CS6132 Advanced Logic Synthesis

Midterm Project Report

111062584 王領崧

- 實驗流程:
  此次作業使用 SIS & ABC 所提供的 script 來優化電路, 透過實驗來探討不同 script 對不同 benchmark 的影響, 藉此分析 script 的作用與 benchmark 的特性, 以及比較 SIS 和 ABC 的優化成果, 並且也嘗試執行不同數量的 iterations, 來觀察 script 是否有持續優化電路的能力。優化成果分為 quality 和 runtime。
  Quality 我使用 print_stats 數據中的 node 數量 (Area) 作為標準, 如果是單就 ABC 不同 script 的比較, 則用 AND gates 作為標準。由於每個 benchmark 的大小不同, 因此我還會用 reduction rate, 也就是 (script - original) / original 來作為分析的依據。
  Runtime 則是用 time command 紀錄 script 優化的執行時間。

- 實驗設定:
  ABC script 使用的有 resyn2, resyn3, compress2, choice2, rwsat, share。(沒有使用的 script 是因為和其他 script 相似 or 無法執行)。
  SIS script 使用的有 script.rugged。
  Benchmark 使用的有 alu4, C6288, C7552, count, dalu, i10, rot, t481, too_large, vda, 總共 10 個, 都是 nodes 數量比較大的。

- 實驗結果和分析:
  1. ABC script results (1 iteration)
     這裡為利用 ABC scripts 對每個 benchmark 各執行一次後的結果。Table 1 顯示的是執行後 AND gates 的數量, Chart 1 是執行後的 reduction rate。
     由實驗結果得知, 大部分的 scripts 在優化後都能減少 AND gates 的數量, reduction rate 在 15% ~ 60% 不等, 其中又以 share 的表現最好, 而 choice2 的表現最差, 甚至還會增加 AND gates 的數量。
     比起其他 scripts 是直接對 AIG graph 進行 balance / rewrite / refactor 等優化, Share 會先將 function 轉換為 multi-input AND-gates network, 透過 two-cube divisors 來找 logic sharing 後, 再用上述方式進行優化, 我認為這樣能增加 optimization space, 使得優化成果是最好的。
     Choice2 使用 Fraiging 作為轉換成 AIG network 的工具, 但根據 document 可以得知 fraiging 產生 AIG node 的 function 具備獨特性, 不會跟其他 node 相同, 我認為這一定程度限制了 optimization space, 使得執行後的結果變得更差。

	Original	resyn2	resyn3	compress2	choice2	rwsat	share
alu4	735	652	676	653	1216	665	630
C6288	2337	1870	2334	1870	3380	2305	1870
C7552	2074	1455	1830	1408	4295	1508	1393
count	127	112	127	112	223	112	98
dalu	1371	1106	1146	1083	1810	1165	826
i10	2675	1829	2122	1816	4631	1933	1676
rot	550	479	492	478	810	477	432
t481	1874	802	1208	671	4050	797	590
too_large	824	460	510	459	1477	464	367

vda	924	622	674	595	2430	630	526

Table 1: AND gates of ABC (1 iteration)

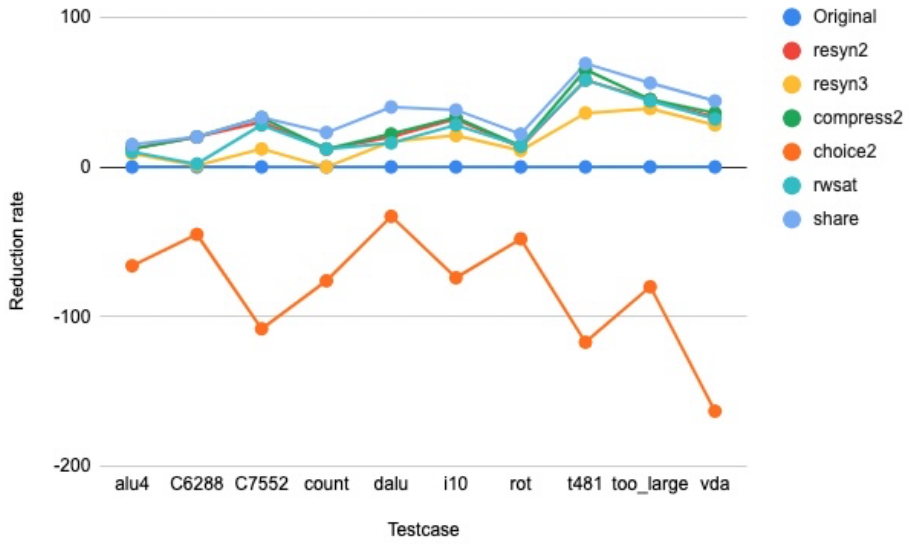

Chart 1: Reduction rate (AND gates) of ABC (1 iteration)

Chart 2 為執行後 AIG graph level 的 reduction rate。由實驗結果得知，大部分優化後 level 都能有所減少，但並沒有某個 script 的成果特別突出。我覺得原因出在 ABC 中，要減少 critical path 只能透過 balance 來達成，而 balance 又和查表得到的 AIG structure 有關，變成所有 script 都可能在 rewrite 過程取得，使得最後的 level 能有更好的表現。

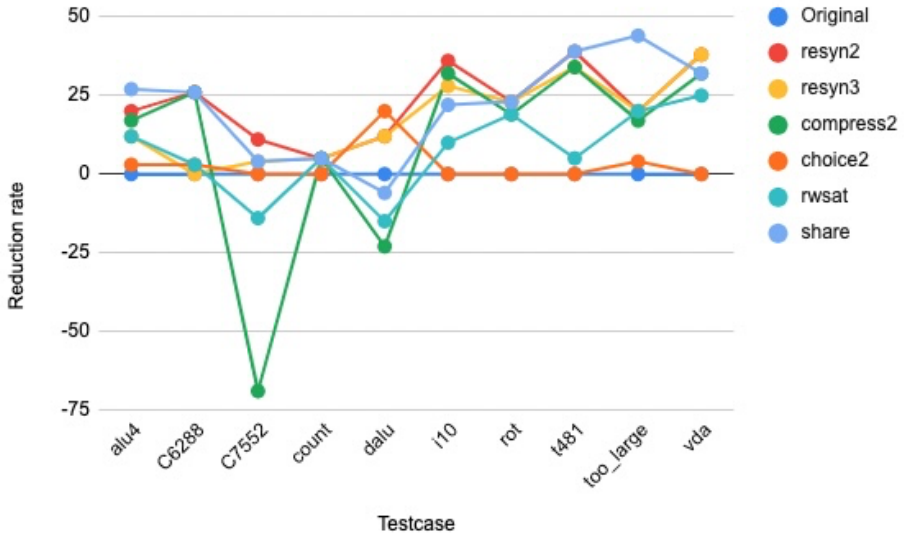

Chart 2: Reduction rate of ABC (1 iteration)

2. ABC script results (1, 5, 10 iterations)

   這裡為 ABC scripts 對每個 benchmark 分別執行 1, 5, 10 iterations 後的結果。Table 2 顯示的是執行 1, 5, 10 iterations 的 **reduction rate improvement**。

   由實驗結果得知大部分的 script 在執行多個 iterations 後 AND gates 不會有明顯的減少，converge 的速度很快，進度幅度 < 10%，唯有 resyn3 和 choice2 在執行多個 iterations 後會有明顯的進步。

   Choice2 我認為單純是因為在第一個 iteration 的表現太差，使得經過多個 iterations 的優化後，才會呈現明顯的進步。

Resyn3 是 script 中唯一大量使用 resub 指令，resub 會嘗試將每個 node 代入 network 中的其他 nodes 來觀察 improvement，由於選擇的 benchmark node 數量都不小，使得 substitution 的組合非常多，因此當嘗試的組合越多，自然而然有更高的機率找到更好的方案，這也使得 resyn3 在執行 10 iterations 後能有明顯進步的原因。

	resyn2	resyn3	compress2	choice2	rwsat	share
alu4	12 / 0 / 0	9 / 0 / 2	12 / 0 / 0	-66 / 14 / 0	10 / 0 / 0	15 / 1 / 2
C6288	20 / 0 / 0	1 / 0 / 19	20 / 0 / 0	-45 / 4 / 0	2 / 0 / 0	20 / -37 / 0
C7552	30 / 3 / 0	12 / 0 / 16	33 / 2 / 1	-108 / 56 / 3	28. / 2 / 0	33 / 4 / 0
count	12 / 0 / 0	0 / 0 / 0	12 / 0 / 0	-76 / -12 / 0	12 / 0 / 0	23 / 0 / 0
dalu	20 / 0 / 0	17 / 0 / 2	22 / 0 / 0	-33 / 11 / 0	16 / 2 / 0	40 / 0 / 0
i10	32 / 2 / 0	21 / 0 / 12	33 / 1 / 1	-74 / 40 / 5	28. / 1 / 0	38 / 1 / 0
rot	13 / 1 / 0	11 / 0 / 0	14. / 0 / 0	-48 / 11 / 0	14. / 1 / 0	22 / -5 / 0
t481	58. / 4 / 1	36 / 0 / 24	65 / 5 / 2	-117 / 18 / 4	58. / 9 / 1	69 / 4 / 3
too_large	45 / 10 / 0	39 / 0 / 10	45 / 7 / -14	-80 / 30 / 17	44 / 5 / 0	56. / 12 / 1
vda	33 / 9 / 1	28. / 1 / 11	36 / 6 / 3	-163 / 44 / 34	32 / 4 / 0	44 / 10 / 3

Table 2: Reduction rate (AND gates) of ABC (1, 5, 10 iteration)

3. ABC script runtime results (1, 5, 10 iterations)
   ABC script 執行速度十分快速，並且不同 script 的差異很小，就算執行 10 iterations 大部分也都可以在 1.5 sec 執行完成，因此鑑於篇幅的關係，這邊就不做過多討論。
4. SIS script.rugged results (1, 5, 10 iterations)
   因為 SIS tool 我只有實驗 script.rugged，所以這裡直接顯示 rugged 在不同的 iterations 下 node reduction rate。Chart 3 顯示的是 rugged 在執行 1, 5, 10 iterations 下的 reduction rate。
   由實驗結果得知，reduction rate 收斂地十分快速，執行 5, 10 iterations 後 node 的數量並沒有明顯的減少，只有測資 too_large 有明顯的進步。我覺得因為 too_large 屬於 node 數量很少，但每個 node 中的 SOP 十分龐大，導致 rugged 在做 factoring 時候，反而創造更多的 nodes 來減少 SOP 的數量，造成執行 1 iteration 後的 rate 是下降的。不過當執行更多 iteration 後，就能透過 factoring 來減少 nodes 的數量，有點類似脫離 local optimal，往更好的 results 前進的概念。

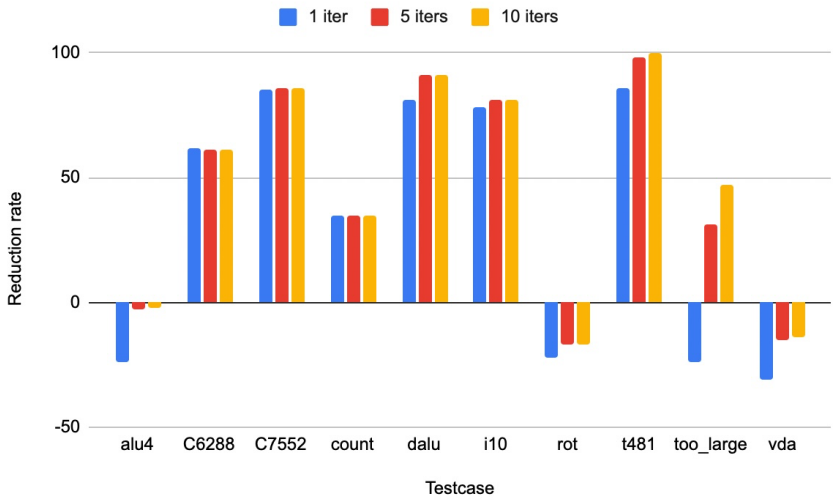

Chart 3: Reduction rate of SIS script.rugged (1, 5, 10 iterations)

5. ABC, SIS scripts node & runtime comparison (1 iteration)

這裡是 ABC 和 SIS script performance 的比較，分別有 node 數量和 runtime。
Chart 4 顯示 SIS 和 ABC script 執行一個 iteration 後，node 數量的 reduction rate (ABC 選擇表現最好的三個來比較)。
由實驗結果得知，rugged 的 reduction rate 遠遠好於 ABC script，並且 ABC script 大部分都是增加 node 的數量，最差的情況甚至會增加至 10 倍。我認為是因為 ABC script 的結果都是 AIG，因此每個 node 一定是 2-input AND gate，然而原始測資和 SIS script 的 node 每個都是由很多 minterms 所組成，導致它們可以使用比較少的 node 來表示 network。不過我們由實驗 1. 可以知道 ABC 能有效地降低 AND gates 的數量，所以我覺得很難從這邊直接比較 SIS 和 ABC 誰的優化成果比較好，可能需要到 mapping 階段才能看出來。

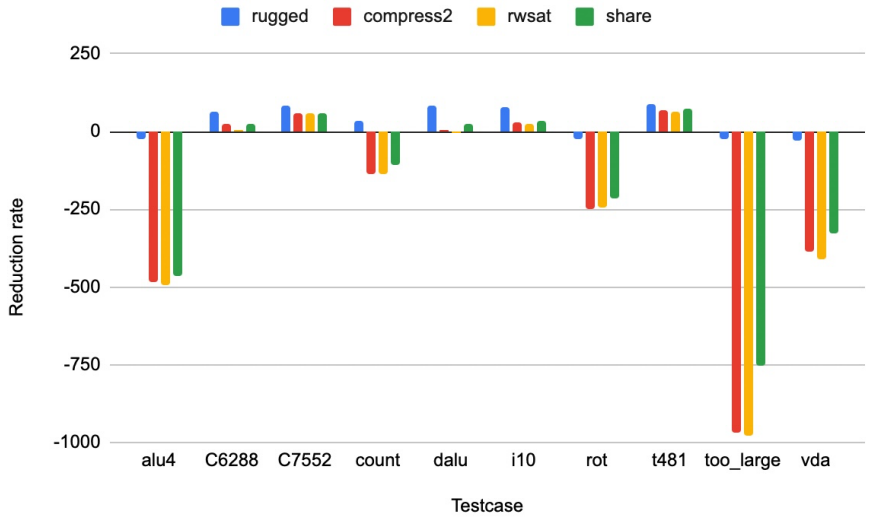

Chart 4: Reduction rate (Nodes) of SIS and ABC (1 iteration)

Table 3 為 SIS 和 ABC 的 runtime。由實驗結果得知，SIS 的 runtime 大概是 ABC 的 100 倍以上，我覺得是因為 rugged 以 factoring 作為優化主體，kernel 的可能性很多，並且每個 function 都要做 algebraic division，總結來看相當耗時；而 ABC 只要在找到 feasible cut 後，透過查表套入 pre-compute 的 AIG graph structure 即可，比較之下執行時間會簡短很多。

	rugged	compress2	rwsat	share
alu4	5.3	0.07	0.05	0.08
C6288	1.5	0.34	0.21	0.27
C7552	3.6	0.13	0.06	0.12
count	0.05	0.01	0.005	0.01
dalu	4.2	0.07	0.04	0.05
i10	25	0.1	0.06	0.1
rot	0.6	0.02	0.01	0.03
t481	6.5	0.05	0.03	0.05
too_large	428.3	0.03	0.02	0.04
vda	1.5	0.04	0.02	0.04

Table 3: Runtime of SIS and ABC (1 iteration)