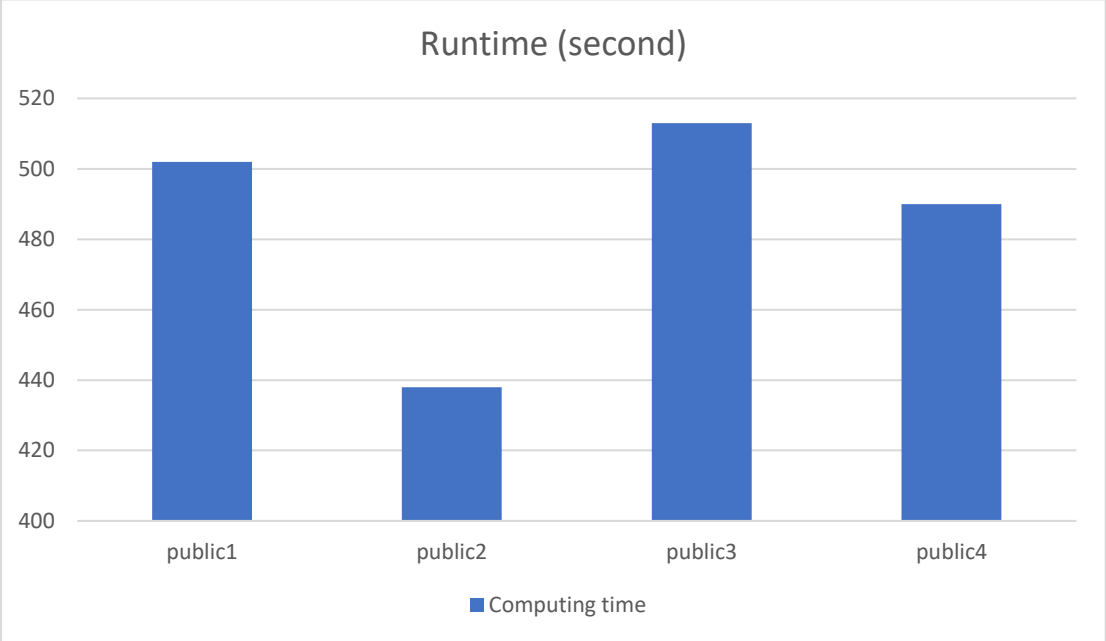
VLSI Physical Design HW2 Report　　111062678 卓榮祥

➢ Compile and execute

```
--How to Compile
  In "HW3/src/", enter the following command:
  $ make
  An executable file "hw3" will be generated in "HW3/bin/"

  If you want to remove it, please enter the following command:
  $ make clean

--How to Run
  Usage:
  $ ./hw3 <txt file> <out file>

  E.g., in "HW2/bin/", enter the following command:
  $ ./hw2 ../testcase/public1.txt ../output/public1.floorplan
```

➢ The wirelength and the runtime of each testcase.



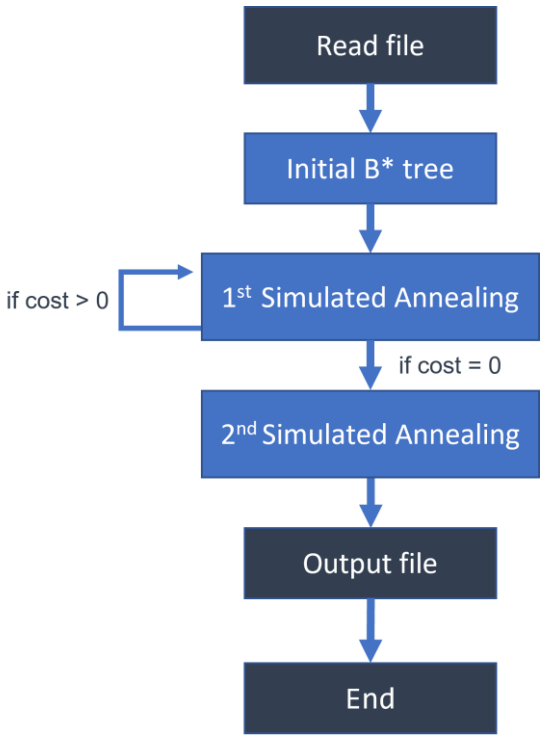
	Public1	Public2	Public3	Public4
Wirelength	157999734	21975532	2089593	100457325
Runtime(s)	502.78	438.54	513.98	490.46

- **Determine the shapes of the soft modules & Benefits of the approach**

```
for (int h = 1; h <= s.min_area; ++h) {

    int w = ceil(double(s.min_area)/double(h));
    if (h >= 0.5 * w && h <= 2 * w ) {

        (s.possible_width).push_back(w);
        (s.possible_height).push_back(h);

    }

}
```

附圖的程式片段是我決定 soft module shape 的方法，高度由 1 開始考慮，並試著找出可能的寬度，若寬高比例符合 spec 要求則將它存進 vector 中，因此此 vector 就收集了 soft module 各種不同的寬高比例，此作法目的是希望在做 simulated annealing 時增加 soft module 擺放的可能性來將 module 塞入 fixed-outline 中且優化 HPWL。

- **The details of the floorplanning algorithm.**



上圖為演算法的流程圖，大致分幾個步驟：

- **Read file**

  讀取 input file 並且將其提供的資訊如 chip size、number of soft module、soft module 種類等存入變數或資料結構中。

- **Initial B*tree**

  初始化 B* tree，B* tree 中每個 node 對應到一個 soft module，如此一來便能夠透過 B* tree 的特性來擺放 soft module。

- **1st Simulated Annealing**

  在這階段中做 Simulated Annealing 主要目的是希望能夠將所有 soft module 放進 outline 中且 module 之間不重疊，因此在這邊 cost 的計算方式為:

```
return (max_x_coord - chip_width) + (max_y_coord  - chip_height) + overlap_area;
```

  max_x_coord：所有 module 中最大的 x 座標

  max_x_coord：所有 module 中最大的 y 座標

  chip_width：spec 中 chip 的寬度

  chip_height：spec 中 chip 的高度

  在 simulated annealing 過程中，會不斷進行 perturb，而我實做了三種 perturb 方式(改變 module 的長寬、移動 B* tree 中的 node、交換 B* tree 中的兩個 node)，而每次 perturb 過後會試著將 module 排入 fix-outline 中，排完後計算一次 cost。在實作中我讓 Simulated Annealing 演算法重複做下去直到產生出的 cost 等於 0 為止，目的是希望能夠想辦法先將 soft module 合法塞入 fixed-outline 中，以此為前提，接著再優化 HPWL。

- **2nd Simulated Annealing**

  這次的 Simulated Annealing，和 1st Simulated Annealing 的方法很像，差在 cost 的評估方式多加入了 HPWL 的值，以降低 cost 做為目

標，目的是希望能夠對 wirelength 做優化(在 soft module 擺放合法的前提下)。

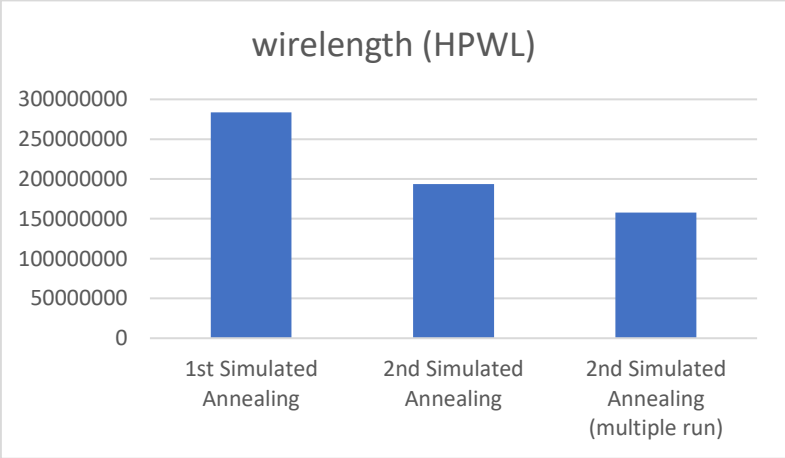
- **Output file:**

  將 wirelength、soft module 座標、寬高等資訊寫入 floorplan file。

➢ Enhance the solution quality

```
while(elapsedTime < maxRunTime){
    simulatedAnnealing(soft_modules, fix_modules, 10000000, 0.1, 0.25, 10000, 1, false);  //to satisfy constraint & minimize hpwl
    for(auto& s: soft_modules) soft_hash[s.name] = &s;
    cout << "wirelength outside:" << get_total_wirelength() << endl;
    currentTime = std::chrono::high_resolution_clock::now();
    elapsedTime = std::chrono::duration_cast<std::chrono::seconds>(currentTime - startTime);
}
```

因為時限為 10 分鐘，因此在 2nd Simulated Annealing 中，我會讓他重複地進行好幾 round，直到程式執行時間大於指定時間為止，目的是希望可以再次重複地進行優化，以 public1 這個 test case 為例，從下面圖表可以知道，多做幾次 simulated annealing 是有機會更加優化 HPWL。



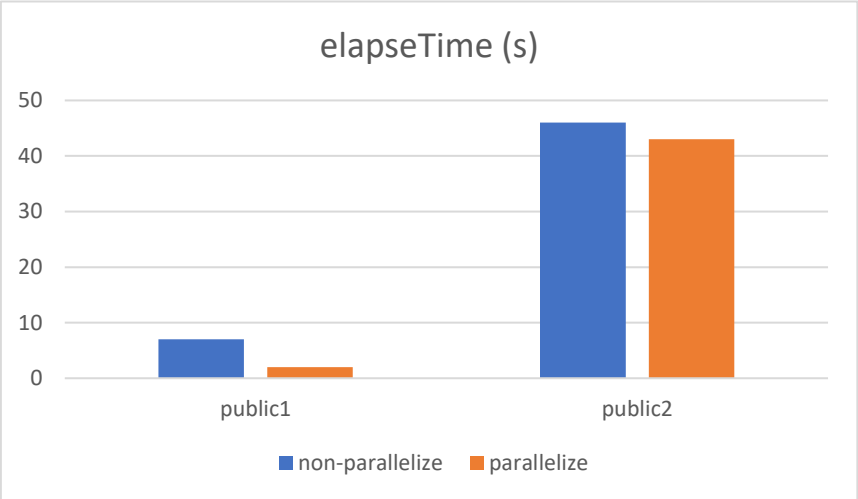
➢ Implement parallelization

在 SA 進行過程中，在計算 cost 時會納入 module 和 module 間重疊的面積，因為每次 iteration 中，module 擺放的方式、位置不見得相同，因此每次 iteration 都須計算重疊的面積，這樣一來會增加不少的 overhead。因此我利用 OpenMP 將這些計算進行平行化：

```
#pragma omp parallel for
for(int i = 0; i < preorder_list.size(); i++){
    for(int j = i + 1; j < preorder_list.size(); j++){

        overlapX1 = max(preorder_list[i] -> x_coord, preorder_list[j] -> x_coord);
        overlapY1 = max(preorder_list[i] -> y_coord, preorder_list[j] -> y_coord);
        overlapX2 = min(preorder_list[i] -> x_coord + preorder_list[i] -> width, preorder_list[j] -> x_coord + preorder_list[j] -> width);
        overlapY2 = min(preorder_list[i] -> y_coord + preorder_list[i] -> height, preorder_list[j] -> y_coord + preorder_list[j] -> height);

        if (overlapX1 < overlapX2 && overlapY1 < overlapY2) {

            overlapSideX = overlapX2 - overlapX1;
            overlapSideY = overlapY2 - overlapY1;
            overlap_area += overlapSideX * overlapSideY;

        }
    }

}
```

```
#pragma omp parallel for
for(int i = 0; i < fix_modules.size(); i++){
    for(int j = 0; j < preorder_list.size(); j++){

        overlapX1 = max(fix_modules[i] -> x_coord, preorder_list[j] -> x_coord);
        overlapY1 = max(fix_modules[i] -> y_coord, preorder_list[j] -> y_coord);
        overlapX2 = min(fix_modules[i] -> x_coord + fix_modules[i] -> width, preorder_list[j] -> x_coord + preorder_list[j] -> width);
        overlapY2 = min(fix_modules[i] -> y_coord + fix_modules[i] -> height, preorder_list[j] -> y_coord + preorder_list[j] -> height);

        if (overlapX1 < overlapX2 && overlapY1 < overlapY2) {

            overlapSideX = overlapX2 - overlapX1;
            overlapSideY = overlapY2 - overlapY1;
            overlap_area += overlapSideX * overlapSideY;

        }
    }

}
return overlap_area;
```

而下圖顯示了未平行化和平行化在 **1st Simulated Annealing** 的執行時間，以 public1、public2 為例（public3、public4 執行時間太短無法看出時間上的差異)。

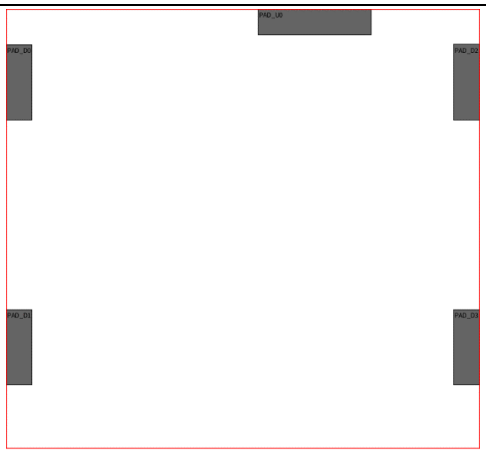
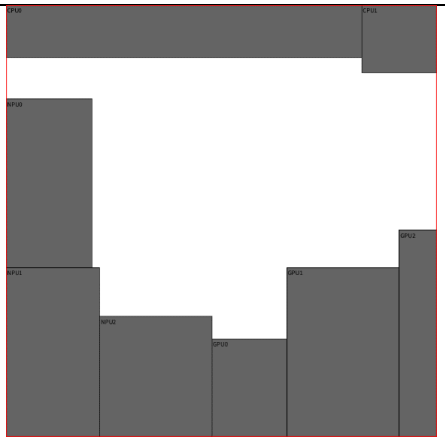
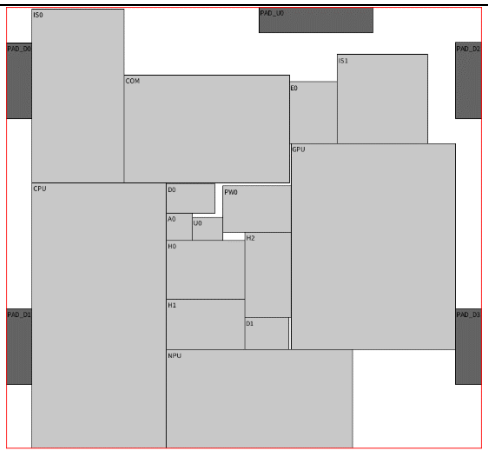


➢ Learned from this homework

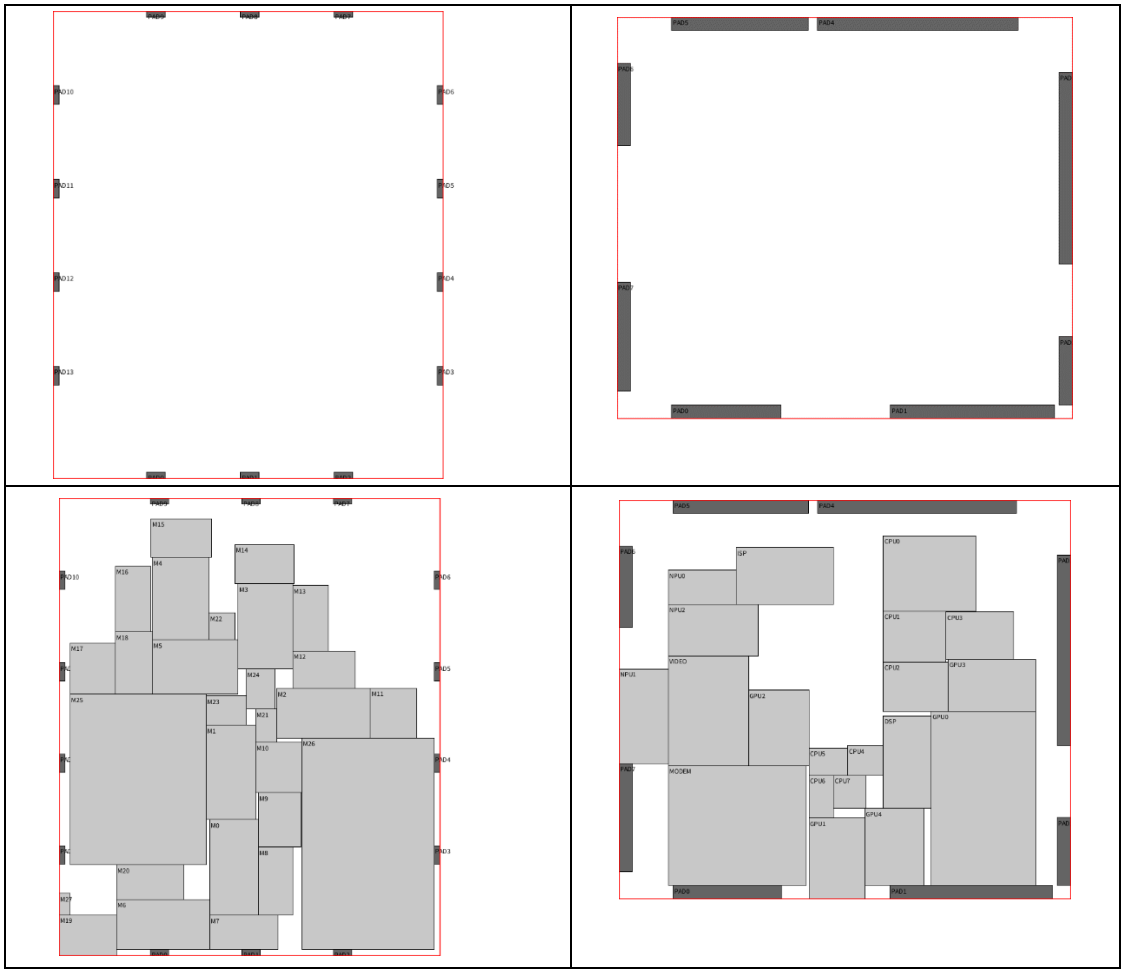
在這次作業中，我覺得挑選一個合適的資料結構非常重要，資料結構影響了執行速度、實作的困難度等，在實做的過程中，我光是想辦法把 soft module 塞進 fixed-outline 中就花了不少時間，但在公布作業成績時 public2

和全部的 hidden case 都沒過，實在是嗤不下這口氣! 因此打算好好完成這次的 makeup，而目標主要是集中精力在構思如何把 module 成功塞進 public 2 這個 case 中，有多的時間再嘗試著利用 simulated annealing 來優化 HPWL。最後成果除了成功將 module 擺入 fixed-outline 中，優化過後的 HPWL 結果都很不錯，甚至在 public1 這個 testcase 中勝過其他同學的結果。

➢ Experiment result

Public1	Public2
	
	

Public3	Public4


PAD10
PAD6
PAD11
PAD5
PAD12
PAD4
PAD13
PAD3
PAD5
PAD4
PAD0
PAD1
M15
M14
M16
M6
M3
M13
M22
M18
M5
M17
M12
M24
M2
M11
M25
M23
M21
M1
M10
M26
M9
M0
M8
M20
M27
M6
M19
M7
ISP
CPU0
NPU0
NPU2
CPU5
CPU3
CPU2
GPU3
NPU1
VIDEO
GPU2
DSP
GPU0
CPU5
CPU4
MODEM
CPU6
CPU7
GPU4
GPU1
PAD8
PAD1