CS6135 VLSI Physical Design Automation

Homework 4 Report: Placement Legalization

108062605 呂宸漢

1. 如何編譯及執行程式：

   進入 HW4/src/後，輸入 make 可編譯程式並輸出執行檔 hw4 至 HW4/bin/。

   如：$ make

   之後進入 HW4/bin/，輸入./hw4 <aux file>即可執行程式。

   如：$ ./hw4 ../testcase/adaptec1/adaptec1.aux

   注意：aux file 的副檔名須為.aux。

2. displacement and runtime for each case：

	adaptec1	adaptec3	ibm01	ibm07	ibm09
total displacement	3922135	7373290	5950950	31396189	48684925
maximum displacement	286.84	350.10	3372.01	10987.77	8961.72
I/O time	0.60	1.32	0.03	0.12	0.14
Abacus time	0.54	2.65	0.01	0.04	0.05
runtime	1.14	3.97	0.04	0.16	0.19
threshold	no	no	yes	no	no

(時間以 clock_gettime 的 CLOCK_MONOTONIC 測量)

Table 1

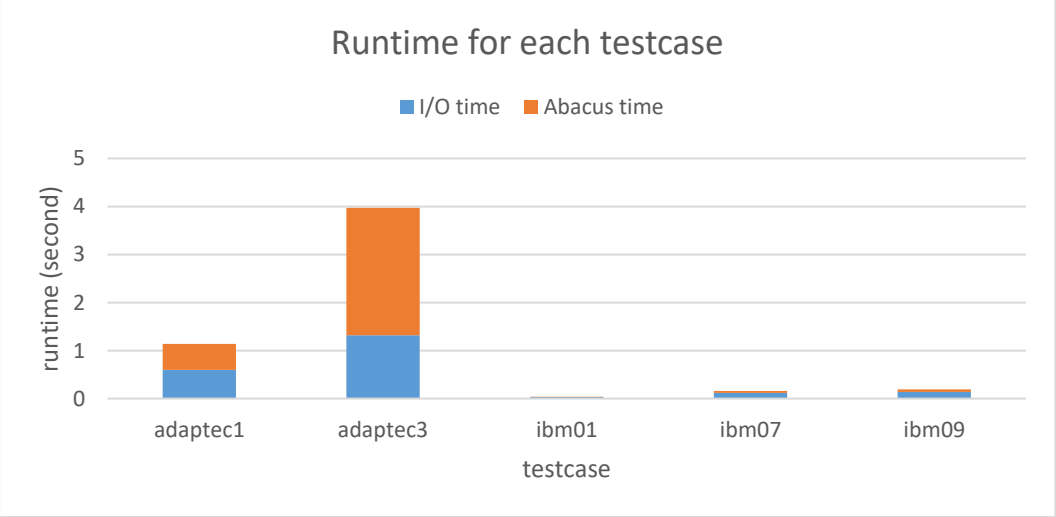

Figure 1

在 I/O 時會先讀取 node file 與 pl file 取得每個 cell 及 terminal 的 x 座標、y 座標、高度及寬度，之後讀取 scl file 取得每個 row 的 y 座標、高度及 x 座標範圍，再利用 terminal 的左邊界及右邊界接 row 切成許多 subrow，並計算可以容納 cell 的空間，以加速之後在 Abacus 尋找 subrow 的過程。

I/O time: $O(C+T) + O(C+T) + O(R) + O(T*R) + O(C+T) = O(C+T*R)$

(C: number of cell, T: number of terminal, R: number of row)

因為讀取 node file 要 $O(C+T)$，讀取 pl file 要 $O(C+T)$，讀取 scl file 要 $O(R)$，切割 subrow 要 $O(T*R)$，寫入 result file 要 $O(C+T)$。

Computation: 即為 Abacus 的時間複雜度

3. the detail of my implementation：

程式流程：

a. 讀取檔案(包含讀取 aux file、node file、pl file 及 scl file)
b. 依照 terminal 切割 subrow
c. 執行 Abacus(除了選擇 subrow 的方式外，其餘與課程相同)
d. 依照 legalize 的結果 align row site
e. 寫入檔案(寫入 result file)

以下列出我的程式與課程 Abacus 的不同處

a. find subrow 不同：

課程 Abacus 選擇 subrow 的方式是將 cell 試擺進該 row 的所有 subrow 後比較他們的 cost，找出 cost 最小的 subrow 後再試擺其他的 row，找到最小的 cost 時紀錄最佳的 row id 及 subrow id，以決定 cell 的最佳位置。不過如果每個 subrow 都試擺太浪費時間，因此我用與了不同的方式，先找到離 cell 最近的 subrow，如果該 subrow 滿了，就往後一個 row 找或往前一個 row 找，雖然可能損失一些較好的位置，不過可以確保移動量可以較小，也比每個 subrow 都算一次的方式更快。

以下呈現不同 find subrow 的方式在 adaptec3 case 的執行結果

	課程	my
total displacement	10973273	7373290
maximum displacement	681.49	350.10
I/O time	1.33	1.32
Abacus time	9.81	2.65
runtime	11.14	3.97
threshold	no	no

Table 2

4. What have you learned from this homework?
   這次作業遇到的第一個問題就是 parser 的 coding，這次的檔案結構比較亂，不同 testcase 的內容格式也不一定相同，因此需要考慮許多的東西，不能像以前直接寫一個讀取指定行數的 parser 就可以 parse。雖然這次的資料結構比較簡單，Abacus 的觀念也很簡單，可是在時作時卻要處理許多例外，尤其是有 terminal (blockage)的 testcase，在切割 subrow 時就要思考會不會有些 terminal 沒有 align 或是超出 subrow 範圍，在找尋最接近的 subrow 時也是，尤其改成我現在的寫法需要更多例外處理。雖然 Abacus 的架構只要依照 paper 的 pseudocode 就可以很快地寫出來，可是例外處理卻讓我想了好一段時間。在最後就是處理 overlap 的問題，需要用很多方式檢查是哪個部分 overlap，再去 trace code 找出可能出問題的部分，幸虧最後有找到一些錯誤才成功解決 overlap 的問題。

5. 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。