資料結構與程式設計

DSnP Final Project 〖Fraig〗

系級：電機三

姓名：蔡秉珈

學號：B99901157

聯絡資料：手機 0931788675

一、 Data Structure

1. CirMgr

當讀進一個電路時，我的 CirMgr 存了 M、I、L、O、A 的值，這幾個值可以讓寫 aag file、print 時更節省時間，用一個 int 的記憶體大小去換取計算數目的時間。

除了基本的值，我的 CirMgr 也存了幾個 list，分別是根據 gate ID 排序的 list、PI list、PO list、floating gate list 以及 FEC groups 的 list，這些 list 記錄下這個程式中可能需要的資料，例如 DFS 的結果、需要用的 gate ID 等等，也是利用記憶體空間去換取操作的時間。

2. Gate

用繼承 CirGate 這個 object 的方式產生不同的 gate，在我的設計中有 Pi、Po、And、Const0、Undef 此五種。

CirGate 中各種 gate 共同的 data member 都存在 CirGate 中，如 ID、fanout list 等，根據對每種 gate 可能有不同的操作 (Pi 與 And 有不同 fanin 數)，此 gate 存有許多 virtual funtion。

3. srash function (Hash)

在 strash 這個 function 中，必須把有相同 fanin 的 AND gate 化簡，過程中使用到了" Hash" 這個資料結構。簡單的流程就是，將 DFS list 中的 gate 依照 fanin 分類，存進 Hash 中，若在存取過程發現已經有同樣分類的 gate，即可得知此 gate 必須被已經在 Hash 中的 gate merge。

a. 建構空間

在開始存取資料前，必須先建構一個足夠的空間以存取資料，至於這個空間有多大，取決於使用者。這個 Hash 中，有許多 bucket，這些 bucket 分別是一個 vector 可以存取數個資料，而這些 bucket 本身是個 array，所以 Hash 的結構

相當於一個二維的陣列，但 Hash 本身的 data member 存的是 vector array 的 pointer。前面提到的空間大小，即是這個 array 的大小，當一個 Hash 被建構時即須告訴他所要新增的大小。

在我的程式中，Hash 的大小是用 util.cpp 檔中 getHashSize 取得，傳入的數字是 DFS list 中 gate 的數目除以二，傳回來的值會是一個比傳入值大的質數。Bucket 的數目為質數，可讓資料更平均分配(見下一點)。

b. 資料分類

有了空間之後，接下來必須將資料存進空間中，存在 Hash 中的資料是"HashNode"，HashNode 是 pair<HashKey, HashData>，以下簡述這兩個資料。

i. HashKey

每次要用 Hash 存取資料，就要做一個 HashKey 的物件，此物件中必須 overload "()" 以及"==" ，()指的是 Hash function，目的在於透過這個 function 可傳出所對應到存取的 bucket 的 index，所以前面提到的 bucket 數目選取質數，即是為了在 mod 時能平均每個 bucket 中的資料數；而==是為了比較同類型的資料，在存取資料時我們會用自己的方式去認定每筆資料是否相同，透過自己設計的 hash，可以用自己的方法去判別兩筆資料是否相同，像在 strash 中，所用的判別方法就是比較 fanin 是否相同。

ii. HashData

HashData 指的是透過比較資料之後，使用者想得到的資料，例如在 strash 中，我透過比較 hashkey 後希望得到相同 fanin 的 CirGate*，所以我就將其設為 HashData。

c. Member function

i. bool check(const HashKey& k, HashData& n)

確認 Hash 中是否已存有 k，若有，則會將 n 改成所存在的 Hashkey 所對應到的 HashData，讓使用者取得 Data。

ii. bool insert(const HashKey& k, const HashData& d)

先確認 Hash 中是否已有 HashNode<k, d>(比較 HashKey)，若沒有則新增一個 HashNode(k, d)存進 Hash 中。

iii. bool replaceInsert(const HashKey& k, const HashData& d)

確認 Hash 中是否已有 HashKey k，若有，則將其對應到的 HashData 改為 d。

iv. void forceInsert(const HashKey& k, const HashData& d)

不管資料是否已經存在 Hash 中，強制新增一個 HashNode(k, d)存進 Hash 中。

d. 操作

當我要利用 strash 化簡電路時，會先 new 出一個 Hash，接著從 DFS list 中，將每個 gate 製作出一個 HashKey，透過前面說過的 check()及 forceInsert()，先確認是否有可 merge 的 gate，若有，check()則會將傳進去的 CirGate*& 改成可用來 merge 的 gate，接著就能將沒有被存進去的這個 gate merge；若沒有找到一樣 fanin 的 gate，則將這個 gate 透過 HashNode 存進 Hash 中。

另外值得一提的是，merge 的過程必須從 fanin 開始，因為每經過一次 merge，往 Po 方向就會有 gate 的 fanin 被改掉，會影響接下來 strash 的結果；另外一個原因，是因為若從 Po 開始往 Pi 方向 merge，可能會讓電路產生 loop，這是我們所不能處理的。

```
CirGate* temp;
HashKey key;

for all AIG in _netList
    key = HashKey((And*)_netList[i]);
    if(myHash->check(key, temp))
        mergeGate(); //use temp to merge the gate
    else
        myHash->forceInsert(key, _netList[i]);
```

e. 優點

在 Hash 中要找資料是要透過 HashKey，因為 Hash function 所花的時間為 constant time，所以不同於 array 或是 tree，每個 HashKey 去找到屬於他的 bucket 時都是固定時間的，再透過打亂順序存進不同的 bucket，每個 bucket 中的資料並不多，比較起來也能花很少時間，Hash 透過這樣利用大容量的方式，達到快速的找到資料，也能自己設計比較資料的機制和所想要取得的資料內容。

二、 Algorithm

在化簡電路時，使用了四種方法，下面會一一敘述其演算法。

在開始敘述這次種功能前，必須先了解到這四個化簡電路的方法其實都可以看做是用一個 gate 取代另一個 gate，所以我在 gate 裡面寫了一個被取代時要用的 function，傳入的值即是要取代自己的那個 gate。在這個 function 中，先處理自己的 fanout，將每個 fanout 的 fanin 改成傳入的 gate，同時也必須將傳入的 gate 的 fanout 新增自己的 fanout;接下來是處理自己的 fanin，必須將自己剔除於自己的 fanin 的 fanout list。

每次做化簡電路時，都是呼叫先將要被刪除的 gate 中上面提

到的 function，將新電路接好之後再把這個 gate 刪掉。進行這些動作的順序前面有提過，必須從 Pi 做到 Po，也就是按照 DFS list 的順序做，否則可能產生 loop。

1. sweep()

   這個功能是將 DFS list 以外的 gate 刪除，我的作法很直觀，先將 DFS list 走過一輪，用之前做 DFS 所使用過的 mark() 將這裡面的 gate 標記起來，接著再走過存有所有 gate 的 list，將沒有被標記的 gate 刪除，這個方法的好處是只需要走兩遍 list，也就是說複雜度是 2n，比起一個一個 gate 去檢查是否在 DFS list 中(n^2)要快許多。

2. optimize()

   這個功能是要化簡四種情況的 ANDgate，(1)fanin 中若有一個是 0，則輸出永遠是 0，(2)fanin 中若有一個是 1，則輸出永遠等於另一個 fanin，(3)若兩個 fanin 都是一樣的 gate，則輸出永遠等於這個 gate，(4)若兩個 fanin 分別為同一個 gate 及這個 gate 的 inverse，輸出則永遠等於 0。

   這四種情況分別代表這個 gate 可被(1)Const0、(2)另一個 fanin gate、(3)fanin gate、(4)Const0 這些 gate 取代，取代的方法就如前面所述，這個功能只需將 DFS list 走過一遍(n)，確認每個 ANDgate 的 fanin 即可。

3. strash()

   透過前面說明 Hash 的方法，將 DFS list 中的 gate 走過一輪(n)，找出同樣 fanin 的 gate，透過從 Hash check()得到的 data 去決定此 gate 是否要被取代以及要被哪個 gate 取代，取代的方法也如前面所述。

4. fraig()

   這個功能可分為兩個部分：simulate 以及 fraig。為了化

簡 functionally equivalent candidate pair，我們要使用 SAT 這個物件去計算出兩個 gate 是否是相等的，但一個電路中的 gate 數量龐大，不可能兩兩做比較，所以我們必須透過模擬的方式先將 gate 分類，同一個 group 中的 gate 即是"有可能"相等的 gate，利用 SAT 確定這些 gate 是否相等，再決定是否取代。

i. simulate

我的程式中，每個 gate 都有 simulate()這個 function 可以計算出模擬的值，每次模擬都從 Pi 開始輸入值，沿著 DFS list 的方向可以將每個 gate 輸出的值算出來並記在 gate 的 data member 中。

輸入 Pi 的值是隨機產生的，透過 parallel 的模擬方式，可以直接輸入一個 unsigned integer，也就是一次輸入 32 個 pattern，增加模擬的效率。

每 32 個 pattern 模擬完之後，必須比較一個 group 中哪些 gate 值一樣，將它們存在同一個 group 中，在這邊因為又必須分類，為了增加效率，我使用了前面提到的 Hash，但這邊的 HashKey"=="的條件式比較的是模擬完的值是否相同或是完全相反，如此才能同時找到 FEC pair 以及 iFEC pair。另外，這邊的 HashData 我選擇使用 GateList*，每個 GateList 存的就是一個 group。

ii. fraig

前面模擬完的結果會被我存在 CirMgr 中的_fecList 中，這個 list 存的是 GateList*，每個 GateList*都指向一個 group，對每個 group 中兩兩 gate 去做 SAT 的模擬，確認是否為 equivalent 或是 inverse equivalent，在進行取代的步驟，取代的方式也如前面所述。

三、 Feedback

修完這門課真的收穫很多，收穫之餘也想寫點心得當作回饋給老師。

其實我自己大一計程學得並不好，很多觀念都不清楚，像是 pointer、call by reference 之類的從來都搞不清楚，但在修資結的過程中慢慢的一點一點學，有時候會去翻之前計程的講義，真的突然就看懂了！原本寫程式就像在算數學一樣，想盡各種辦法達到目的就好，但從 HW1 到 Final，先是學會利用 function 去簡化程式，讓這個 function 可以被重複使用，接著是慢慢去體會物件導向的概念，將 data 包在一個 class 中去考慮如何讓外面的人使用，然後開始了解到程式的 performance，用 reference 去節省要記憶體的時間、new 和 delete 的意義……

以前寫程式就是想到什麼就往下拼命寫，寫出來的東西亂七八糟也不知道從何 debug，但現在寫程式都會先花很多時間去想整個架構，想到如何寫才能較輕鬆的 debug，如何用物件導向的概念去將 data 包起來自己使用以及怎麼讓外面的人使用而不做太複雜的操作，寫得過程也開始會去注意效率及記憶體，雖然這部分我還做得不是很好，但比起學期初真的進步很多。另外也學到了如何去看較大型的 code，較懂得去看別人怎麼去 implement 不同的程式，我覺得這也是很大的收穫。

這門課雖然真的很重，但也真的收穫很多，希望老師之後都能繼續開這門課，讓更多學弟妹更加認識程式，有更多的寫程式經驗，這學期辛苦老師了！謝謝老師！