperations 這個函數進行排程，接收的參數有加法器、乘法器的數量以及加法、乘法所需的時間。主要 loop 測資中的所有指令，對每一個指令找出他能執行的最早 step，找的方式是先去遞回已經排進去 schedule table 裡面之前的指令，去檢查有沒有 data consistency 的問題，接著探討該 step 下是否有足夠的硬件可以提供該運算，針對這兩種情況，正確地設置該指令可以執行的最早 step，接著更新 ALU 的狀態，以及進行排序後印出。



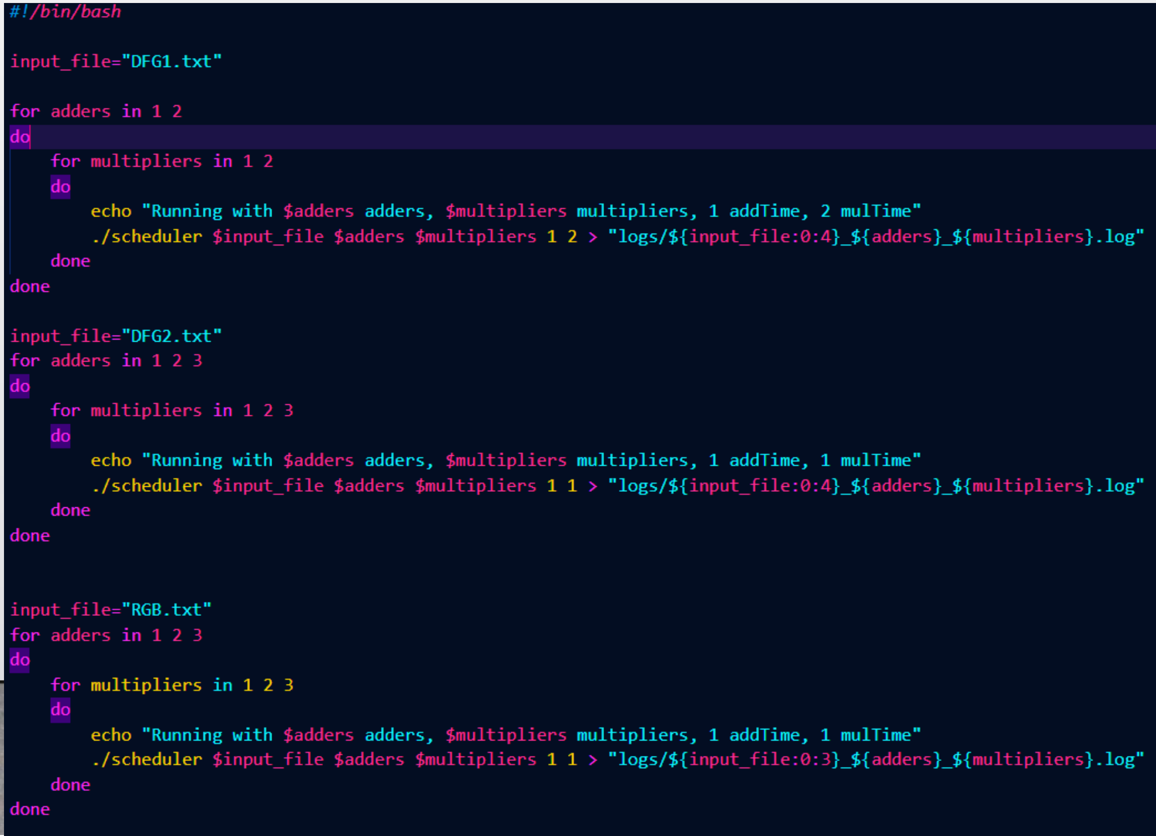
```
class ListScheduler{

    void addOperation(int index, int type, int input1, int input2, int output) {
        operations[index] = {type, input1, input2, output};
    }

    void scheduleOperations(int numAdders, int numMultipliers, int addTime, int mulTime) {
        ALU alu(numAdders, numMultipliers, 100);

        vector<ScheduleEntry> schedule;
        for (size_t i = 0; i < operations.size(); ++i) {
            int earliestStep = findEarliestStep(i, schedule, addTime, mulTime, alu);
            schedule.push_back({earliestStep, static_cast<int>(i)});
            updateALU(alu, operations[i].type, earliestStep, addTime, mulTime);
        }
        sort(schedule.begin(), schedule.end(), CompareScheduleEntry());
        for (size_t i = 0; i < schedule.size(); ++i) {
            ScheduleEntry& entry = schedule[i];
            Operation& op = operations[entry.opIndex];
            cout << "Step " << entry.step << ": Perform operation " << op.type
                 << " on inputs " << op.input1 << " and " << op.input2
                 << " to produce output " << op.output << endl;
        }
    }

private:
    vector<Operation> operations;

    int findEarliestStep(int opIndex, const vector<ScheduleEntry>& schedule, int addTime, int mulTime, ALU& alu) {
        int earliestStep = 1; // initial step
        int operationType = operations[opIndex].type;
        int typeIndex = (operationType == 1 ? 0 : 1);  // 0 for adders, 1 for multipliers

        for (size_t j = 0; j < schedule.size(); ++j) {
            const ScheduleEntry& entry = schedule[j];
            const Operation& previousOp = operations[entry.opIndex];
            if (previousOp.output == operations[opIndex].input1 ||
                previousOp.output == operations[opIndex].input2) {
                int operationTime = (previousOp.type == 1 ? addTime : mulTime);
                earliestStep = max(earliestStep, entry.step + operationTime);
            }
        }
        while (alu.get(earliestStep, typeIndex) <= 0) {
            //cout << "Stuck at step: " << earliestStep << " for type: " << (typeIndex == 0 ? "Adder" : "Multiplier") << " with available: " << alu.get(earliestStep, typeIndex) << endl;
            earliestStep++;
        }

        return earliestStep;
    }


    void updateALU(ALU& alu, int operationType, int startStep, int addTime, int mulTime) {
        int operationTime = (operationType == 1 ? addTime : mulTime);
        int typeIndex = (operationType == 1 ? 0 : 1);  // 0 for adders , 1 for multipliers
        alu.use(startStep, typeIndex, operationTime);
    }

};
```

實驗過程-程式碼說明

最後我主要是寫一個 script 去跑實驗，透過給予參數來跑程式，左圖為程式中的 main function ，接收的參數有檔案名稱，加法器、乘法器的數量以及加法、乘法所需的時間。主程式主要讀入檔案，簡單呼叫 scheduleOperations 這個函數進行排程。右圖為 script 主要透過回圈遍歷加法器乘法器數量的不同組合並跑三個檔案,這裡註記 DFG1 使用的加法所需時間是 1、乘法所需時間為 2，而 DFG2 及 RGB 的加法、乘法所需時間皆為 1。



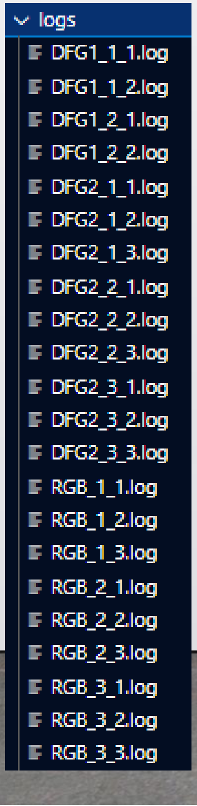
```cpp
int main(int argc, char* argv[]) {
    string filename = argv[1];
    ifstream file(filename.c_str());
    if (!file.is_open()) {
        cerr << "Failed to open the file." << endl;
        return 1;
    }

    int numOperations;
    file >> numOperations;
    ListScheduler scheduler(numOperations);

    for (int i = 0; i < numOperations; i++) {
        int type, input1, input2, output;
        file >> type >> input1 >> input2 >> output;
        scheduler.addOperation(i, type, input1, input2, output);
    }

    file.close();

    int numAdders = atoi(argv[2]), numMultipliers = atoi(argv[3]), addTime = atoi(argv[4]), mulTime = atoi(argv[5]);
    scheduler.scheduleOperations(numAdders, numMultipliers, addTime, mulTime);

    return 0;
}
```



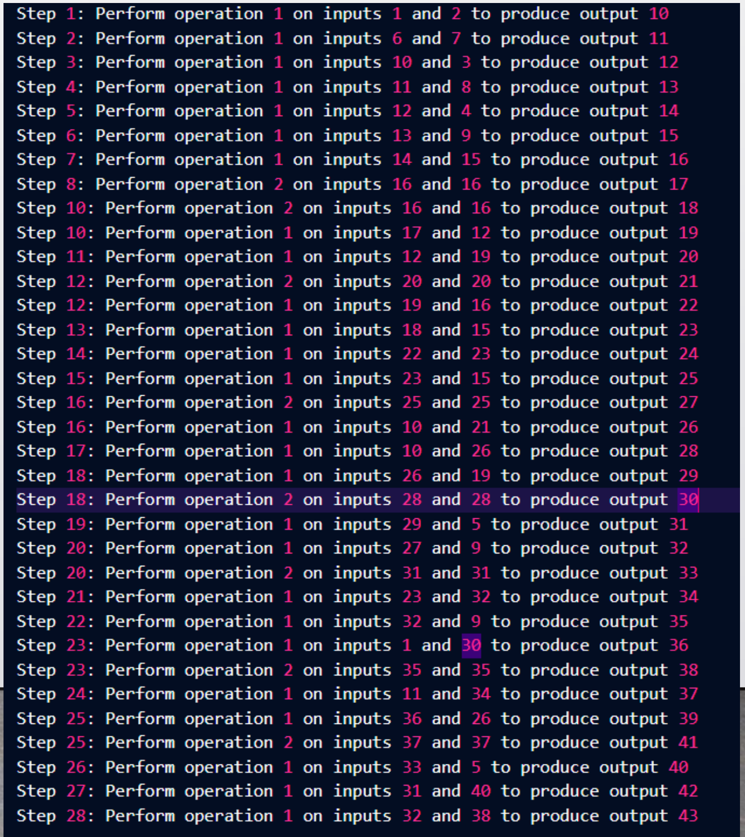
```bash
#!/bin/bash

input_file="DFG1.txt"

for adders in 1 2
do
    for multipliers in 1 2
    do
        echo "Running with $adders adders, $multipliers multipliers, 1 addTime, 2 mulTime"
        ./scheduler $input_file $adders $multipliers 1 2 > "logs/${input_file:0:4}_${adders}_${multipliers}.log"
    done
done

input_file="DFG2.txt"
for adders in 1 2 3
do
    for multipliers in 1 2 3
    do
        echo "Running with $adders adders, $multipliers multipliers, 1 addTime, 1 mulTime"
        ./scheduler $input_file $adders $multipliers 1 1 > "logs/${input_file:0:4}_${adders}_${multipliers}.log"
    done
done


input_file="RGB.txt"
for adders in 1 2 3
do
    for multipliers in 1 2 3
    do
        echo "Running with $adders adders, $multipliers multipliers, 1 addTime, 1 mulTime"
        ./scheduler $input_file $adders $multipliers 1 1 > "logs/${input_file:0:3}_${adders}_${multipliers}.log"
    done
done
```

實驗結果及分析

我對每個情況的執行結果都輸出到檔案中，存放在 logs 此資料夾底下，命名方式為“檔名_加法器數量_乘法器數量.log”，如左圖。每個檔案的輸出一個 step 執行什麼形態的運算、兩個輸入以及輸出，如右圖。



```
logs
  DFG1_1_1.log
  DFG1_1_2.log
  DFG1_2_1.log
  DFG1_2_2.log
  DFG2_1_1.log
  DFG2_1_2.log
  DFG2_1_3.log
  DFG2_2_1.log
  DFG2_2_2.log
  DFG2_2_3.log
  DFG2_3_1.log
  DFG2_3_2.log
  DFG2_3_3.log
  RGB_1_1.log
  RGB_1_2.log
  RGB_1_3.log
  RGB_2_1.log
  RGB_2_2.log
  RGB_2_3.log
  RGB_3_1.log
  RGB_3_2.log
  RGB_3_3.log
```



```
Step 1: Perform operation 1 on inputs 1 and 2 to produce output 10
Step 2: Perform operation 1 on inputs 6 and 7 to produce output 11
Step 3: Perform operation 1 on inputs 10 and 3 to produce output 12
Step 4: Perform operation 1 on inputs 11 and 8 to produce output 13
Step 5: Perform operation 1 on inputs 12 and 4 to produce output 14
Step 6: Perform operation 1 on inputs 13 and 9 to produce output 15
Step 7: Perform operation 1 on inputs 14 and 15 to produce output 16
Step 8: Perform operation 2 on inputs 16 and 16 to produce output 17
Step 10: Perform operation 2 on inputs 16 and 16 to produce output 18
Step 10: Perform operation 1 on inputs 17 and 12 to produce output 19
Step 11: Perform operation 1 on inputs 12 and 19 to produce output 20
Step 12: Perform operation 2 on inputs 20 and 20 to produce output 21
Step 12: Perform operation 1 on inputs 19 and 16 to produce output 22
Step 13: Perform operation 1 on inputs 18 and 15 to produce output 23
Step 14: Perform operation 1 on inputs 22 and 23 to produce output 24
Step 15: Perform operation 1 on inputs 23 and 15 to produce output 25
Step 16: Perform operation 2 on inputs 25 and 25 to produce output 27
Step 16: Perform operation 1 on inputs 10 and 21 to produce output 26
Step 17: Perform operation 1 on inputs 10 and 26 to produce output 28
Step 18: Perform operation 1 on inputs 26 and 19 to produce output 29
Step 18: Perform operation 2 on inputs 28 and 28 to produce output 30
Step 19: Perform operation 1 on inputs 29 and 5 to produce output 31
Step 20: Perform operation 1 on inputs 27 and 9 to produce output 32
Step 20: Perform operation 2 on inputs 31 and 31 to produce output 33
Step 21: Perform operation 1 on inputs 23 and 32 to produce output 34
Step 22: Perform operation 1 on inputs 32 and 9 to produce output 35
Step 23: Perform operation 1 on inputs 1 and 30 to produce output 36
Step 23: Perform operation 2 on inputs 35 and 35 to produce output 38
Step 24: Perform operation 1 on inputs 11 and 34 to produce output 37
Step 25: Perform operation 1 on inputs 36 and 26 to produce output 39
Step 25: Perform operation 2 on inputs 37 and 37 to produce output 41
Step 26: Perform operation 1 on inputs 33 and 5 to produce output 40
Step 27: Perform operation 1 on inputs 31 and 40 to produce output 42
Step 28: Perform operation 1 on inputs 32 and 38 to produce output 43
```

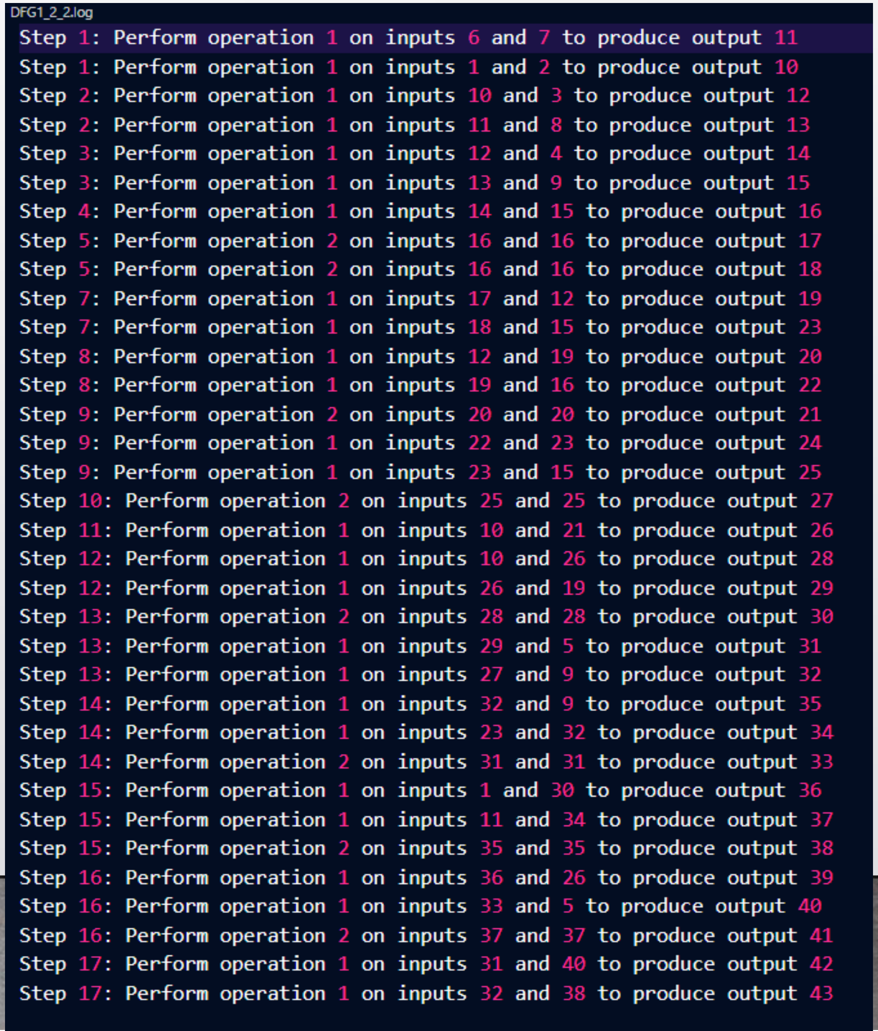
實驗結果及分析-DFG1

Resource Constraints:

+	*	Steps
1	1	28
1	2	28
2	1	19
2	2	17

我發現透過增加加法器，可以減少的 step 數較多。

DFG1_2_2.log:



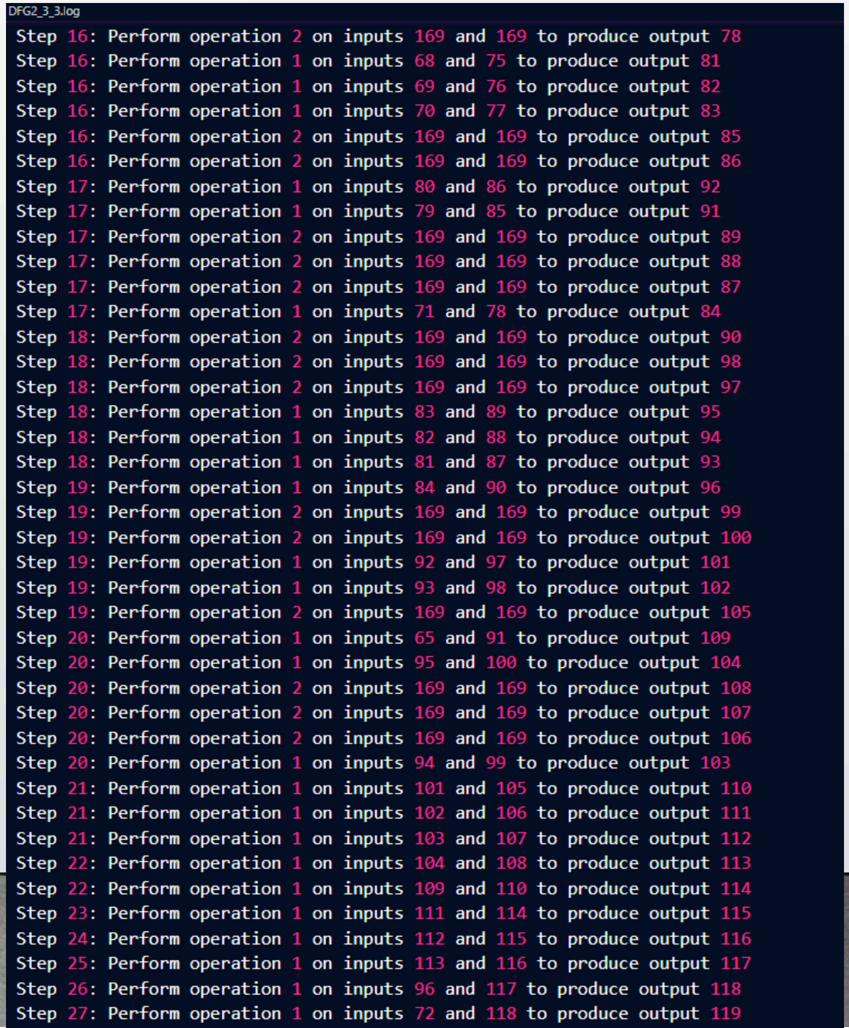
```
Step 1: Perform operation 1 on inputs 6 and 7 to produce output 11
Step 1: Perform operation 1 on inputs 1 and 2 to produce output 10
Step 2: Perform operation 1 on inputs 10 and 3 to produce output 12
Step 2: Perform operation 1 on inputs 11 and 8 to produce output 13
Step 3: Perform operation 1 on inputs 12 and 4 to produce output 14
Step 3: Perform operation 1 on inputs 13 and 9 to produce output 15
Step 4: Perform operation 1 on inputs 14 and 15 to produce output 16
Step 5: Perform operation 2 on inputs 16 and 16 to produce output 17
Step 5: Perform operation 2 on inputs 16 and 16 to produce output 18
Step 7: Perform operation 1 on inputs 17 and 12 to produce output 19
Step 7: Perform operation 1 on inputs 18 and 15 to produce output 23
Step 8: Perform operation 1 on inputs 12 and 19 to produce output 20
Step 8: Perform operation 1 on inputs 19 and 16 to produce output 22
Step 9: Perform operation 2 on inputs 20 and 20 to produce output 21
Step 9: Perform operation 1 on inputs 22 and 23 to produce output 24
Step 9: Perform operation 1 on inputs 23 and 15 to produce output 25
Step 10: Perform operation 2 on inputs 25 and 25 to produce output 27
Step 11: Perform operation 1 on inputs 10 and 21 to produce output 26
Step 12: Perform operation 1 on inputs 10 and 26 to produce output 28
Step 12: Perform operation 1 on inputs 26 and 19 to produce output 29
Step 13: Perform operation 2 on inputs 28 and 28 to produce output 30
Step 13: Perform operation 1 on inputs 29 and 5 to produce output 31
Step 13: Perform operation 1 on inputs 27 and 9 to produce output 32
Step 14: Perform operation 1 on inputs 32 and 9 to produce output 35
Step 14: Perform operation 1 on inputs 23 and 32 to produce output 34
Step 14: Perform operation 2 on inputs 31 and 31 to produce output 33
Step 15: Perform operation 1 on inputs 1 and 30 to produce output 36
Step 15: Perform operation 1 on inputs 11 and 34 to produce output 37
Step 15: Perform operation 2 on inputs 35 and 35 to produce output 38
Step 16: Perform operation 1 on inputs 36 and 26 to produce output 39
Step 16: Perform operation 1 on inputs 33 and 5 to produce output 40
Step 16: Perform operation 2 on inputs 37 and 37 to produce output 41
Step 17: Perform operation 1 on inputs 31 and 40 to produce output 42
Step 17: Perform operation 1 on inputs 32 and 38 to produce output 43
```

實驗結果及分析-DFG2

Resource Constraints:

+	*	Steps
1	1	68
1	2	64
1	3	62
2	1	64
2	2	37
2	3	36
3	1	64
3	2	36
3	3	27

DFG2_3_3.log:



```
Step 16: Perform operation 2 on inputs 169 and 169 to produce output 78
Step 16: Perform operation 1 on inputs 68 and 75 to produce output 81
Step 16: Perform operation 1 on inputs 69 and 76 to produce output 82
Step 16: Perform operation 1 on inputs 70 and 77 to produce output 83
Step 16: Perform operation 2 on inputs 169 and 169 to produce output 85
Step 16: Perform operation 2 on inputs 169 and 169 to produce output 86
Step 17: Perform operation 1 on inputs 80 and 86 to produce output 92
Step 17: Perform operation 1 on inputs 79 and 85 to produce output 91
Step 17: Perform operation 2 on inputs 169 and 169 to produce output 89
Step 17: Perform operation 2 on inputs 169 and 169 to produce output 88
Step 17: Perform operation 2 on inputs 169 and 169 to produce output 87
Step 17: Perform operation 1 on inputs 71 and 78 to produce output 84
Step 18: Perform operation 2 on inputs 169 and 169 to produce output 90
Step 18: Perform operation 2 on inputs 169 and 169 to produce output 98
Step 18: Perform operation 2 on inputs 169 and 169 to produce output 97
Step 18: Perform operation 1 on inputs 83 and 89 to produce output 95
Step 18: Perform operation 1 on inputs 82 and 88 to produce output 94
Step 18: Perform operation 1 on inputs 81 and 87 to produce output 93
Step 19: Perform operation 1 on inputs 84 and 90 to produce output 96
Step 19: Perform operation 2 on inputs 169 and 169 to produce output 99
Step 19: Perform operation 2 on inputs 169 and 169 to produce output 100
Step 19: Perform operation 1 on inputs 92 and 97 to produce output 101
Step 19: Perform operation 1 on inputs 93 and 98 to produce output 102
Step 19: Perform operation 2 on inputs 169 and 169 to produce output 105
Step 20: Perform operation 1 on inputs 65 and 91 to produce output 109
Step 20: Perform operation 1 on inputs 95 and 100 to produce output 104
Step 20: Perform operation 2 on inputs 169 and 169 to produce output 108
Step 20: Perform operation 2 on inputs 169 and 169 to produce output 107
Step 20: Perform operation 2 on inputs 169 and 169 to produce output 106
Step 20: Perform operation 1 on inputs 94 and 99 to produce output 103
Step 21: Perform operation 1 on inputs 101 and 105 to produce output 110
Step 21: Perform operation 1 on inputs 102 and 106 to produce output 111
Step 21: Perform operation 1 on inputs 103 and 107 to produce output 112
Step 22: Perform operation 1 on inputs 104 and 108 to produce output 113
Step 22: Perform operation 1 on inputs 109 and 110 to produce output 114
Step 23: Perform operation 1 on inputs 111 and 114 to produce output 115
Step 24: Perform operation 1 on inputs 112 and 115 to produce output 116
Step 25: Perform operation 1 on inputs 113 and 116 to produce output 117
Step 26: Perform operation 1 on inputs 96 and 117 to produce output 118
Step 27: Perform operation 1 on inputs 72 and 118 to produce output 119
```

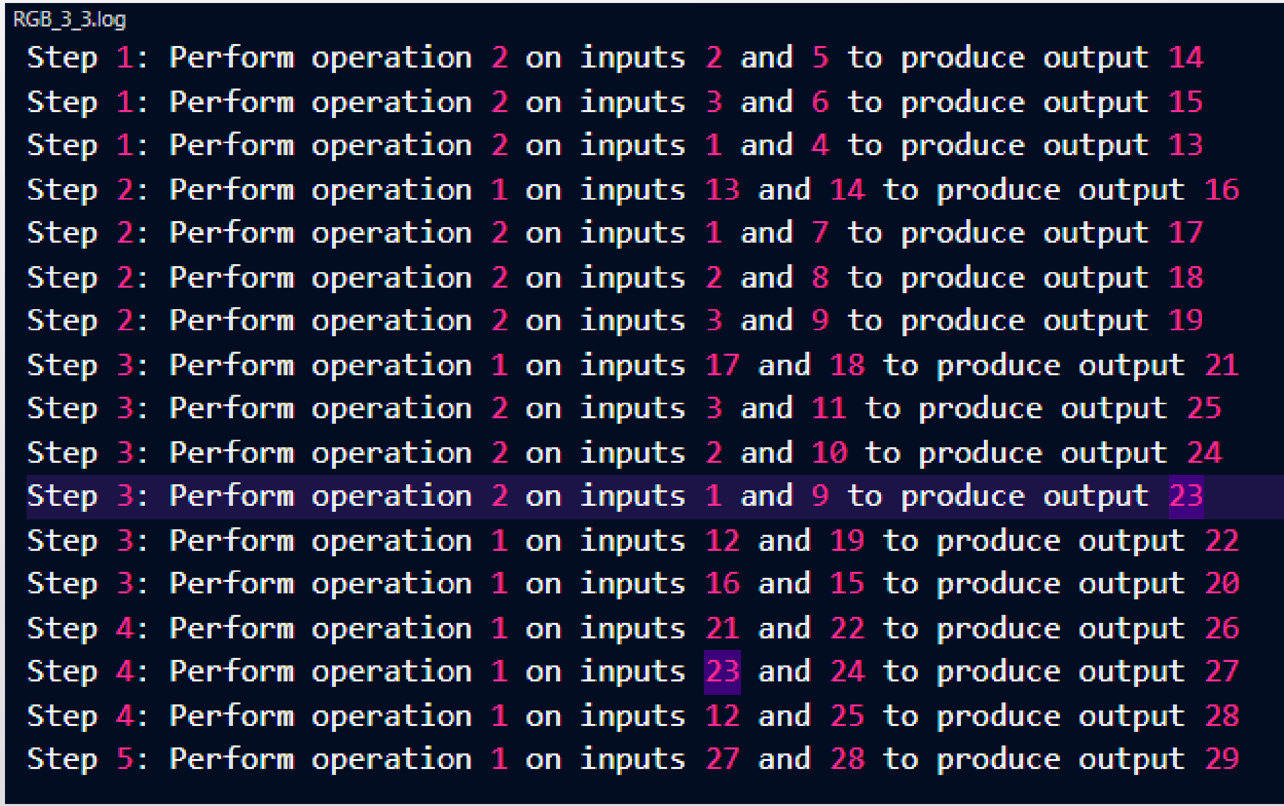
我發現透過在有2個加法器以上的情況下，增加乘法器會使steps 數下降較多。

實驗結果及分析-RGB to YUV

Resource Constraints:

+	*	Steps
1	1	11
1	2	9
1	3	9
2	1	11
2	2	7
2	3	6
3	1	11
3	2	7
3	3	5

RGB_3_3.log:


RGB_3_3.log

```
Step 1: Perform operation 2 on inputs 2 and 5 to produce output 14
Step 1: Perform operation 2 on inputs 3 and 6 to produce output 15
Step 1: Perform operation 2 on inputs 1 and 4 to produce output 13
Step 2: Perform operation 1 on inputs 13 and 14 to produce output 16
Step 2: Perform operation 2 on inputs 1 and 7 to produce output 17
Step 2: Perform operation 2 on inputs 2 and 8 to produce output 18
Step 2: Perform operation 2 on inputs 3 and 9 to produce output 19
Step 3: Perform operation 1 on inputs 17 and 18 to produce output 21
Step 3: Perform operation 2 on inputs 3 and 11 to produce output 25
Step 3: Perform operation 2 on inputs 2 and 10 to produce output 24
Step 3: Perform operation 2 on inputs 1 and 9 to produce output 23
Step 3: Perform operation 1 on inputs 12 and 19 to produce output 22
Step 3: Perform operation 1 on inputs 16 and 15 to produce output 20
Step 4: Perform operation 1 on inputs 21 and 22 to produce output 26
Step 4: Perform operation 1 on inputs 23 and 24 to produce output 27
Step 4: Perform operation 1 on inputs 12 and 25 to produce output 28
Step 5: Perform operation 1 on inputs 27 and 28 to produce output 29
```

我發現增加乘法器會使 steps 數下降較多，在乘法器數量都為 1 的時候，增加加法器都沒有使 steps 數下降,推測原因為每個顏色第一步都需要做三個乘法，因此在此情況下，我認為乘法器是影響 steps 多寡的重要因素。

實驗心得

此次實驗本以為有架構程式可以參考會比較快完成，但實際在寫的時候，發現其實沒有那麼簡單，一開始我花了很多時間再想要如何去排程，要如何精準地計算出在哪個 step 下才能執行那個指令、如何去處理 data consistency 的問題，在記錄 ALU 狀態的二維 vector 也花了較多的時間去 debug，才成功記錄正確，不然一開始似乎因為沒有好好記錄導致讀不到可用的硬件，導致陷入無窮回圈，透過此次實驗，我更了解了排程的實際運作模式，以及進行排程時需要考慮的問題。