

Data Structure and Programming – Final Project

FRAIG

B06602037 徐子程

E-Mail : b06602037@ntu.edu.tw

Phone : 0918800688

A. Design of the data structure

1. CirGate (base class)

Stored information

- `unsigned _lineNo`
Line number in aag file
- `uint8_t _typeID:`
對應到 enum 定義的 derivative gate types
- `unsigned _id`
Literal gate ID
- `string *_symbol`
symbol string pointer of the gate (if symbol not exist , == NULL)
- `CirGate *_fin[2]`
Gate pointer array to store 2 fanins that connected to the gate.
- `bool _inv[2]`
Bool array to store 2 fanin inverting state.
- `vector<CirGate *> _outList`
dynamic array to store fanout gates
- `size_t _simVal`
simulation value
- `mutable unsigned _ref`
Current reference count for DFS traversal
- `static unsigned _globalRef`
Global reference count for DFS traversal

2. Derivative gate types (using inheritance)

- CONST : CONST0 gate
- UNDEF : undefined gate
- AIG : and-inverting date
- PI : input gates
- PO : output gates

B. Feature implementation

1. Command “CIRSweep”

Sweep 要移除的 gate 是除了 PI 之外的 Undefined, Unused 和的 gate，換句話說，它們不會被 DFS 走到，因此不會出現在 `_dfsList` 裡面。只要比較 `_dfsList` 內的 gate 和所有 gate List，找出沒有出現在 `_dfsList` 裡的 gate，刪除之，就完程 sweep。

由於在我的實做方式中，所有 gate 的 list(`vector<unsigned>_gateList`)，是按 gate ID 排序，並初始化成足以容納所有 gate，故 array 的 index 為 gate ID，可以進行 random access，時間複雜度為 $O(1)$ 。

但 `_dfsList` 是按 DFS traversal 順序進行 push back，故內部沒有按 gate ID 排序，若要在 `_gateList` 內按 ID 順序尋找是否同樣的 gate 存在 `_dfsList` 內，對於每個 gate，需要對 `_dfsList` 做 linear search($O(n)$)或 binary search($O(\log n)$)。走完一遍 `_gateList` 總共需要花費 $O(n^2)$ 或 $O(n \log n)$ ，實在不是一個好辦法。

故改建一個 bool array，capacity 與 `_gateList` 一樣大，走過一遍 `_dfsList` 將所有 ID mapping 至 array($O(n)$)。再走過一遍 `_gateList`($O(n)$)就可以知道哪些 gate 是應該要刪除的。時間複雜度為 $O(2n)$ ，維持在 $O(n)$ 等級。

Pseudo code:

```
for i in _dfsList
    dfs_array[i] = true
for i in _gateList
    if dfs_array[i] && _gateList[i] != NULL
        removeGate(i)
```

1. Command “CIROptimize”

Optimize 算是比較複雜的操作，因為處理不同 case 之間 gate 連接的關係，花了不少時間 debug。

實做上是按 DFS 順序逐一檢查符合條件的 gate，並刪除，重走一遍 DFS，更新 DFS List。

2. Command “CIRStrash”

使用 STL 內建 unordered_map 作為 hash map 使用，建立 unordered_map<size_t, CirGate *>，型態儲存 fanin pair，其中 hash key 為自行設計的 hash function 產生。

Pseudo code:

```
size_t finHashKey(CirGate *g)
    bool sel = false
    if (g->fin2 < g->fin1) sel = true
    a = g->fin[sel]->id
    b = g->fin[!sel]->id
    return (a << 32) + (b << 2) + (g->_inv[!sel] << 1)
        + g->_inv[sel]
```

3. Command “CIRSimulate”

FileSim 運作流程：

- (1) 逐行讀檔
- (2) 檢查字串內容字否合法
- (3) 將字串轉為 bits，推入每個 PI gate
- (4) 每讀 64 行，進行 simulation（一次 64bit）
- (5) 找出 FEC groups，並與前次 simulation 的 group 比較，拆分之
- (6) 清空 simulation 結果
- (7) 寫檔（若有 -o 指令），每次 64 行
- (8) 對 FEC groups 排序
- (9) 關閉檔案

其中 simulation 使用 STL 的 unordered_map<size_t, vector<unsigned>> 儲存產生的 FEC groups，vector<unsigned> 儲存 gate ID 及相位（方法同 aag 檔定義）。排序則新增一個 vector<vector<unsigned>*>，指向 unordered map 中的 vector<unsigned>* 以節省記憶體。

Pseudo code：

```
while (patternFile >> line)
    checkPattern() // Check input pattern
    simPI() // Perform simulation to PI gates
    if (lineNumber %64 == 0) // Simulate every 64 lines
        translationPatterns() // Keep input patterns
        simAllGate() // Simulate gates after PI
        constructFEC() // find FEC groups
        clearResults() // Clear simulated value

translationPatterns()
simAllGate()
constructFEC()
clearResults()
```

CIRPrint -FECpairs :

將 `vector<vector<unsigned>*>` 裡已經排序好的 FEC groups 依序印出。

CIRGate <<(int gateId)> :

其中 FECs 是利用當前 gate 的 sim value 查詢

`unordered_map<size_t, vector<unsigned>>`，回傳 array 的 pointer，並 print 出裡面的內容。達到共享以節省 memory。

遇到的困難：

因為讀檔的順序與 sim value 的方向相反（讀檔的上往下對應到 bit 由左往右讀），使得處理 input pattern 得部份變得複雜。需要將 input pattern 依序推進 PI gate，並且要注意是否超出範圍。

由於先前提過改用 `size_t` 紀錄 input 和 output pattern，要轉換成 output file 又是浩大工程，花費不少時間在研究如何將 simulation 結果對應到 output file 格式。

後面的區分 FEC groups 更是具有挑戰性，在重複 simulation 需要與前一次的結果進行比較，並拆成更小的 Group，這個操作到目前為止仍有 bug，看來需要捲土重來了。

效能改善：

記憶體：

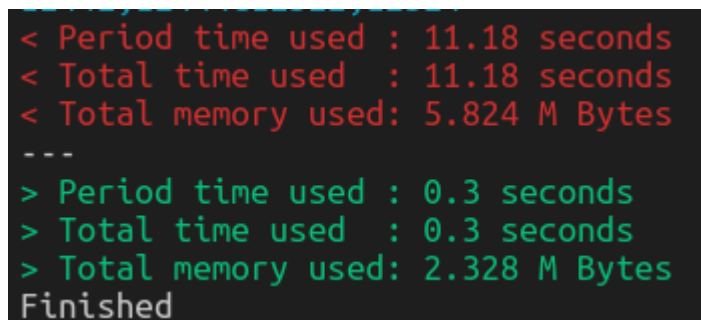
原先使用 `vector<string>` 紀錄讀進來的 input pattern，以 sim12.aag 測試，結果需要使用約 8X MB 的 memory。後來改以 `vector<size_t>` 儲存，充分利用每個 bit，並且每 simulation 一次就寫入 filestream，不用紀錄所有 input 和 output pattern。經過測試，memory 使用約減少一半。

另外，先前是使用兩個 hash map 儲存 FEC groups，後將每次 simulation 完成後的結果覆蓋掉原本的 hash map。

時間：

原先是每次找完 FEC groups 後將 hash map 內每個 ID array sort 一遍，後改為全部 simulate 過後才進行 sort。測試後 sim12.aag 由花費 7X 秒減少至 1X 秒。

再經過部份調整後，目前效能已有大幅改善（下圖未用-O3 編譯）。



```
< Period time used : 11.18 seconds
< Total time used   : 11.18 seconds
< Total memory used: 5.824 M Bytes
- - -
> Period time used : 0.3 seconds
> Total time used   : 0.3 seconds
> Total memory used: 2.328 M Bytes
Finished
```

上：測試結果，下：ref

4. Command “CIRFraig”

沒有做完 QQ

C. 心得

這學期花了不少時間在這門課上面，如同開學第一堂課說的，為了寫作業也拿了好多肝來換。不只是程式，生活中的時間管理是重要的。

我認為整學期學習到最多的，是要如何架構一個大型程式。在此之前，雖然高中就已經會寫 C/C++，不過這學期讓我重新認識物件導向程式設計，之前學的相形之下都不算什麼。一方面也是因為有許多大神同學，讓我可以向他們請教。看見自己的不足

還有指標是如何使用的，以往對於指標都是一知半解，寫了作業之後逐漸感受到指標的方便，也對記憶體管理有較多的認識。

這學期的課程進度有些 delay，希望之後能更好的掌控課程進度，才能讓同學們較充足的時間寫作業。

最後要感謝所有幫助我的同學們，和老師的指導，I ♥ DSnP。