CS6135 VLSI Physical Design Automation

Homework 2 Report: Two-way Min-cut Partitioning

108062605 呂宸漢

1. 如何編譯及執行程式：

   進入 HW2/src/後，輸入 make 即可編譯程式並輸出執行檔 FM_Partitioner 至 HW2/bin/。

   如：$ make

   之後進入 HW2/bin/，輸入./FM_Partitioner <cell file> <net file> <output file>即可執行程式。

   如：$ ./FM_Partitioner ../testcases/p2-1.cells ../testcases/p2-1.nets ../output/p2-1.out

   注意：cell file 的副檔名須為.cells、net file 的副檔名須為.nets、output file 的副檔名須為.out。

2. cut size and runtime for each case：

	p2-1	p2-2	p2-3	p2-4
cut size	6	272	1449	42950
runtime	0.006	0.14	2.43	5.79

   (以 clock_gettime 的 CLOCK_MONOTONIC 測量)

3. I/O time and computation time for each case：

	p2-1	p2-2	p2-3	p2-4
I/O	0.005	0.06	1.17	2.22
computation	0.001	0.08	1.26	3.57
runtime	0.006	0.14	2.43	5.79

   (以 clock_gettime 的 CLOCK_MONOTONIC 測量)

   因為在 I/O 時會先讀取 cell file 將每個 cell 創一個 unique id，在讀取 net file 便會開始記錄每個 net 有接到哪些 cell 和每個 cell 有接到哪些 net，並且做 balance，因此 I/O 時間有些看起來會比較長，尤其當 cell 及 net 較多時會更慢，反而 FM 的計算時間有些還比 I/O 時間短，因此 I/O 及 balance 算是這個程式的 bottleneck。

   I/O time: *O(C) + N*O(C) + O(N) = O(C*N) (C: number of cell, N: number of net)*

   因為讀取 cell file 要 *O(C)*，讀取 net file 並儲存 cell list 及 net list 要 *N*O(C)*，寫入 output file 要 *O(C)*。

   Computation: *i*C*O(P) (i: iteration, C: number of cell, P: number of pin)*

   因為執行一次 FM 要 *C*O(P)*，*i* 為執行次數。

4. The detail of my implementation：
   I. Where is the difference between your algorithm and FM algorithm described in class?
      在 update gain 的方面我與講義上的 pseudocode 做法相同，一樣是找到 base cell 後，更新 base cell 會影響到的 critical net 的 cell gain。不過講義上沒有特別註明要找兩個 bucket list 中最大的 gain 當 base cell，因此我嘗試先找其中一個 bucket list 的 max gain，在不符合 balance factor 時或 bucklist 為空再找另一個 bucket list 的 max gain，雖然有時另一條 bucket list 的 gain 會比較大，可是以目前 test case 的執行結果而言，這種方法的 cut size 與 runtime 皆比原版的要好，我覺得是因為 initial partition 的方式與 find max gain 的方式相同，才呈現這樣的結果。

   II. Did you implement the bucket list data structure?
      有，bucket list 的結構與講義上相同，每一邊有自己的 bucket list，因此整個程式有兩條 bucket list 須要維護。由於 bucket list 的 index 是從 Pmax 到-Pmax，因此我採用 map 實作 bucket list，在使用上較方便且直觀，另外在寫三個函式 build_bucket_list()、insert_node()與 remove_node()維護 bucket list，在 coding 時比較方便且除錯容易。

   III. How did you find the maximum partial sum and restore the result?
      如同課本上描述，當 partial sum 為 0 時，表示 FM 已經 trace 完所有的 cell，在從中找出 max partial sum 即可。在我的實作中，當 base cell 被 lock 住時，我會將其 node 從 bucket list 移除，所以當兩條 bucket list 皆為空時，也表示 FM 已經 trace 完所有的 cell。在每次 update gain 時，我會用一個 cell_stack 將 base cell 的 id 記錄下來，如果當下的 partial sum 比 max partial sum 更佳時，除了更新 max partial sum 之外，會有一個 best_step 的變數記錄這是第幾個 base cell，當 FM 執行完後，若想 trace back 到 max partial sum 的 partition 時，只需要由後往前讀取 cell_stack，並將 cell 在搬回原本的 set，最後再重新計算每個 net 的 cell 在不同 set 的個數即可還原到 max partial sum 的 partition。

   IV. Compare with top 5 student's result：
      雖然不應該分開來比較 cut size 與 runtime，不過因為不知道 cut size 及 runtime 所佔的比例及如何排名，我先分開來分析 cut size 及 runtime，最後在綜合評分。

cut size					
	p2-1	p2-2	p2-3	p2-4	積分
1	6 (1)	130 (1)	824 (1)	44959 (4)	7
my	6 (1)	272 (5)	1449 (2)	42950 (3)	11
2	6 (1)	221 (4)	1630 (3)	46323 (6)	14
3	16 (6)	332 (6)	2216 (4)	45646 (5)	21
4	6 (1)	193 (2)	3490 (6)	40254 (1)	10
5	8 (5)	210 (3)	2722 (5)	42743 (2)	15

上表為 cut size 的排名，雖然我在 case p2-2 表現較差，可是我在其他 case 的排名均在前半，以所有數據而言，我認為我贏過第二名。

runtime					
	p2-1	p2-2	p2-3	p2-4	積分
my	0.006 (1)	0.140 (6)	2.430 (3)	5.790 (1)	11
1	0.020 (6)	0.098 (4)	9.304 (5)	9.324 (2)	17
2	0.010 (4)	0.112 (5)	2.354 (2)	10.956 (3)	14
3	0.006 (1)	0.096 (3)	3.607 (4)	25.154 (4)	12
4	0.010 (4)	0.074 (2)	10.75 (6)	175.404 (6)	18
5	0.009 (3)	0.060 (1)	1.621 (1)	101.243 (5)	10

上表為 runtime 的排名，雖然我依然在 case p2-2 表現較差，可是我在其他 case 的排名均在前半，以所有數據而言，並用積分計算，我認為我贏過第一名。

綜合上述結果，我認為我的總排名應該還是第二名，因為我的 runtime 數據與第二名較接近，可是 cut size 卻比第二名更小，而與第一名相比，雖然我的 run time 較小，可是相差不大，反而第一名的 cut size 有些只有我的一半，因此我認為與去年數據相比，我應該是第二名。

V. What else did you do to enhance your solution quality or to speed up your program?

其實在實作 FM algorithm 的作法大概都與講義上的相同，所以我大多是透過改變 initial partition 的方式改進 cut size，雖然不同的 test case 也都會有不同的 net 組合，有時候一個 case 很好一個 case 很差，雖然有同學會對不同的 test case 改變 initial partition 的方式，不過我還是只用幾個固定的 initial partition 方式，並找出對每個 case 都不錯的當作最後的 partition 方式，這麼做是因為在不確定 hidden case 的情況下，這麼做雖然不一定會最好，可是至少在其他的 test case 下有較好的 cut size 及 runtime，在 hidden case 表現較好的機率較大。

VI. What have you learned from this homework?

這次作業比較燒腦的地方是在設計儲存 cell 及 net 的資料結構以及回復到 max partition sum 的 partition 的方式，不同的資料結構對執行的速度有很大的差異，還有判斷式放的地方和更新每個 cell gain 的方式也很重要。不過這次卡最久的是寫 bucket list 的時候，當時沒有把 insert 及 remove 的指標改變方式設計好，導致在執行程式的時候一直 segment fault，但是 cell 的數量及 net 的數量又大，花的很多時間才找到到底是更新到哪個 cell gain 時發生的 segment fault，或許是不常用指標的關係，花了一整個晚上在解決這個問題。不過，在這個作業其實學到很多之前沒有碰過的事物，比如 bucket list 的 index 需要從 Pmax 到-Pmax，若以陣列實作可以 shift -Pmax 到 0 即可，可是我發現可以利用 map 去對應 Pmax 到-Pmax，除了節省 coding 的麻煩，也可以增加可讀性；在轉換 cell name 及 net name 到各自的 unique id 時，也可以用 map 去對應，不必再寫判斷式轉換成 id，節省了很多時間。

VII. If you implement parallelization, please describe the implementation details and provide some experimental results.

我沒有實作平行化的程式。

5. 與 hMETIS 之比較：

		p2-1	p2-2	p2-3	p2-4
cut size	hMETIS	5	118	505	35460
	my	6	272	1449	42950
I/O time	hMETIS	0.001	0.008	0.121	0.170
	my	0.005	0.06	1.17	2.22
computation time	hMETIS	0.036	0.317	9.863	101.985
	my	0.001	0.08	1.26	3.57

(以上測試採用 shmetis 進行測試，balance factor 為 5)

比較兩者的 cut size 可以發現 hMETIS 的 cut size 皆較小，以多次測試所得到的結果而言，hMETIS 在規模較小的 case，如：p2-1 及 p2-2，每次得到的 cut size 都相同，我認為 hMETIS 在規模小的 case 有找到最佳的 cut size。

相較之下，雖然 p2-3 及 p2-4 的 cut size 也小很多，不過他每次得到的結果卻不太相同，再參照 hMETIS 的 manual 後發現，hMETIS 在 initial 是偏向 random 的，而且 shmetis 會做 10 次(by default)不同 initial partition 的 partition，再取最小的 cut size 當作最終答案，所以 hMETIS 的 computation time 才會較長，不過得到的 cut size 較佳。

而我實作的 FM initial partition 的方式是固定的，所以每次得到的 cut size 與時間都相同，雖然可以縮小 cut size，但是不一定會是最佳的。