資料結構

與程式設計

FRAIG

(Functionally Reduced And-Inverter Graph)

REPORT

系級: 電機三

學號: B04901069

姓名: 林志皓

信箱: b04901069@ntu.edu.tw

目錄

==========

一、基本資料結構設計

我在 CirGate.h 中，有一個 class CirGate，儲存所有 gate 都會用到的 member function、data member，(i.e. gateID、lineNumber、simulation value…)，然後用三種 class 去做 inheritate 他，分別有自己才會用到的 function、data。

分類如下：
(1) Class PIgate:電路中的 PI、Constant 0。有 name、紀錄 output 的 vector
(2) Class AIGgate:電路中的 AIG gate，有紀錄 FanIn 的 array、FanOut 的 vector
(3) Class POgate:電路中的 PO，有 name、紀錄 FanIn 的 array

我記錄 fan in 是用 size = 2 的 array，紀錄 fan out 則是用 vector，而兩者都是儲存 CirGate pointer，另外有一個 size = 2 的 bool array 紀錄是否為 inverse fan in 而在 class CirMgr 中，我開一個 size = M+O(first line in aag file)的 vector，根據每個 gate 的 ID 存在相對應的位置(儲存 CirGate*)，使得在 read circuit 時可以在 constant time 取道相對應的 gate，加快速度。另外有兩個分別記錄 PI、PO 的 vector，方便 simulation、需要 DFS 時的使用。

效能分析:

相較於教授推薦的實行方式，將 inverse 的資訊寫在一個 class 中，我的方式在刪除 gate，連接 gate 的時候需要花心力去 maintain bool array，但這方式對我來說較為直觀所以就選擇用這方式；而教授把 output list 記錄在 CirMgr 裡，我的則是直接記錄在 gate 中，在後續的一些功能實行上，delete / merge gate 時需要比較多心力去 maintain，但是 performance 上和 reference code 差不多(read circuit 時) 像是 sim13.aag(有上萬個 gate)，memory 和 ref 相差無幾，run time 和 ref 比略為 3 倍左右(ref: 0.04s ←→ mine:0.12s)

二、Sweep、Simple Optimization、Structural Hash 實行方式

Sweep

先跑一次 DFS，mark 所有 reachable 的 gates，接著在 CirMgr 儲存所有 gates 的 array 中把其他所有 unmarked 的 gate 刪除。

Simple Optimization

根據 DFS 順序，由 PI 往 PO 依序 ciropt，分別根據不同的 case(fan in =0/1,same fan in…etc)進行 optimization，即可得到正確的結果。(若是由上而下，可能會因為 fan in 的 gate 尚未被處理過，而將該消除的 gate 未被消除。)

Structural Hash

我用 HashMap 這個資料結構去實行，把每個 AIGgate 的兩個 fan in 的 gateID 當作 Hashkey，丟進 HashMap 中之後就會把相同 key 值的 hashData(CirGate*)merge 在一起，順序依樣是根據 DFS 的順序，由 PI 往 PO 的方向做 strash。

效能分析:

以上提到的三項功能，就正確性和速度都和 reference code 相差無幾，就檔案中所有提供的測資而言。

三、Simulation 實行方式與限制

在每一個 gate 中有一個 data member 叫做"_value"，是一個 size_t，可以記錄 64 位元的 simulation result，可實行 parallel simulation。
而我另外創造了兩個 class(在 CirDef.h 中)_，分別是 :

(1) FECgroup :

是一個紀錄 FEC groups 的 vector<int>，儲存的數字是每個同 group 的 gateID，至於為何要用 int 來存呢?我讓是否 inverse 這項資訊用「正負」來表示；舉例來說，如果一個 FEC group 中存: (5 ,-9)，則表示編號 5 的 gate 和編號 9 的 gate 的 inverse 是同一個 FEC group，到時候如果要得到 group 中 gate 的 value 時，只要把數字加上絕對值即是 gateID，即可在 vector 中取得 CirGate*。

(2) FECtable :

是一個儲存所有 FEC groups 的容器，紀錄有多少數量，並在必要時更新，我在 class CirMgr 中有一個這樣的 class，記錄整個電路中的 FEC groups，在需要時重新排序過並整理。

我在每個 gate 中儲存了一個 int _fec，記錄他屬於 FECtable 的哪一個 FECgroup，正負號表示在該 FECgroup 中，此 gate 是否違反向。如此一來便可在 report gate information 時便可直接在 CirMgr 中找到相對應的值並印出，不需要在每個 gate 中儲存一個 FECgroup，省下不少記憶體。

限制與效能分析 ：

整個 simulation 的時間並不會太長，但是在印出 FEC groups 時由於要先排序再印出(排序是自己寫的 insertion Sort，因此需要點時間，但是檔案最大的 sim13.aag 也不需要 5s。由於我 random simulation 的時候都固定次數(9600 次)，因此在檔案較小時可以分出較多 groups，但檔案較大時就相反過來了

(在印出時也需要比較多時間排序)。

四、Fraig 實行方式與限制

我開一個跟 FECgroup 數一樣的 array : table(存 CirGate*)，根據 DFS 的順序，如果遇到的 gate 所屬的 FECgroup 數在 table 上是空的，就放進去；如果已有儲存其他 gate*，則呼叫 SAT 工具去證明是否為相等

(1) 若相等，則將兩者 merge 起來(table 中的 CirGate*不會變)

(2) 若不相等，則將 counter example 儲存起來，當蒐集到 64 個的時候重做一次 simulation，將 FECgroups 分的更散一些；再回到一開始周而復始，直到再也沒有任何 FECgroups 為止。

而為了加速 SAT 工具的速度，當需要再 simulation 一次的時候，就重新建造一次 SatSolver，減少需要證明的複雜度，多了這個機制，我的程式快了不少。

限制與效能分析：

依照 DFS 的順序可以正確且較為快速地證完所有 FECgroups，而與 ref 相比，我的程式略慢一些，在最大的 sim13.aag(有將近 10 萬個 gates)，我的程式跑的時間大概是 ref 的 6 倍左右，其他的測資幾乎都能乎不長的時間，正確的得到簡化的電路。

五、修課心得

寫到這部分的時候，代表為期一學期的 DSnP 也要進入尾聲了。

當初帶著不怕死的心情來修，老實說 HW0 對我來說並不輕鬆，但覺得想嘗試看看，所以還是決定修這門「9 學分」的課。在寫 HW3 的時候壓力很大，因為幾乎看不懂架構，很懷疑自己的作業是不是會開天窗，到後來就算不是很順利，但還是跌跌撞撞的教出一次次的作業，到最後的 FRAIG，雖不完美，但跟學期初的我比起來，真的已經進步很多很多了。

非常感謝這門課的眾多助教們，這門課將近 200 人修，每次改作業都是一個非常浩大的工程，有人忘記交，有人 make fail(其中有幾次也有我 XD)，助教還要發文通知一個個處理，更不用說平常還要被大家纏著問問題了，之後批改作業有多辛苦我也無法想像，再次感謝助教們驚人的耐心。

當然最感謝的還是教授了，一整學期下來，不論是 C++的設計技巧上或是資料結構的介紹與實作，真的都讓我們這一大堆學生能更加理解，在每週花大把時間寫這些精心設計的作業後，相信這些都至少不會是走馬看花，可能會深刻的停留在腦中好一陣子了。而教授開這門課將近 10 年(還是超過?)，每年都教一樣的東西對自己來說也許不是一件太有趣的事情，但是對於第一次修的人收穫都很大，每次作業教授都會在 FB 上耐心回答大家的問題(每個人都會回)，每學期 7 次作業，1 個 final，如此持續了好幾年，這到底需要多少的耐心?

有耳聞教授近幾年在考慮是否要繼續開這門課，還好能趕在停開之前修到，分量如傳聞中的重，但我也因此學到很多，肯定有很多人也是，再次感謝教授、助教們、會跟我討論的朋友，還有在網路上發問讓我也能學到一些的同學，這學期的 DSnP 感謝你們照顧了!