資料結構與程式設計

Final Project

B00901141 電機二 黃昱嘉

信箱：b00901141@ntu.edu.tw

一、功能描述

此程式功能為：從一個電路描述檔(.aag)建立電路，並實做了清除 redundant gates、模擬輸出、找出 functionally equivalent candidates(FEC)、以及利用 SAT 程式確認並 merge equivalent gates。

二、資料結構設計

1. class CirMgr

CirMgr 為主要的執行者，實作了大部分功能。

CirMgr 裡存放了資料如下：

與建立電路相關：

```
size_t              M, I, L, O, A; // 存放參數
vector<CirGateV*>   input;         // 照 aag 檔中順序
                                   // 存放電路中所有 PI
vector<CirGateV*>   output;        // 存放電路中所有 PO
vector<CirGateV*>   aig;           // 存放所有 Aig gate
vector<CirGateV*>   undef;         // 存放電路中所有
                                   // undefined gate
CirGateV**          _idList;       // 照 id 順序存放 gate
vector<CirGateV*>   totalList;     // 依照 depth first
                                   // search(dfs)順序，
                                   // 存放有連至 PO 的 gate
```

與 strash 相關：

```
Hash<HashKey, CirGateV>* _hash;    // 存放 strash 的
                                   // hash table
```

與 simulation 相關：

```
ofstream*      _simLog;            // 輸出檔案
unsigned int*  simulationValue;    // 存放 simulation 結果
vector<vector<CirGateV>*>* _fecGroup;
                                   // 存放所有 FEC groups
size_t         _simNum;            // 存放 simulation 總次數
```

其他：

```
bool        *flag;             // 用以標示是否已執行過某動作
string      _fileName;         // 存放檔案名稱
bool        needToUpdate;      // 用以標示是否需重作 dfs
```

2. class CirGate

CirGate 即為一個 gate。一個 CirGate 物件便代表了一個電路上的 gate。

CirGate 裡存放了資料如下：

與 fraig 相關：

```
private:
 unsigned        _gid;
 Var             _var;
```

因為需要繼承，故使用 protected：

```
protected:
 size_t          _id;       // gate 的 id
 size_t          _lineNum;  // gate 在 aag 檔中的行數
 string          _name;     // gate 的名稱（若有指定）
 bool             _flag;      // 用以標示是否已做過某動作
 vector<CirGateV*> _faninList;    // 存放所有 fanin
 vector<CirGateV*> _fanoutList;   // 存放所有 fanout
```

CirGate 下有 CirPiGate, CirPoGate, CirAigGate 三個 derived class，僅有 virtual function 不同，並無增加 data member。

3. class CirGateV

CirGateV 主要是用於包裝 CirGate* 指標，並利用一些 bitwise 的技巧，紀錄是否有 invert 與是否為 undefined gate。

CirGateV 裡存放了資料如下：

```
private:
  size_t          _gateV;    // gate 的記憶體位置
```

_gateV 除了記錄了記憶體位置，同時也記錄了是否 inverted 和 undefined。由於一個 pointer 最後兩個 bit 必為 0，記錄方法即為覆寫 pointer 的最後兩個 bit：最後一個 bit 記錄是否為 inverted，倒數第二個 bit 記錄是否為 undefined。

由於 _gateV 是 private，因此 implement 了 getter：

```
CirGate* gate() const {
 size_t i = (~0x0)-(0x3);
 return (CirGate*)(_gateV & i);
}
```

(~0x0)-(0x3) 即為 11111…100，除最後兩位是 0 之外，其餘皆為 1。_gateV & I 的目的是要清除用來記錄 inverted 與 undefined 的最後兩個 bit。

三、演算法設計

1. 建立電路

首先讀進檔案，並分別以不同 gate 討論：

a. 若是 PI，建立一個 CirPiGate，放進 input。
b. 若是 PO，建立一個 undefined CirAigGate，此為 PO 連接的 gate。將這個 CirAigGate 放進 undef。另外建立一個 CirPoGate，其 fanin 為剛建立好的 CirAigGate，並將它放進 output。
c. 若是 Aig，首先從 _idList 找尋是否有先前建好的 undefined gate。若有，則將它移出 undef；若無則建立一個新的 CirAigGate，放進 aig。Fanin 也先找尋是否有已建好的 gate，若無則建立一個新的，並放進 undef。
d. 所有新建的 gate 都會被放進 _idList，以方便尋找與連接。

2. Simulation

分為兩種：從檔案讀取或亂數模擬：

(1) 從檔案讀取：

a. 建立一個 unsigned int array，大小為 PI 的個數。此為 input pattern sequence。
b. 確認沒有格式錯誤後，一行一行讀入。若讀入 1 則在對應的 array 值上加上 2^(行數-1)，如此一來， unsigned int 的一個 bit 便代表一個 PI 的 input pattern。由於 unsigned int 僅有 32 bit，至多讀 32 行。
c. 將建立好的 pattern 送入 simulation。利用 (sequence/2^(位數))%2 可以取出單一個 bit 的 0/1 值。記錄每個 gate 的輸出值，存入 CirMgr 的 data member sequenceValue。
d. 重複 a~c，直到將檔案讀完。

(2) 亂數模擬

a. 基本原理同 (1)，除了將讀檔案建立 PI 改為用亂數建立 unsigned int。

b. 停止條件是：當連續幾次輸入都沒有讓 FEC Group 個數減少時便停止。這個 threshold 的定義是 $\log_{10}$(PI 數+1)*10，因此停止條件將與 PI 個數有正相關。當 PI 少時，較容易將所有 simulation pattern 都測過，因此不需要做額外的測試。當 PI 多時，則 threshold 較大、simulation 較難停止，因此可以測較多 pattern，以減少 FEC groups 數。

3. Fraig

將 FEC groups 中任兩個 gate XOR 後，以 SAT 測試是否可能為 1。若為 1 則表示存在一組 pattern 使此兩個 gate non-equivalent；若為 0 則表示此 pattern 不存在，因此將他們 merge。

實作上需對每個 FEC group 兩兩檢查，因此時間複雜度是 C^n_2，即 n^2。

四、效能實驗設計

利用 sim06.aag 來測試以下功能：

```
cirr sim06.aag
cirp -n
cirSw
cirp
cirSim –r
cirp -fec
cirFraig
cirp
```

並與 reference program 比較，找出其中差異處進行討論。

五、實驗結果與討論

(1) 我的 fraig 實驗結果：

```
cirr sim06.aag
Period time used : 0.16 seconds
Total memory used: 1.168 M Bytes
```

```
cirSw
cirp

Circuit Statistics
==================
  PI          4
  PO       2176
  AIG      2176
------------------
  Total    4356

Period time used : 0.13 seconds
Total memory used: 1.168 M Bytes

cirSim -r
288 patterns simulated.
cirp –fec
(共 250 個 FEC groups)
Period time used : 2.06 seconds
Total memory used: 3.469 M Bytes

cirFraig
Period time used : 2.35 seconds
Total memory used: 4.293 M Bytes

cirp

Circuit Statistics
==================
  PI          4
  PO       2176
  AIG       532
------------------
  Total    2712
```

最後結果：

```
Total time used  : 4.7 seconds
Total memory used: 4.293 M Bytes
```

(2) Reference program 的實驗結果

```
cirr sim06.aag
Period time used : 0.01 seconds
Total memory used: 0.5781 M Bytes

cirSw
cirp
Circuit Statistics
==================
  PI          4
  PO       2176
  AIG      2176
------------------
  Total    4356
Period time used : 0 seconds
Total memory used: 0.5781 M Bytes

cirSim -r
512  patterns simulated.
cirp –fec
(共 250 個 FEC groups)
Period time used : 0.01 seconds
Total memory used: 0.5781 M Bytes

cirFraig
Period time used : 0.45 seconds
Total memory used: 1.621 1 M Bytes

cirp
Circuit Statistics
==================
  PI          4
  PO       2176
  AIG       531
------------------
  Total    2711
```

最後結果：

```
Total time used  : 0.47 seconds
Total memory used: 1.621 M Bytes
```

(3) 比較與討論

由實驗結果可以看出，在記憶體使用量方面，由於我的 CirMgr 存了很多 data member，且有許多是 list，因此一開檔記憶體使用量便很大，約是 reference 的 2 倍。最後總記憶體使用量則約是 2.7 倍。

在花費時間方面，由於我的 simulation 是用 vector 存，沒有使用 hash，因此取值時時間複雜度是 O(n)，比 hash 的 O(1) 慢了很多。因此最後花費時間是 reference 的 10 倍。

另外，由於記憶體使用量與花費時間皆與電路大小有關，當檔案變大時，我的程式與 reference 的差距將會愈來愈大。主要的瓶頸在於 implement 時未考慮 random access 與 hash，造成 linear time 與 constant time 的差別。

六、結論

(1) 此程式在記憶體使用量與執行效率上皆有可改進之處。若能減少不必要的 data member，並盡量使用時間複雜度低的資料結構實作，應可有效改善。

(2) 在對較大檔案做 fraig 以及部分開檔錯誤情況下，可能會有記憶體錯誤的問題。

(3) 總體而言，這是一個大致能執行，並且輸出結果並不算太差的程式。