Fraig

B06502158 陳柏元
email:b06502158@ntu.edu.tw

I. Introduction

此專案的主要思想是創建一個程式來管理/優化/簡化 AIG 電路。

（作業 6 有引用網路上資源：https://github.com/yan12125/DSnP/tree/master/hw6）

整個程序的結構主要由兩部分組成：類別 CirGate 和類別 CirMgr。

母類別 CirGate 處理所有 Gate（子類別）的功能以及存儲大多數信息。類別 CirGate 的每個子類別都可以視為電路中的子元件。正是因為要管理所有電路中的子類別（元件），我們必須創建一個操作(母)類別 CirGate 來讓所有子類別（AndGate,Input/OutputGate...）去繼承他，並實現各種子元件所需要的功能。

類別 CirMgr 可以將其視為一電路板，所有的 Gate 都掛街到類別 CirMgr 上，所以如果想對整體路進行整理、重建等行為的話，變需使用類別 CirMgr 上的各式功能和變數。因此，我們可以在此類中完成許多結構功能，大多數優化等行為也將發生在此類中。

II. Implementation

在這部分，我將介紹一些特殊的 member 和功能。

1. CirGate:

對於所有的 CirGate 的子類別，每個存儲自己的 ID，第幾行讀入，為何種 Gate。為了連接 gate 與 gate，我使用兩個 vector<unsigned int>來記錄其 fanin 和 fanout。為表示是否在 DFS 中是否被經過，因此額外的紀錄了 dfsOrderWithUNDEF_flag 跟 dfsOrderWith_out_UNDEF_flag。我還存儲了其他一些數據在各子類別上，和一些 function，例如：

Members：

CirIOGate:

unsigned int id：紀錄 InputGate 本身的 ID, OutputGate 的輸入 Gate 的 ID。

bool inverted：是否為 invert。

string name：read 的時候取的名字。

int n：OutputGate 本身 ID。

CirAndGate：

unsigned int pin[3]：本身, fanin1, fanin2 的 ID。

bool inv[3]：本身, fanin1, fanin2 是否為 invert。

CirUndefGate：

CirConstGate：

Functions：

getTypeStr()：Gate 型式。

getID()：取得 Gate ID。

isInvert(CirGate*)：取得是否為 invert。

reportGate()，reportFanin(int level)，eportFanout(int level)：print 用的。

2. CirMgr:

Members:

fstream* fCir:存取讀進來檔案。

hasCircuit：是否有電路了。

M：maximum variable index。

I：number of inputs。

L：number of latches。

O：number of outputs。

A：number of AND gates。

CirGate** gates：Gates 的 pointer。

vector<unsigned int> PI：InputGates。
vector<unsigned int> PO：OutputGates。
vector<unsigned int> undefs：未定義 Gate。
vector<unsigned int> dfsOrderWithUNDEF：DFS list 當中不包含未定義 Gate。
vector<unsigned int> dfsOrderWith_out_UNDEF：DFS list 當中包含未定義 Gate。
vector<unsigned int> AIGinDFSOrder：DFS list 當中 AIG Gate。
vector<unsigned int> notInDFS_DfButNtUs：未在 DFS list 中但包含定義未使用 Gate。
set<unsigned int> notInDFS_withoutUNDEF：未在 DFS list 當中且不包含未定義 Gate。
vector<unsigned int> floatingFanin：Fanin PI 無法到達。

Functions：

CirGate* getGate(unsigned gid) const：return '0' if "gid" corresponds to an undefined gate.
bool readCircuit(const string&)：讀 AAG 檔。
void printSummary() const：印 AIG, PI, PO 及總數。
void printNetlist() const：印 DFS 所經過之 Gate。
void printPIs() const：印 PI Gate。
void printPOs() const：印 PO Gate。
void printFloatGates() const：印 Fanin PI 無法到達 Gate。
void printFECPairs() const：
void writeAag(ostream&) const：寫出檔案。
void writeGate(ostream&, CirGate*) const;寫出 Gate。
void sweep()，void optimize()，void strash()：後詳述。
unsigned int buildDFSOrderWithUNDEF(CirGate*, unsigned int)：建立 DFS 並包含未定義 Gate。
unsigned int buildDFSOrderWith_out_UNDEF(CirGate*, unsigned int)：建立 DFS 但不包含未定義 Gate。
void dealWithDFS(bool, bool)：為上述兩建立 DFS 的方式建立一個使用 table。
void cleanDFS_flag(bool, bool)：將 dealWithDFS(bool, bool)所經過 Gate 的注記消除。
void removeTable(vector<bool>&, bool, bool, bool, bool, bool)：為 sweep 建立 table 來消除 DFS 以外不存在的 Gate ID。
void merging(unsigned int, unsigned int,int)：將多餘 Gate 消除並結合。
void rematchFanin(unsigned int,unsigned int,vector<unsigned int>*)：重新掛接 merge 後的 Gate 的 Fanin。
void rematchFanout(unsigned int,unsigned int,vector<unsigned int>*)：重新掛接 merge 後的 Gate 的 Fanout。
HashKey getKey(CirGate* cur)：取得 Hashtable 的 HashKey。

##

void sweep()：

```
void
CirMgr::sweep(){
  //利用包含未定義Gate的DFS來建立不在DFS中且不包含未定義Gate的GateList,notInDFS_withoutUNDEF,
  notInDFS_withoutUNDEF.clear();
  for(unsigned int i=1;i<=M;++i){
    if(gates[i]){
      notInDFS_withoutUNDEF.insert(i);
    }
  }
  for(unsigned int i = 0;i < dfsOrderWithUNDEF.size();++i){
    notInDFS_withoutUNDEF.erase(dfsOrderWithUNDEF[i]);
  }
  vector<bool> removeList;
  removeList.reserve(M+O+1);
  for(unsigned int i = 0;i < M+O+1;++i){
    removeList.push_back(false);
  }
  //利用notInDFS_withoutUNDEF建立removeList(DFS外的所有Gate),
  for(set<unsigned int>::iterator it = notInDFS_withoutUNDEF.begin();it != notInDFS_withoutUNDEF.end();++it){
    if(gates[*it]->gateType == PI_GATE){
      removeList[*it] = false;
    }else{
        cout << "Sweeping: " << gates[*it]->getTypeStr() << "(" <<  reinterpret_cast<CirAndGate*>(gates[*it])->pin[0] << ") removed...\n";
        removeList[*it] = true;
    }
  }
  //更新未定義Gate（UNDEF）,floating的fanin,跟定義但未使用Gate,
  removeTable(removeList,1,1,1,0,1);
  //將removeList中的Gate刪除
  for(unsigned int i=0;i<=M;++i){
    if(removeList[i]){
        delete gates[i];
        gates[i] = NULL;
    }
  }
  //因不能刪除PI，所以更新DFS外無法使用的PI Gate,
  for(unsigned int i = 0; i< PI.size();++i){
    if(gates[ PI[i]/2]->fanout.empty()){
        notInDFS_DfButNtUs.push_back(PI[i]/2);
    }
  }
  //將重複的PI Gate刪除
  sort(notInDFS_DfButNtUs.begin(), notInDFS_DfButNtUs.end());
  notInDFS_DfButNtUs.erase(unique(notInDFS_DfButNtUs.begin(), notInDFS_DfButNtUs.end()), notInDFS_DfButNtUs.end());
}
void CirMgr::removeTable(vector<bool> &removing,bool _undef, bool _floatingfanin, bool _notinDFS_dfbtntus, bool _notinDFS_2, bool _fanout){
   if(_undef){
      for(vector<unsigned int>::iterator it = undefs.begin();it != undefs.end();){
         if(removing[*it]){
            it = undefs.erase(it);
         }else{
            ++it;
         }
      }
   }
   if(_floatingfanin){
      for(vector<unsigned int>::iterator it = floatingFanin.begin();it != floatingFanin.end();){
         if(removing[*it]){
            it = floatingFanin.erase(it);
         }else{
            ++it;
         }
      }
   }
   if(_notinDFS_dfbtntus){
      for(vector<unsigned int>::iterator it = notInDFS_DfButNtUs.begin();it != notInDFS_DfButNtUs.end();){
         if(removing[*it]){
            it = notInDFS_DfButNtUs.erase(it);
         }else{
            ++it;
         }
      }
   }
   if(_notinDFS_2){
      for(set<unsigned int>::iterator it = notInDFS_withoutUNDEF.begin();it != notInDFS_withoutUNDEF.end();){
         if(removing[*it]){
            it = notInDFS_withoutUNDEF.erase(it);
         }else{
            ++it;
         }
      }
   }
   if(_fanout){
      for(unsigned int i=0;i<=M;++i){
         if(gates[i]){
            for(vector<unsigned int>::iterator it = gates[i]->fanout.begin();it != gates[i]->fanout.end();){
               if(removing[*it]){
                  it = gates[i]->fanout.erase(it);
               }else{
                  ++it;
               }
            }
         }
      }
```

void optimize():

```
void
CirMgr::optimize(){
  for(vector<unsigned int>::iterator it = AIGinDFSOrder.begin();it != AIGinDFSOrder.end();++it){
    if(gates[*it]->fanin[0] == 1 || gates[*it]->fanin[1] == 1){//optimize()分為四種AND Gate的fanin型式，merge涵式為merge(欲結合其他人的Gate ID,被結合（將消失）的Gate ID)
      if(gates[*it]->fanin[0] == 1){
        merging((gates[*it]->fanin[1]/2)*2,2*(*it)+gates[*it]->fanin[1]%2,1);//[1]+[..]
      }else{
        merging((gates[*it]->fanin[0]/2)*2,2*(*it)+gates[*it]->fanin[0]%2,1);//[..]+[1]
      }
    }else if(gates[*it]->fanin[0] == 0 || gates[*it]->fanin[1] == 0){
      if(gates[*it]->fanin[0] == 0){
        merging(0, 2*(*it),1);//[0]+[..]
      }else{
        merging(0, 2*(*it),1);//[..]+[0]
      }
    }else if(gates[*it]->fanin[0] == gates[*it]->fanin[1]){
      merging((gates[*it]->fanin[1]/2)*2,2*(*it)+gates[*it]->fanin[1]%2,1);//[..]+[..],[!..]+[!..]
    }else if(gates[*it]->fanin[0]/2 == gates[*it]->fanin[1]/2){
      merging(0, 2*(*it),1);//[!..]+[..]
    }
  }//optimize需處理UNDEF Gate，並將其刪除
  for(vector<unsigned int>::iterator it = undefs.begin();it != undefs.end();){
    if(gates[*it]->fanout.empty()){
      delete gates[*it];
      gates[*it] = NULL;
      it = undefs.erase(it);
    }else{
      ++it;
    }
  }//因merge會改變DFS，重新建立DFS
  dealWithDFS(1,0);
  cleanDFS_flag(1,0);
  floatingFanin.clear();//Floating Fanin會因UNDEF的刪除，所以需更新Floating Fanin
  for(vector<unsigned int>::iterator it = undefs.begin();it != undefs.end();++it){
    vector<unsigned int>& fanoutList = gates[*it]->fanout;
    for(vector<unsigned int>::iterator it2 = fanoutList.begin();it2 != fanoutList.end();++it2){
        floatingFanin.push_back(*it2);
    }
  }
  sort(floatingFanin.begin(), floatingFanin.end());
  floatingFanin.erase(unique(floatingFanin.begin(), floatingFanin.end()), floatingFanin.end());
  notInDFS_DfButNtUs.clear();//merge也會導致產生一些PO連不到的Gate，所以也需更新notInDFS_DfButNtUs
  for(unsigned int i = 1;i <= M;i++){
    if(gates[i]){
      if(gates[i]->gateType != UNDEF_GATE && gates[i]->fanout.empty()){
        notInDFS_DfButNtUs.push_back(i);
      }
    }
  }
  sort(notInDFS_DfButNtUs.begin(), notInDFS_DfButNtUs.end());
  notInDFS_DfButNtUs.erase(unique(notInDFS_DfButNtUs.begin(), notInDFS_DfButNtUs.end()), notInDFS_DfButNtUs.end());
}
```

```
void CirMgr::merging(unsigned int _merge, unsigned int _merged, int _mergedID){
  if(_mergedID==1)//處理merge型式
  cout << "Simplifying" << ": " << inv(_merge) << _merge/2 << " merging " << inv(_merged) << _merged/2 << "..." << endl;
  else if(_mergedID==2)
  cout << "Strashing" << ": " << inv(_merge) << _merge/2 << " merging " << inv(_merged) << _merged/2 << "..." << endl;
  if(_merge == 0){//[0]+[..] or [0]+[..]
    if(gates[_merged/2]->fanin[0] == 0 || gates[_merged/2]->fanin[1] == 0){
      unsigned int target;
      if(gates[_merged/2]->fanin[0] == 0){
        target = gates[_merged/2]->fanin[1] ;//[0]+[..]
      }else{
        target =  gates[_merged/2]->fanin[0];}//[..]+[0]
      rematchFanout(_merged/2, 0, &gates[_merged/2]->fanout);//重新建立Fanout,連接到即將刪除Gate的Fanin
      rematchFanin(_merged/2, _merged%2, &gates[_merged/2]->fanout);//重新建立Fanin,連接即將刪除Gate的Fanout的Fanin
      if(target/2!=0){//若非[0]+[0]
        rematchFanout(_merged/2,target, NULL);//重新建立Fanout,連接到即將刪除Gate的Fanin
      }
    }else {//![..]+[..]
      for(vector<unsigned int>::iterator it = gates[_merged/2]->fanout.begin();it != gates[_merged/2]->fanout.end();++it){
        gates[0]->fanout.push_back(*it); // CONST 0 is not in original fanout, so add directly
      }
      rematchFanin(_merged/2, 0,&gates[_merged/2]->fanout);
      rematchFanout(_merged/2, gates[_merged/2]->fanin[0], NULL);
      if(gates[_merged/2]->fanin[1] != gates[_merged/2]->fanin[0]){
        rematchFanout(_merged/2, gates[_merged/2]->fanin[1], NULL);
      }
    }//[1]+[..]or[..]+[1]or[..]+[..]or![..]+![..]
  }else if(gates[_merged/2]->fanin[0]/2 == _merge/2 || gates[_merged/2]->fanin[1]/2 == _merge/2){
    unsigned int target;
      if(gates[_merged/2]->fanin[0] == _merge/2){
        target = gates[_merged/2]->fanin[1] ;
      }else{
        target =  gates[_merged/2]->fanin[0];}
    rematchFanout(_merged/2, _merge, &gates[_merged/2]->fanout);
    if(target == 1){
      rematchFanout(_merged/2, 0, NULL);
    }else if(gates[_merged/2]->fanin[0]/2 != gates[_merged/2]->fanin[1]/2){
      rematchFanout(_merged/2,target,NULL);
    }
    rematchFanin(_merged/2, _merge ^ (_merge%2 != _merged%2),&gates[_merged/2]->fanout);
  }else{//strash
    rematchFanout(_merged/2,gates[_merged/2]->fanin[0],NULL);
    if(gates[_merged/2]->fanin[0]/2 != gates[_merged/2]->fanin[1]/2){
      rematchFanout(_merged/2,gates[_merged/2]->fanin[1],NULL);
    }
    rematchFanin(_merged/2, _merge ,&gates[_merged/2]->fanout);
    for(vector<unsigned int>::iterator it = gates[_merged/2]->fanout.begin();it != gates[_merged/2]->fanout.end();++it){
      gates[_merge/2]->fanout.push_back(*it);
    }
  }
  delete gates[_merged/2];
  gates[_merged/2] = NULL;
```

void strash():

```
HashKey
CirMgr::getKey(CirGate* cur)
{//取得Hashkey
  assert(cur->getTypeStr() == "AIG");
  CirAndGate* _cur = reinterpret_cast<CirAndGate*>(cur);
  unsigned int a = _cur->fanin[0], b = _cur->fanin[1];
  if(a < b){//這樣才能確保fanin[a]+[b]和[b]+[a]的Key是一樣的
    return HashKey(a, b);
  }
  else {
    return HashKey(b, a);
  }
}
/************************************/
/*   Static varaibles and functions   */
/************************************/

/*****************************************/
/*   Public member functions about fraig   */
/*****************************************/
void
CirMgr::strash()
{
  HashMap<HashKey, CirAndGate*> hash(getHashSize(M));//建立hashtable
  for(vector<unsigned int>::iterator it = AIGinDFSOrder.begin();it != AIGinDFSOrder.end();++it){
    HashKey k = getKey(gates[*it]);
    CirAndGate* _merge = reinterpret_cast<CirAndGate*>(gates[*it]);
    if(!hash.insert(k, _merge)) {//檢查hashkey k是否已有值
        CirAndGate* _merged;
        hash.check(k, _merged);//若已有，取得_merged（即將被取代Gate）
        merging(_merged->pin[0]*2,(*it)*2,2);//取代
    }
  }
  dealWithDFS(1,0);//因更動DFS，所以需重新建制DFS
  cleanDFS_flag(1,0);
  floatingFanin.clear();//floatingFanin有可能更動，需檢查更新
  for(vector<unsigned int>::iterator it = undefs.begin();it != undefs.end();it++){
    vector<unsigned int>& fanoutList = gates[*it]->fanout;
    for(vector<unsigned int>::iterator it2 = fanoutList.begin();it2 != fanoutList.end();it2++){
        floatingFanin.push_back(*it2);
    }
  }
  sort(floatingFanin.begin(), floatingFanin.end());
  floatingFanin.erase(unique(floatingFanin.begin(), floatingFanin.end()), floatingFanin.end());
}
```

III. Performance and Discussion
Use the largest case as benchmark: sim13.aag

My program	Reference
fraig> cirr sim13.aag fraig> usage Period time used : 0.13 seconds Total time used : 0.13 seconds Total memory used: 16.2 M Bytes	fraig> cirr sim13.aag fraig> usage Period time used : 0.04 seconds Total time used : 0.04 seconds Total memory used: 13.93 M Bytes
fraig> cirsw fraig> usage Period time used : 0.19 seconds Total time used : 0.32 seconds Total memory used: 19.93 M Bytes	fraig> cirsw fraig> usage Period time used : 0 seconds Total time used : 0.04 seconds Total memory used: 13.93 M Bytes
fraig> ciropt fraig> usaGE Period time used : 0.14 seconds Total time used : 0.14 seconds Total memory used: 16.24 M Bytes	fraig> ciropt fraig> usAGE Period time used : 0.08 seconds Total time used : 0.08 seconds Total memory used: 14.65 M Bytes
fraig> cirstrash fraig> uSAGE Period time used : 0.22 seconds Total time used : 0.22 seconds Total memory used: 19.55 M Bytes	fraig> cirstrash fraig> uSAGE Period time used : 0.07 seconds Total time used : 0.07 seconds Total memory used: 17.7 M Bytes

在觀察中，儘管許多功能比參考程式慢，但他們都與參考 Diff 相同。 瓶頸大概就是，花了蠻多時間找出 optimize 中 merging 產生的各種問題，後悔大概就是使用別人的作業 6，儘管或去他能在作業六得到高分，除了有許多地方需要修改，最關鍵的大概就是需要花很多時間理解別人寫的 code，正因作業 6 是機械系很多必修的期中考，導致當時沒太多時間完成。程式的邏輯可以很快理解，但常常花時間的反而是了解他的變數名稱，以及瞭解他建構 AAG 的方式。
這次期末 project，因期末參加比賽，期末考完後才開始寫，慌亂的打 code 只完成了前三項，未完成的部份有 sim 跟 fraig，希望能再暑假之後完成，即便拿不到分數了 XD。