資料結構與程式設計 期末專題Report

前言 :

在這份報告當中，我會先大概簡述每個指令我所用的演算法，而sim和fraig我則會特別提到我實驗各個演算法來增加效能的過程。另外，HW6部分的資料結構，我是參考陳泓廷(學號B05901101)的，這部分就不多做解釋。

Sweep :

Sweep command的部分，做法是先創一個ID的array來存所有有在DFSList裡面的gateID，再把這個array sort好。Sort好之後再去跟\_gateList比對，然後開一個GateList(vector)，只要沒有在那個array裡面的ID對應的gate就把他的pointer push進去，最後再一次處理。處理的方式就是先判斷他的gate型態是AIG還是UNDEF，如果是AIG就要處理fanout的問題(把fanin的fanout刪掉)，最後再把\_gateList裡面的pointer設為NULL，也就是這個gate已經被刪掉了。

Optimize :

至於optimize的部分，其實作法頗單純，就是在整個DFSList中找AIG gate，看他是否有符合可以optimize的四種情況；如果有符合，就先handle fanin和fanout的問題；方法是先把那個gate 兩個fanin的fanout都刪掉一個(那個gate)，然後接著把那個gate fanout的fanin改為要取代那個gate的input(const 或 aig)，最後把那個gate的input gate的fanout連到原本gate的fanout。處理完fanin,out之後再把\_gateList裡面的那個gate pointer設為NULL。因為有動到gate，所以DFSList要重新產生。

Strash :

Strash的部分，最主要就是要inplement HashMap，而我的作法所使用的key是一個包含兩個size\_t 的class，overload的()function就會讓第一個size\_t除以一個很大的質數，再%那個質數，相乘，然後第二個size\_t也做一樣的事，兩個再相加；而這樣做的目的是除了要讓她比較平均的分散的各個bucket，也要確保兩個size\_t的貢獻是一樣的，如此這樣fanin是ab和ba的情形才會塞到相同的bucket。而key overload的 == 則是確認他們是ab, ab或者是ab , ba的情況都算，也因此key相等就能夠確定兩個gate一定是equivalent。 HashMap處理完後，剩下的就是merge的時候處理fanin fanout問題，這邊就跟opt沒有相差太多，就不贅述。當然最後DFSList也要重新產生。

Simulation

再來是simulation。首先我寫了一個genRandSim的函式來處理random pattern generation的問題，方法是先rand一個數字，shift 32 個bit，再rand一個數字，兩者相加；原因是rand()的數字上限是32bit的大小。接著，每次的simulation是simulate64個bit，也就是如老師投影片裡的parallel simulation；每個PI\_GATE都simulate一個word之後，就依照電路(對兩個fanin去做bitwise&)把每個gate的值都simulate出來，每個gate都有一個\_simResult來存放每次的simulation value；以上的步驟，就能夠決定每個電路的一個simulation value。

而每次的simulation就看同個FEC group裡面，最新一次的simulation value是否相同，來確定他們是否還在同個FECgroup裡面。接下來就是要一個一個FEC group去弄到Hash裡面，最後再把分完的FEC group，也就是沒有落單的gate從Hash裡面拿出來，形成新的FEC group；每個FEC group重複一次這個動作。而我這裡HashKey的兩個size\_t則是用\_simResult和他的bit wise invert(~simResult)，這樣可以確保inverted value也會有同樣的Key。而HashData則是GateList，iterate到每個gate的時候，check那個gate造成的Key(也就是Value)有無再Hash裡面，若沒有就insert進去；如果有，就把裡面的GateList拿出來，然後再push\_back當時的那個gate，再update進去；最後做完整個FEC group，再把Hash裡面的Data，只要size()大於二就代表是FEC group，就先把它存到一個暫時的tempFecGrps裡面，最後全部FEC Groups都做完之後，再把tempFecGrps assign 給\_fecGrps，也就是存在cirMgr裡面的vector<GateList>。這樣就是一次64bit的simulation，而至於重複幾次後面會提到。

而在做完simulation之後，我會對\_fecGrps做sort(用vector的sort)，讓他按照順序排，以便print –fec時候是按照順序的。最後要print –fec的時候，因為是sort好的，所以直接印出來就好；然後要report gate的時候，就會印出最後一次sim的value，然後FEC pairs就從FEC group裡面去找，找到跟他一樣的再把那個group印出來。(因為是sort好的，所以找起來很快)

但我的simulation function效能不好，主因是tempFecGrps那邊做了一份copy，最後再把他assign給\_fecGrps，就需要花很多時間；而再Hash那邊要從Hash裡面一個一個檢查是不是size()大於二，再把它拿出來，也花了一些時間；也因此我這個寫法，效能會到達一個bottleneck，暫時無法解決；而也因為這樣我嘗試了其他方法，但不管怎麼寫好像都會有問題，因此為了繼續寫fraig只好先犧牲效能。

而至於決定random simulation怎麼停止的方法，則是考慮兩種limit，一個是跟Gate的多寡有關，另一個則是觀察如果FEC groups變動的幅度很小，就不再繼續往下做simulation；那我有設定一些參數，例如simulation的上限就會跟gate的多寡有關，而觀察gate變動幅度，幅度小的定義也是會根據gate的比例，因此會根據gate的多寡而有所不同。

另外還有一個很tricky的問題，不知道是否是我的演算法所致，Hash的bucketNum對效率的影響非常的顯著，原本一開始設定bucketNum是13999(按照老師HW6的設置)，結果效率十分低落，因此我一路測試從1399，599，一路測試到<100的數字，發現大約在59的效率最高，如果數字再小效率又會下降。但其實詳細的原因我並不是很清楚。

以下附上我做的小小實驗以及數據；我是對sim13去做simulation，看bucket大小對效能的影響。由於我先前作的實驗發現在100以下的效能最好，因此bucketNum在100以下的數據就不在這邊列出來。實驗結果如下表。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Size | 9 | 19 | 39 | 49 | 59 | 69 | 79 | 199 | 1999 |
| Time(s) | 23.35 | 22.95 | 15.93 | 15.53 | 15.35 | 15.64 | 15.37 | 18.1 | 66.79 |

由此可見，59附近是bucketNum的最佳值；而另外用比較大的bucketNum，也會導致比較大的記憶體用量，因此59算是個非常適當的數值。

Fraig

Fraig的部份，我的作法是把Var存在Gate裡面，而基本的做法則是從\_DFSList.去做iteration，從\_DFSList[0]出發，一個一個去看他的FEC group跟\_DFSList[i]有沒有一樣，一樣的話就用\_DFSList[i]去merge他，不一樣就不做事，最後再把自己從FEC group裡面拿掉，這樣FEC group裡面就只剩跟\_DFSList[i]不相等的gate；如果對\_DFSList整個做完一遍，可能會耗掉非常多的時間，也因此，我原本打算在收集一定數量的SAT之後，把導致SAT的pattern收集起來，再對剩下的FEC group做simulation，應該可以再分開一些group，simulation完再用SAT solver去證，如此重複直到分開所有FEC group。但我實作之後發現，可能由於我simulation的速度不夠快，重建solver的model可能又要耗一些時間，再加上collect value的overhead，因此我collect pattern之後再去做simulation並沒有加快我fraig的速度，因此我的作法就是直接從\_DFSList的begin()做到end()，結束就跳出fraig這個function。也因為時間上不太足夠，因此我就沒有再去嘗試其他的演算法，例如再fraig裡面去做strash之類的，就全部用SAT solver去解他，也因此效率非常差，跑sim13可能跑不出來。

結論 :

在寫了這個project之後，我才知道有些東西演算法不同效能可以差別到這麼大，也才漸漸在寫程式的時候會去這樣的演算法有沒有甚麼不必要的overhead，或者怎樣寫效率才會比較高。或許最後我的fraig跟sim的演算法效率都很差，但至少我在有限的時間內試過不同的方法，有些看起來比較快的方法，實際試起來卻沒有比較快，這些也都是要經過實驗才能試出來的。另外還有Hash的大小的影響，也是在我隨便亂試的過程中不小心發現的，所以其實可能程式各個部分都對演算法有影響，只是我們不一定有發現。有時在修改演算法的過程會因為寫了一大堆程式結果出來反而比較慢而覺得煩躁，但後來才漸漸發覺如果沒有這個嘗試的過程，也沒辦法得到最佳的程式。雖然這次時間不夠沒把效能的部分寫好，但我如果有空一定會好好想想要怎麼在增加他的效能的。