FRAIG Report

Dsnp final project

電機三 b03901191 劉馥瑄

contact information:

email: b03901191@ntu.edu.tw

mobile: 0921425016

Introduction

This project's main idea is to design a program to optimise and simply AIG circuits in mainly two parts: CirGate and CirMgr.

CirGate:

- Create gate, store information of each gate.
- First design a base class of Gate, different kinds of gates derived from it.
- Functions implementation about handle per-gate.

CirMgr:

- Act as a manager.
- Build up a circuit between gates and gates.
- Functions about structural-commands performance.

Design of Data Structure

CirGate

basic information of each gates: ID, line#in aag file, name(type)…

A template class, and then different types of gate inherit it :

(ex: class PIGate: public CirGate)

```
CirGate(unsigned gid = 0, unsigned lineNo = 0)
   : gateid(gid), lineNum(lineNo), name(""){
   invert_or_not.resize(2, true);          true: not invert
   float_or_not = false;
   def_or_not = false;
   in[0] = in[1] = 0;                      for every gate at most two fanin
   op = 0;                                 value of gate output (ex: 1* 2 = 2)
 }
 virtual ~CirGate() {fanin.clear(); fanout.clear();}
```

gate class: PIGate, POGate, ANDGate, UNDEFGate, CONSTGate

- special member and functions:

members:

vector<CirGate*> fanin;	vector type is very flexible to maintain
vector<CirGate*> fanout;	and handle Cirgate*, also easy to attach!
vector<bool> invert_or_not;	see from RHS to LHS, discuss about
	fanin line.
bool float_or_not;	
bool def_or_not;	

functions:

void myFanin(CirGate*, unsigned, unsigned, bool);	
void myFanout(CirGate*, unsigned, unsigned, bool);	
	recursively call the func., and mark gate.
bool isGlobalRef() { return (_ref == _globalRef); }	
void setToGlobalRef() { _ref = _globalRef; }	
static void setGlobalRef() { _globalRef++; }	
	Very useful to mark nodes been traveled!

CirMgr

Store gates in a GateList glist, which its index is the ID of gate.(ex: glist[i]: store a gate with ID = i). By the way, glist[0] stores a CONST type gate.

- special member and functions:

members:

GateList glist;	store original data gates (type: CirGate*)
GateList DFSlist;	store gates run in DFS order
vector<unsigned> trueAIG;	store legal AND gates in dfs
vector<unsigned> truePO;	POs in DFSlist, stores id
vector<unsigned> finalPO;	(this and above function is for writeaag( )
	after ciropt)

functions:

bool readHeader(string&);	
void connectCircuit(CirGate*, CirGate*, int);	
void buildCircuit(GateList&);	this two build up circuit & connect gates

void DFSTraversal(CirGate*);	inner func of dfslistbuild(), recursively call
void DFSlistbuild();	use in many commands functions !
void sweep();	
void cutline(CirGate* &);	cut the sweeping-gate's fanin and fanout also delete the gate. (for sweep)
void optimize();	
void myOptimize(CirGate*);	discuss 4 cases.(call by optimize())
void kill(CirGate* &a , CirGate* b , bool inv = true);	a: cursor gate, b: pre-gate connect to a, inv: a's input line invert or not. reconnect b -> a's fanout, handle fanning fanout, delete gates (call by myOptimize())

Implementation

CIRSWeep: Scan through glist, find AIG not in DFSlist , then call cutline( ) function to remove.

CIROPTimize: Scan through DFSlist

—> consider four cases:

1. Const 0 fanin : replace with constant0
2. Const 1 fanin : replace with other fanin
3. x x : replace with x
4. x x' : replace with constant 0

—> call kill(d, pre, inv).

CIRSTRash: Using HashMap. The hash function in HashKey:

For each fanin in AIG, invert: (id*2+1), not: (id*2),
—> send a pair to the constructor. (smaller one in the left, and another one in the right.)
Therefore create distinct keys for every different fanin.
—> insert all AIG gates into the hash, replace all repeated gates.

CIRSim: Do not have enough time….Sorry T_T

CIRFraig: Do not have enough time….Sorry T_T

Results and analysis

• comparison

> take sim13.aag for testing:

	My Fraig	Ref Fraig
fraig> cirr	`Period time used : 0.28 seconds` `Total time used  : 0.28 seconds` `Total memory used: 21.37 M Bytes`	`Period time used : 0.06 seconds` `Total time used  : 0.06 seconds` `Total memory used: 12.02 M Bytes`
fraig> cirsw	`Period time used : 26.84 seconds` `Total time used  : 26.84 seconds` `Total memory used: 23.05 M Bytes`	`Period time used : 0 seconds` `Total time used  : 0.06 seconds` `Total memory used: 12.04 M Bytes`
	`Period time used : 0.03 seconds` `Total time used  : 26.87 seconds` `Total memory used: 23.05 M Byte`	`Period time used : 0.01 seconds` `Total time used  : 0.07 seconds` `Total memory used: 12.15 M Bytes`
fraig> cirstra	`Period time used : 0.08 seconds` `Total time used  : 26.95 seconds` `Total memory used: 24.42 M Bytes`	`Period time used : 0.03 seconds` `Total time used  : 0.1 seconds` `Total memory used: 16.05 M Bytes`
fraig> cirw	`Period time used : 0.46 seconds` `Total time used  : 27.41 seconds` `Total memory used: 24.5 M Bytes`	`Period time used : 0.28 seconds` `Total time used  : 0.38 seconds` `Total memory used: 16.05 M Bytes`

This is the largest case in tests.fraig folder, but
In small data such as "do.opt", my code has similar speed and memory used in reference code.

From the Table above, we can see the main difference between reference code and mine, its the running time that function run-through DFS. I think that because I call dfs every time when needed, it does waste a lot of time. Moreover, my dfstraversal is call in a recursive way. But in fact, this file have nothing to sweep, that is, no undef, floating gates. Solving way: build up a unused gatelist at read in process, then before running dfs, we can determine whether unused list.empty().

Talk about memory used, the way I store data is a constant size, which decided at the read-in process, total size is an array with (M+O+1) which is max Id, disadvantage may be memory size, but for me, it's much easier to find and think. Other factors resulting in larger memory bytes: when create a vector list or GateList(CirGate*), GateList is larger for it store much information. In addition, size_t, unsigned, int…. have different size when create an variable.

Discussion

This project need highly concentrated on handling details. Especially the memory allocate, or access invalid data… informations need to be update immediately to maintain the circuit well.

Here are some of most impressive bottlenecks I conquered. For example, I forgot to clear up the fanins and fanout, which cause a big fatal problem , every time replacing a new circuit, randomly have segmentation fault.
Another problem that bother me a lot is that when I test strash05.aag, I found my netlist and fanin cannot access to a floating UNDEF gate! Fortunately, after modifying my buildCircuit(), and dfs functions, it does work.
For Optimize function, carefully check the cirw (writeAag function), the output will be changed after call optimize(). There are also many cases and conditions we should take care and pay attention on in this final project.
Strash, the core idea of the function is the design of its HashMap, HashKey, this function is not take as long as the above functions' time to write, but I thought it's more deep than the above two functions.
I was stuck to a problem that after merging gates, do sweep might be segmentation fault or doing wrong, unnecessary sweeping.

Regrettably, I don't have enough time to complete the following two complex functions, still hope that I can realize the concept of Simulation and FRAIG, also comprehend the data structure of this project.

Comments to the class

Glad to have a opportunity to take this awesome course. For me, Dsnp is not an easy course, actually, coding for me was a suffer in the past.(now…?)
During this semester, I have more experience at coding, learn how to find open sources, references, also debug by myself(cout ..XD). In my opinion, it's great if the course can introduce more detailed about Data Structure. And realize the linkage between every single file, (or even have an ability to write) is very significant, too. Last, Thanks to Prof. Ric-Huang and all the TAs of Dsnp.
Happy lunar New Year ~XD.