

DSNP FINAL REPORT

電機二 B05901082 楊晟甫

b05901082@ntu.edu.tw

0910784797

1. Data Structure:

CirGate :

以兩個 `size_t` 作為其 `fanin`，其值為 literal 而非 ID，奇數代表兩者之間有 invert 關係，所以要存取的話就用 `Gate_List[Gate_List[i].fanin >> 1]`。而以一個 `vector<size_t>` 作為 `fanout_list`，兩者之間的關係如同 `fanin`。`bool printed` 是在做 DFS_List traversal 的時候用來記錄是否走過。`simulate` 紀錄最後 64 個 `simulate` 出來的值。`IFEC` 紀錄最後 64 個 `simulate` bit 的 MSB 是否為 1。`FEC_ID` 紀錄所在的 FEC group 的 leader ID，如果沒有 group，則 default 為 -1。

CirMgr :

用一個 `CirGate*` 存所有的 gate，用一個 `vector<size_t>` 在 `cirread` 完所有 circuit 之後做 DFS 並 `push_back` 進此 `vecotr`。`MILOA` 各用一個 `int` 存，然後用三個 `unsigned *` 存 `PI`、`PO`、`AIG LIST`。除此之外，有另外寫了幾個 class 包含在 `CirMgr` 裡面，有：`GateNode`、`StrashKey`、`FECKey`，三者皆是應用在 `hashmap` 裡的 class，後段介紹 function 時會再詳細說明。`FECmap` 是自定義的 `hashmap` 型態，而 `FECgrp` 則是 C++ 的 `map` 型態，這兩者會在 `simulation/fraig` 中被使用，而後者使用的比重較重。

2. Function :

CirSweep :

用 probing 的方式，設定兩個變數 `i = j = 0`，`i` 走 DFS_List，`j` 走 Gate_List，當 `DFS_List[i] == Gate_List[j]` 時代表此 gate can be reached from PO，因此 `i++ j++`。當不相等時，代表此 gate 必須被 sweep 掉，並且 `j++`，直到兩者相等可以繼續前進。做完後 `updateDFS_List`。

CirOpt :

一樣從 DFS_List 的頭開始走，才能確保能一路 simplify 到 PO，然後另外寫了一個函數 `mergeAIG`，對於 `fanin` 跟 `fanout` 是否 invert 分成不同的 case 來 merge，`fanin`、`fanout` 的增加及刪減要 maintain 好。做完後 `updateDFS_List`。

CirStrash :

這個 function 是要 merge `fanin` 一樣的 gate，因此以兩個 `fanin` 相乘作為 key，放進 `hashmap` 裡面，為了防止 collision，在 `StrashKey` overload `==` operator，來確保兩個 gate 是真的有一樣的 `fanin`，而不是 `key%_numbucket` 有一樣的值所產生的誤判，一樣使用 `mergeAIG` 來合併，並在做完後 `updateDFS_List`。

CirSimulate :

For random：使用系統內建 `rand()` 函數，因為其最大為 16bit，因此要先轉成 `unsigned long long` 再往左 shift，不然會有 overflow 的問題，做四次以獲取一個 64bit 的 pattern，開始 simulate 後根據 `fanin` 是否 invert 來決定要做哪些 bitwise operation，做完後把 `simulate` 的 pattern `>> 63` 當作 IFEC，在第一次 simulation 的時候，將 gate 以 `<GateNode, FECKey>` 的形式放進 `FECmap` 裡面，其中 `FECKey` 是用 c++ 內建的 hash function 來產生，全部放進 hash 以後，為了減少 collision，

因此將資料轉移進一個 `map<size_t, vector<size_t>> FECgrp`，iterate 整個 `FECmap`，如果該 `bucket` 有超過兩個 `gate`，則以 `bucket` 的第一個元素作為 `key`，將該 `bucket` 的 `gate` `push_back` 進 `FECgrp` 裡面，然後第二、第三次模擬則以這個 `map` 作為基準，去檢查每個 `group` 裡面的 `gate` 是否跟該 `group` 的 `leader` 相同，不同的話就以該不同的 `gate` 開始向後 iterate 到該 `key` 的 `vector` 的最後，如果後面的 `gate` 的 `pattern` 跟其一樣，也就說明這兩個都不一樣，因此就新增一個 `key` 和其 `vector`，`erase` 掉舊的。在每次 `sim` 時以一個 `bool change` 紀錄是否有變動，如果沒有的話這次 `sim` 就算 `fail`，當 `fail` 超過 `I+1` 時，就終止 `simulate`。

For file：先讀檔，並把 `pattern` 用 `vector<string>` 存在 `CirMgr`，然後把 `pattern` 一個一個 0/1 `shift` 進 `PI` 的 `simulate`，然後開始模擬，其實作方式與 `random` 相同，如果有 `output option` 的話，則把 `pattern` 傳給 `*_simLog`。

CirFraig：

首先建立 `sat` 引擎，然後對每個非 `UNDEF` 的 `Gate` `assign` 他們一個 `var`，對於 `UNDEF` 則把 `CONST 0` 的 `var` `assign` 給他們，對於每個 `AIG gate` 添加 `AigCNF` 的 `clause`，如果 `input` 是 `invert` 就傳入 `true`，並傳入 `fanin` 的 `var`，然後把 `pattern` 清空以利接下來出現 `SAT` 的時候可以存取 `counter example`，接著開始從 `DFS_List` 的頭開始走，加入 `CONST 0` 的 `property` 以及對於每個 `gate`，與其在 `FECgrp` 中的 `leader gate` 做 `XOR`，如果此 `property` 為 `true` 代表兩者相異，透過 `getValue()` 存取此 `counter example` 並進入 `fsim()`，在這邊我是一次 `SAT` 就進一次，但感覺還有更好的辦法，但想不到也沒有時間去實作，進 `fsim()` 之後，基本上與 `simulation` 大同小異，就是對 `FECgrp` 做維護，刪減掉這個 `gate` 或是新開一個 `key`；若是 `UNSAT`，則代表兩個 `gate` `functionally equivalent`，則進行合併，合併完以後一樣要從 `FECgrp` 中刪除，最後 `updateDFS_List`。

3. Performance

CirSweep、Ciropt、CirStrash：

這三個指令我分別一次執行 `opt01~07.aag` `strash01~10.aag` 並且 `combine` 三種指令的 `dofile`，所得到的 `memory used` 跟 `runtime` 分別都是 0。差比較多的部分是 `Simulation` 還有 `Fraig` 的部分。

Simulation -Random：(sim13.aag 左為 ref)

```
fraig> cirfraig
Updating by SAT... Total #FEC Group = 0

fraig>
joey@joey-VirtualBox:~/Desktop/fraig$ ./fraig-linux
fraig> cirr tests.fraig/sim14.aag

fraig> cirsim -r
5376 patterns simulated.

fraig>
joey@joey-VirtualBox:~/Desktop/fraig$ ./fraig-linux
fraig> cirr tests.fraig/sim13.aag

fraig> cirsim -r
138816 patterns simulated.

fraig> usage
Period time used : 9.75 seconds
Total time used : 9.75 seconds
Total memory used: 17.48 M Bytes

fraig>

Note: original circuit is replaced...
Note: original circuit is replaced...
Note: original circuit is replaced...
Note: original circuit is replaced...
Note: original circuit is replaced...
Note: original circuit is replaced...
Note: original circuit is replaced...
joey@joey-VirtualBox:~/Desktop/fraig$ diff output.log outputr.log
374d373
< Sweeping: AIG(7) removed...
375a375
> Sweeping: AIG(7) removed...
joey@joey-VirtualBox:~/Desktop/fraig$ ./fraig
fraig> cirr tests.fraig/sim13.aag
c
fraig> cirsim -r
277056 patterns simulated.

fraig> usage
Period time used : 70.18 seconds
Total time used : 70.18 seconds
Total memory used: 26.53 M Bytes

fraig>
```

在 small circuit 中其實差不多，以 tests.fraig/ISCAS85/C432.aag 為例，ref cirsim -r 花了 0.02 sec 而我的則是 0.05 sec，我覺得我的速度可能主要差在兩個部分：一是要不斷 iterate 我的 FECgrp，其次是在每個 group 中的 vector 要刪掉已經不存在的 Gate，erase() 據我所知是非常 time consuming 的。

Simulation - File : (sim13.aag 左為 ref)

```
joey@joey-VirtualBox: ~/Desktop/fraig
fraig> cirr -r tests.fraig/ISCAS85/C432.aag
Note: original circuit is replaced...

fraig> cirsim -r
3200 patterns simulated.

fraig> usage
Period time used : 0.02 seconds
Total time used : 9.77 seconds
Total memory used: 17.48 M Bytes

joey@joey-VirtualBox:~/Desktop/fraig$ ./fraig-linux
fraig> cirr tests.fraig/sim13.aag

fraig> cirsim -f tests.fraig/pattern.13
22912 patterns simulated.

fraig> usage
Period time used : 2.35 seconds
Total time used : 2.35 seconds
Total memory used: 18.59 M Bytes

fraig>

joey@joey-VirtualBox: ~/Desktop/fraig
fraig> cirr -r tests.fraig/ISCAS85/C432.aag
Note: original circuit is replaced...

fraig> cirsim -r
2752 patterns simulated.

fraig> usage
Period time used : 0.05 seconds
Total time used : 70.23 seconds
Total memory used: 26.53 M Bytes

joey@joey-VirtualBox:~/Desktop/fraig$ ./fraig
fraig> cirr tests.fraig/sim13.aag
c
fraig> cirsim -f tests.fraig/pattern.13
22912 patterns simulated.

fraig> usage
Period time used : 16.4 seconds
Total time used : 16.4 seconds
Total memory used: 118.9 M Bytes

fraig>
```

```
[3793] 84546 !84549
[3794] 84565 84619
[3795] 84596 !84597
[3796] 84624 !84625
[3797] 84650 !84651
[3798] 84677 !84678
[3799] 84702 !84703
[3800] 84729 !84730
[3801] 84754 !84755
[3802] 84781 !84782
[3803] 84806 !84807
[3804] 84833 !84834
[3805] 84858 !84859
[3806] 84885 !84886
[3807] 84910 !84911
[3808] 84935 !84936
[3809] 84947 84949
[3810] 84956 84957
[3811] 84991 !84995 !84996
[3812] 85004 !85005
[3813] 85007 !85008 !85009
[3814] 85012 !85013

fraig>
```

```
[3793] 84546 !84549
[3794] 84565 84619
[3795] 84596 !84597
[3796] 84624 !84625
[3797] 84650 !84651
[3798] 84677 !84678
[3799] 84702 !84703
[3800] 84729 !84730
[3801] 84754 !84755
[3802] 84781 !84782
[3803] 84806 !84807
[3804] 84833 !84834
[3805] 84858 !84859
[3806] 84885 !84886
[3807] 84910 !84911
[3808] 84935 !84936
[3809] 84947 84949
[3810] 84956 84957
[3811] 84991 !84995 !84996
[3812] 85004 !85005
[3813] 85007 !85008 !85009
[3814] 85012 !85013

fraig>
```

由於與 sim - random 實作方式相同，因此時間上的差距是可以預見的，但記憶體暴增我想是因為我用 vector 把所有 input pattern 都存下來而不是一次讀 64 個 bit 然後直接 simulate(我猜測老師可能這樣做?)

Fraig : (sim13.aag 左為 ref)

```
Joey@Joey-VirtualBox: ~/Desktop/fraig
Fraig: 81435 merging 85017...
Fraig: 84078 merging 85053...
Updating by UNSAT... Total #FEC Group = 0
Updating by SAT... Total #FEC Group = 0
Strashing: 33654 merging 34412...
Strashing: 49004 merging 49835...
Strashing: 66957 merging 68012...

fraig> usage
Period time used : 111.4 seconds
Total time used : 111.4 seconds
Total memory used: 52.77 M Bytes

fraig> cirp -s

Circuit Statistics
=====
PI      3357
PO      3343
AIG     78878
-----
Total   85578

fraig>

Joey@Joey-VirtualBox: ~/Desktop/fraig
Fraig: 0 merging 84240...
Fraig: 0 merging 184242...
Fraig: 0 merging 84948...
Fraig: 84947 merging 84949...
Fraig: 81435 merging 85017...
Fraig: 70497 merging 185029...
Fraig: 84078 merging 85053...

fraig> cirp -s

Circuit Statistics
=====
PI      3357
PO      3343
AIG     78879
-----
Total   85579

fraig> usage
Period time used : 5536 seconds
Total time used : 5594 seconds
Total memory used: 143.3 M Bytes

fraig>
```

92 分鐘的 runtime...然後 gate 數量也不對，我的 fraig 在大測資應該是拿不到什麼分了，也許小測資有機會，我覺得時間的差距是在老師上課提到的 reuse counter example，但我沒有很明確的想法，所以我是拿到一次 pattern，也就是 SAT 一次就進 simulation 一次，由於機制跟前面 simulation 差不多，又要 iterate 又要 erase()，可以預見所花的時間應該蠻多的。

(sim14.aag 左為 ref)，小測資正確率較高。

```
Fraig: 0 merging 804...
Fraig: 0 merging 1806...
Fraig: 656 merging 1421...
Fraig: 0 merging 802...
Fraig: 0 merging 801...
Fraig: 0 merging 1803...
Fraig: 0 merging 1897...
Fraig: 0 merging 1899...
Fraig: 0 merging 1903...
Fraig: 0 merging 1911...
Fraig: 0 merging 1927...
Updating by UNSAT... Total #FEC Group = 0

fraig> cirp -s

Circuit Statistics
=====
PI      41
PO      1
AIG     611
-----
Total   653

fraig>

Fraig: 0 merging 805...
Fraig: 0 merging 804...
Fraig: 0 merging 1806...
Fraig: 656 merging 1421...
Fraig: 0 merging 802...
Fraig: 0 merging 801...
Fraig: 0 merging 1803...
Fraig: 0 merging 1897...
Fraig: 0 merging 1899...
Fraig: 0 merging 1903...
Fraig: 0 merging 1911...
Fraig: 0 merging 1927...

fraig> cirp -s

Circuit Statistics
=====
PI      41
PO      1
AIG     611
-----
Total   653

fraig>
```

4. Class Review

這門課的 loading 真的很重，不是開玩笑的，這學期學下來除了更加熟悉不同型態的資料操作之外，我覺得學到最多的是 class 的設計，特別是 Encapsulation，還有 debugger 的使用、linux 系統的操作還有 dofile 的使用，這些都是在上這堂課以前完全不會的，所以雖然很辛苦，但實在是收穫許多的一堂好課。