資料結構與程式設計

Functionally Reduced And-Inverter Graph (FRAIG)

B03901026 許凱傑

B03901026@ntu.edu.tw

1. Class Structure

ClassGate 被 PIGate、POGate、AIGGate、UNDEFGate、CONSTGate 所繼承，

其中_fanin、_fanout 為獨有的 data member，CirGate 的 data member 如下圖

```
protected:
        bool                separate;     //separate from FEC group or not
        unsigned                          line;        //which line define this gate
        unsigned                          dfsId;       //location in _DfsList
        unsigned                          gateId;      //variable ID i.e.( >>1 = /2 )
        unsigned                          FecNum;      //location in _FecList
        unsigned                          value;       //simulation value
        GateType                          gateType;
        static unsigned         _globalRef;
        mutable unsigned        _ref;                       //can be altered in const functions

};
```

ClassMgr 的 data member 如右圖，另外 VECTOR<GateList>_DfsList 與 vector< vector< GateList> >_FecList 為 global variable 宣告在外面

```
static GateList     _GateList;
//static GateList _DfsList;

private:
    IdList          PiList;     //storing PIGate's variable ID
    IdList          PoList;     //storing POGate's variable ID
    mutable size_t  _AIGNum;    //the number of AIGgates
    ofstream*       _simLog;
```

2. Algorithm

首先先介紹兩個常被呼叫的函式：

(1) buildDfsList (bool rebuild)

這個函式是用來建立_DfsList，因為在我們做優化的過程中，會有很多 gate merge 的過程(見下面 reconnect 函式)，為了確保_DfsList 一直記錄從 PO 做 DFS 的路徑，我會在每次做完有可能有 merge 的函式後就呼叫一次。| rebuild 是用來決定是不是要把原本建立的_DfsList 覆蓋掉，大多數的情況都是需要的，另外我選擇在 readcircuit 中就先做好剛輸入的時候的_DfsList，而因為現在有將_DfsList 存下來，所以 printNetList 函式也有跟著改動，不須每次都做 DFS。

(2) CirGate::reconnect(CirGate* replace, bool inv_or_not)

呼叫的這個 gate 會被 replace 取代，inv_or_not 則是代表這個 replace 需不需要經過 INV 再取代，因為 reconnect 時我只確保 input 的正確，再藉由 buildDfsList 來把 fanout 連起來，所以如同上面所述，做完優化後必須馬上呼叫 buildDfsList，但是因為 optimize()中 ConstGate 的 output 必須馬上更新才能 sweep，所以我有特別處理這個情況。

接著是這次作業中主要的幾個函式

(3) sweep()

這是裡面最簡單的函式，只需要從 PO 做 DFS 並將_GateList 走一次，確定誰沒有走到並把它刪掉、更新_GateList 就好

(4) optimize()

將_DfsList 走過一次並處理總共 4 種情況，我選擇先處理比較簡單的兩種，也就是有兩個相同 input 的 gate(不管 input 的 phase) 就呼叫 reconnect，如果 phase 相同就用 input 取代，如果不同就用 CONSTGate 取代。之後再解決 CONSTGate 的 output，一樣是被自己的另一個 input 或是被 CONST0 取代。

(5) strash()

首先我先完成了 hashMap，裡面有一個稍微不太一樣的函式，我有設計可以把原本儲存中有一樣的 key 的 hashnode 讀出來。

```
// return false is k is already in the hash ==> will not insert and info will get stored data
bool insert(const HashKey& k, const HashData& d, HashData& info) {
  size_t b = bucketNum(k);
  for (size_t i = 0; i < _buckets[b].size(); i++)
    if (_buckets[b][i].first == k) {
       info = _buckets[b][i].second;
       return false;
    }
  _buckets[b].push_back(HashNode(k, d));
  return true;
}
```

我的 StrashKey 很簡單就是 AIGGate 的兩個 input 的 literalID，如果是 UNDEFGate 就是 UINT_MAX，並且讓 fanin1 是較小的那個，之後就是將 _DfsList 走過一次，並用 insert 的方式放進 strashMap 裡面，由於是 DFS 的 list，所以較靠 input 端的 gate 會先被放進去，就不用去可能這個 gate 的 input 還沒放進去過的問題。而 operator()的產生方式我是將 fanin2 設為亂數種子，並把 fanin1 左移 9 位再做 bitwise XOR，用這個方式做出來的速度還不錯，我就沒有特別再去改。

(6) Simulation

首先先介紹 FECKey 就是存這個 FEC group 的共有 value(隨便存一種 phase 沒關係，因為 operator==會處理掉 phase 問題)

另外由於在分離 FEC group 所用到的函式應該是一樣的，所以我就把它拉出來成為 bool CirMgr::separateFEC(unsigned num)，如果新的_FecList 的 size 沒有變少就會 return false，在這個函式裡面就會把所有_DfsList 上的 gate 的 value 更新一遍並建立新的 FECList，並在最後把它跟舊的 swap。

在 FileSim 跟 RandomSim 如果 list 是空的就把_DfsList 上的 UNDEF、AIG 跟 CONST 放進去，然後再呼叫 separateFEC，最後 sort _FecList 並更新所有 gate 的 FECNum，如果需要 output log file 再輸出。randSim 中 maxFail = log(_DfsList.size())/log(2.1)

(7) Fraig()

使用 SATengine 來確保有沒有等價，如果等價就呼叫 reconnect 來 merge，如果不等價就記下 input special pattern 等一下呼叫 specialSim，specialSim

跟 fileSim 做法類似，只是多傳入一個 vector<vector<unsigned>>這個參數。

其中有設下兩個防止花費太多時間的參數，一個是 fail 一個是 Maxfail，fail 指的是同一個 gate 連續兩次 SAT 那就先不比對它，測試得知 fail=1 分離效果不好，fail>2 就又花太長時間。因為 fail 在比對成功後就會歸零，所以設計 maxfail 來計算總共失敗次數，並設計它必須小於

limit=sqrt(_DfsList.size())*30

3. Results and analysis

Sweep、opt、strash 速度都跟 ref 差不多，記憶體也只略多一點

Simulation 速度的部分較慢但也還可接受，記憶體大約是老師兩倍之內。

Fraig 則有比較大的問題，再處理較大的測資時如 sim13.aag 因為較難分開(sim06)或驗證所以時間較久，因為可能我的方法比較直觀就是全部跑一次，所以花費時間太久，必須設下這些限制才能改善。