資料結構與程式設計
期末專題報告

FRAIG

姓名：王傑生
學號：B03901052
E-MAIL：b03901052@ntu.edu.tw
聯絡電話：0925888582
日期：2016年1月21日

一、資料結構設計(Design of Data Structures)

1. 概觀

電路的部分整體包含CirGate以及CirMgr兩個部分，CirMgr為一個控制整體電路操作的class，CirGate則是電路中gate的class，由CirGate定義一個各種gate的base class，有CirAigGate、CirPiGate、CirPoGate、CirConstGate等四個derived class分別定義ANDGate、INPUT、OUTPUT以及CONSTANT，另有一個CirGateIn class控制各個gate的input。

2. Circuit Gate

在各個gate的base class CirGate之中定義了下列的data member(皆為protected)：

- size_t _flag：做電路的traversal時所做的標記，為mutable。
- size_t _globalFlag：traversal時用的共同flag，為static。
- unsigned _id：aag中定義的各個gate的ID。
- unsigned _lineNo, string _name：aag中的的行數以及定義的gate name。
- vector<CirGate*> _outList：記錄gate所output到的gate，po中size為0。
- vector<CirGateIn> _inList：記錄gate的input，aig中size為2，po為1，pi及const為0。
- unsigned _dfsOrder：gate在一開始的DFS list中的順序(topological sort)，主要供fraig使用。
- unsigned FECId：在fraig中要被merge的 group中所屬的group id。
- size_t _simOut：gate在simulation中的輸出值，在PI中則代表circuit的輸入。
- Var _satVar：SAT engine中的gate id。

在各derived class當中，沒有新增其他的data member，主要為一些virtual function的繼承，包括getType、printGate、simulate等。

在CirGateIn當中定義了CirGate* _gate儲存input gate的記憶體位置，或在input為undefined的情況下設為0，bool inv 記錄input是否要invert，unsigned _undefID 儲存若input為undefined，在aag file中宣告的ID。class中也設置各種access以及方便modify的function使得input的部分可以完全透過這個class來掌控。

3. Circuit Manager

CirMgr為管理整個電路的class，故儲存了關於電路的各種資訊，以及所有gate的pointer。

- 包括IdList(即vector<unsigned>) _piList、_poList、_floatingList、_unUsedList、_netList，分表儲存所有PI、PO、floating gate、unused gate、DFS order的ID，並由一個GateList(即vector<CirGate*>) _ids 以ID做為index，所對應gate的pointer做為data的陣列，做為使用id查詢gate的table所使用，也方便在destruct時一同delete各個gate的記憶體。
- size_t noPI、noPO、noLA、noAIG、maxGateNo儲存各種元件的數量以及id最大值。

- bool FECdone 作為flag判對是否已simulate。
- unsigned maxSimFail、maxSimNo 作為random simulation的控制條件使用。
- list< IdList* > _FEClist：儲存FEC groups，因在collect FEC groups時會需要大量使用insert及delete操作，故使用list，list的node為IdList的pointer，而非直接使用IdList物件避免產生copy整個vector的情況影響效率。
- 其餘各演算法中使用的Data Structure，於二、演算法設計中說明。

二、演算法設計(Design of Algorithms)

1. Sweep Gate

建立一個unsigned的HashSet，將_netList(DFS)中的AIG gate ID放入HashSet

掃過_ids (總GateList)，對於所有存在的AIG gate：

- 檢查id是否在hashset中
- 若不在(即要被sweep的gate) 則對其存在的input，將_outList中的此gate刪除(delOut(g))
- 最後刪除此gate (delGate(g))

其中delOut(定義在CirGate中)：掃過_outList搜尋g，將_outList中最後一個element取代g，pop_back (因順序不重要)，時間複雜度linear to _outList size。

其中delGate：首先搜尋floatingList是否包含g，若有則刪除，接著釋放g的記憶體，將_ids(ID對gate的table)中g的部分設為0，並將noAIG(AIG的數量)減一。

結束後float 及 unused list可能會變動，但因不會使用到這兩個list，故不在此時更新list(print floating時會更新)。

2. Trivial Optimization

按照_netList(DFS)順序逐一檢驗optimization type(const0 fanin, const1 fanin, inverted fanin, same fanin)，並進行對應的merge(function: optDelGate)，const0 fanin及inverted fanin以const0 gate merge，const1 fanin 以另一個input gate merge，same fanin以此fanin gate merge。

其中optDelGate：

- 傳入參數為要被merge的gate g以及merge g的CirGateIn&(handle input的class) in。
- 將g所output到的gate中原為g的fanin設為in並考慮invert的問題 (setGateInv(in))。
- 將in所對應的gate的output增加g的output。
- 將原先到g的output刪除(delOut)並刪除g(delGate)

結束後若有更動到_netList，則重新DFS更新_netList，_unUsedList可能會有變化，但不在此時更新，需要用到前再進行更新(print floating)，因此function會在做FRAIG時被使用到，多更新unUsedList會造成不必要的效能影響。

3. Structural Hash(Strash)

建立一個<GateHashKey, CirGate*>的HashMap，其中GateHashKey包含兩個unsigned的HashKey，兩個unsigned a, b分別對應AIG gate的兩個input gate，Hash function為a^3 + b^3。

按照_netList順序，對於所有AIG gate：

- 以input gate ID * 2 + inverted創造HashKey。
- check 此HashKey是否已在HashMap中，若是則以已在HashMap中的key所對應的gate來merge此gate，若否則將此HashKey放入HashMap中。

結束後若有更動到_netList，則重新DFS更新_netList，_floatingList可能會有變化，但同Sweep，不在此做更新，因此function會再FRAIG中被使用，減少不必要的更新以增加效率。

4. Simulation & FEC groups collection

Simulation：

以亂數或file讀入的pattern創造出一組大小為size_t的pattern並載入PI gate當中，按照_netList的順序逐一做每個gate的simulation更新_simOut。

FEC groups collection：分為第一次simulation跟其餘simulation兩種狀況

第一次simulation後：

首先建立一個<SimHashKey, IdList* >的HashMap，其中SimHashKey為為包含一個size_t的HashKey，size_t表示gate的simulation結果，Hash function直接return此size_t。

原先所有的gate都以gate ID * 2的形式(表示non-inverted)存在FEC list的第一個group中，對於此group中的所有gate ID：

- 以此gate的_simOut創造對應的SimHashKey。
- 分別以此key及inverted後的key query HashMap。
- 若已存在，則push_back入query到的IdList，否則若inverted已存在，則將ID+1(表示inverted)後push_back入query到的IdList，若都不存在則創建一個新的IdList (pointer)，push_back後將這組key及IdList放入HashMap中。

最後將HashMap中所有的size大於等於2的IdList插入_FECList中，並刪除原本的group。

其餘simulation後：

大致上概念與前者相同，可視為是對於_FECList中的各個group進行前述的步驟，但由於此時各個gate是否inverted已經確定，故每個gate只需針對現在是否是inverted，創建對應的

key，query一次即可。由於_FECList為linked list，故delete及insert皆只需要constant time。
另外對於size為2的group，可能的結果只有留下或是刪除，故不需使用前述的方法，直接比較兩個gate的simulation結果，決定是否留下此group即可，如此可增加group collection的效率。

整個simulate指令大致上就是上述simulation -> group collection的循環。在random simulation時，會根據電路的大小設置maxFail(即經過cycle後_FECList沒有變化的次數上限)以及maxCycle(即最大的cycle次數)，此兩參數的決定方法見三、效能實測與最佳化。在file simulation時，則simulation直到讀入的pattern的結束為止，若有最後一個pattern不滿size_t大小，則補0。

5. FRAIG
FRAIG大致上的流程為 create sat model -> prove -> merge -> re-simulate 的循環直到_FECList變為empty為止。
在每個循環內：
Prove：
根據每個FECGroup中第一個gate的DFS order(若第一個為const0 則改用第二個)建立一個priority queue(min-heap)，每次從queue中取得一個group出來作為接下來要驗證的group：

- 若group size為2，則直接由sat驗證這個pair，若確實等價則將此group放入一個待merge的list當中(vector< IdList* >)，若不等價，則取得造成不等價的pattern，放入PI gates當中等待下一次，並釋放此group的記憶體。不論驗證的結果，都將此group由_FECList及queue中移除。
- 若group size大於2，則將每個group中的gate與第一個gate組成一個pair進行驗證，若等價則將此gate保留在group中，若不等價則取得pattern放入PI，並從group中移除並放入另一個暫存的group。最後若剩下的group size大於等於2則放入待merge的list中，否則刪除，若暫存的group size大於等於2，則放入_FEClist及priority queue中，否則刪除。

prove直到滿足特定條件為止，如SAT次數超過某個數等，見三、效能實測與最佳化。

Merge：
若待merge的list不是empty，則進行merging。使用類似Strash的方法來merge，確保merging順序正確不會造成cycle，與Strash的差別在於，此時改用一個unsigned作為HashKey。
首先將_netList內各Gate的FECId設為0，接著將各待merge的group中gate的FECId設為該group在list(vector)中的index+1(避免與0重複)，若gate為inverted則將FECId invert。

剩下與以與Strash相同的方法進行merging，惟分別用FECId及~FECId創建HashKey進行query決定merging時是否要invert。由於先前prove的限制條件可保證待merge list的size小於2^16，故此方法可以正確運作。Merging後重新DFS更新_netList。

Re-simulate：
先利用與sweep類似的方法，將不在_netList中的gate從_FECList當中移除，接著以先前在prove步驟中放入PI的pattern進行一次simulation與FEC groups collection。

加速FRAIG的方法：

i. 在進行一個size較大的group驗證時，如產生數組不等價(SAT)時利用取得的結果跑一次simulation，在做剩下的驗證時，可先比較simulation結果，若不同則不必prove此組pair，由於與sat engine prove相比，進行一次simulation花費明顯較少，可提升不少效率。
ii. 當從SAT engine取得size_t大小(64)組pattern時，可直接中止prove，進行merging與re-simulate，當在某組當中prove到一半，將已驗證的部分組成一個group放入待merge的list，剩下的尚未驗證的部分與先前驗證結果不等價的以及原本group的第一個gate組成另一個group放回_FEClist當中，由於這些pattern在re-simulate時可以將原先驗證不等價的部分分開，因此不會有中斷後，下個cycle會要prove同一個pair的情形發生。
iii. merging完後，進行trivial optimization 以及 Strash，由於可被trivial optimized以及strash的gate皆可被sat engine驗證，先進行這些步驟不但可以減少sat engine需要驗證的pair數量，也可在下次simulate及prove之前讓電路變的更簡單，增加效率。

三、效能實測與最佳化(Performance and Optimization)

1. Sweep, Trivial Opt & Strash

此三個為較單純的指令，皆為linear time，沒有需要調整的參數，加上相比simulation及FRAIG花的時間通常十分少，因此對整體效能影響小，經測試這三個指令效能都大約與reference program相等，故不在多加敘述。

2. Simulation

Random simulation可調整的參數有max cycle數以及max fail數，以下先測試不同的PI、AIG數量在不同的max cycle及max fail下花費的時間及FEC group的數量及_FECList中總gate數。

實驗步驟：read -> random simulation -> check result

實驗結果：(maxFail, time (s), cycle count, # of groups, # of gates)

sim06.aag(# of PI = 4, # of AIG = 4270)	sim07.aag(# of PI = 4, # of AIG = 9437)
(1, 0, 2, 250, 1895) (5, 0, 5, 250, 1895)	(1, 0, 2, 879, 8181) (5, 0.01, 5, 879, 8181)

sim09.aag(# of PI = 178, # of AIG = 3286)
(1, 0, 10, 330, 2390) (5, 0.01, 15, 332, 2387) (10, 0.01, 23, 334, 2386)
(20, 0.01, 37, 338, 2386) (40, 0.02, 58, 339, 2386)

sim10.aag(# of PI = 36, # of AIG = 176) (CONST1)
(1, 0, 8, 245, 681) (5, 0, 13, 249, 681) (10, 0, 18, 249, 681)

sim12.aag(# of PI = 277, # of AIG = 9364) (CONST0)
(1, 0.01, 19, 2584, 6534) (5, 0.02, 25, 2597, 6529) (10, 0.03, 35, 2602, 6505)
(20, 0.04, 51, 2601, 6462) (40, 0.07, 81, 2605, 6425) (100, 0.11, 145, 2602, 6407)

sim13.aag(# of PI = 3357, # of AIG = 81710)
(1, 0.67, 91, 4283, 17385) (5, 0.95, 128, 4096, 16487) (10, 1.15, 166, 4028, 15889)
(20, 1.46, 213, 3956 15337) (40, 1.89, 294, 3852, 14360) (100, 2.73, 437, 3765, 13423)
(200, 3.67, 613, 3619, 12551) (400, 5.38, 926, 3458, 11521)
(800, 8.28, 1426, 3350, 10797) (1600, 12.4, 2313, 3184, 10170)

sim15.aag(# of PI = 41, # of AIG = 886)
(1, 0, 16, 58, 129) (5, 0, 23, 42, 87) (10, 0, 30, 37, 75) (20, 0, 41, 33, 66) (40, 0, 61, 31, 62)

miter13.aag(# of PI = 3357, # of AIG = 169001) (miter, CONST0)
(100, 18.3, 502, 70760, 167413) (200, 25.17, 690, 71462, 167411)

由上述實驗數據可看出對於PI少的電路而言，需要的simulation次數少，因所有可能input的組合少。而對於小電路而言，即便sim的次數高，總花費時間依然很少，故主要針對較大的電路進行最佳化。對於整個電路而言，所有不同種的input共有2^PI種組合，因此根據PI數量來設定maxFail的值，並用AIG的數量設定一個總上限。
根據實驗的數據，最後決定的maxCycle設為1 + AIG/50，maxFail設為3 + PI/8。

3. FRAIG
FRAIG可調整的部分有：
- maxCycle：即不論驗證結果，驗證多少次後強制進入merge。
- maxSATCount：由sat engine驗證出多少不等價的次數，即取得多少組可用的pattern後強制進入merge
- maxGroupSAT：在每個group當中每取得多少sat engine的pattern就重新simulation，利用simulation結果加速跳過部分驗證(即二的第五點中加速方法的第一項)

一些固定的參數：當group size大於等於5時才會進行simulation加速驗證。首次得到3組pattern後即會進行simulation加速驗證，後續的頻率才由maxGroupSAT決定。

實驗步驟：read -> random simulation -> FRAIG(會做完所有FEC group)
實驗結果：(maxCycle, maxSATCount, maxGroupSAT, simulation time, FRAIG time) ref參考時間
sim07.aag(# of PI = 4, # of AIG = 9437) (ref: sim = 0.03, FRAIG = 10.94 s)
(800, 31, 17, 0, 1.97) (800, 31, 11, 0, 2.02) (800, 63, 22, 0, 1.99) (100, 63, 22, 0, 2.19)

sim12.aag(# of PI = 277, # of AIG = 9364) (CONST0) (ref: sim = 0.26, FRAIG = 2.72 s)
(800, 31, 17, 0.06, 2.36) (800, 31, 11, 0.06, 2.46) (800, 63, 22, 0.06, 2.25)
(100, 63, 22, 0.06, 2.69)

sim13.aag(# of PI = 3357, # of AIG = 81710) (ref: sim = 3.87, FRAIG = 103.5 s)
(800, 31, 17, 5.53, 80.88) (800, 31, 11, 5.53, 75.63) (800, 63, 22, 5.53, 62.35)
(100, 63, 22, 5.53, 68.05)

miter07.aag(# of PI = 4, # of AIG = 11182) (ref: sim = 0.03, FRAIG = 21.15 s)
(800, 31, 17, 0.01, 2.69) (800, 31, 11, 0.01, 2.71) (800, 63, 22, 0.01, 2.69)
(100, 63, 22, 0.01, 2.66)

miter13.aag(# of PI = 3357, # of AIG = 169001) (miter, CONST0)
(100, 63, 22, 29.9, 282.6) (200, 63, 22, 30.0, 262.1) (400, 63, 22, 29.8, 231.5)
(800, 63, 22, 29.5, 235.2)

由上述的數據可以看出對於較大的測資而言，因進行sat驗證需時較長，若maxGroupSAT較小，即利用simulation加速驗證的更新頻率越高，對效率會有正面幫助。而較高的maxSATCount，即collect較多的pattern後再進行re-simulate效能會較好，但由於每個PI最多只能儲存64(size_t)個pattern，因此有其上限。maxCycle對程式效率也有一定程度的影響，數值直接由實驗結果來決定。
最後決定maxCycle = 400, maxSATCount = 63, maxGroupSAT = 22。