# CS6135 VLSI Physical Design Automation

# Homework 2: Two-way Min-cut Partitioning

**112062682 張宇越**

1. How to compile and execute my program :
   - **Compile :**

     Enter HW2/src/ and make, it'll generate the executable file to HW2/bin/

     $ cd HW2/src/

     $ make

   - **Execute :**

     $ cd HW2/bin/

     $ ./hw2 <testcase file> <output file>

     E.g.

     $ ./hw2 ../testcase/public1.txt ../output/public1.out

2. The screenshot of the result of running the HW2_grading.sh :

```
[g112062682@ic22 ~/HW2_grading]$ ./HW2_grading.sh
+-------------------------------------------------+
|                                                 |
|    This script is used for PDA HW2 grading.     |
|                                                 |
+-------------------------------------------------+
host name: ic22
compiler version: g++ (GCC) 7.3.0

grading on 112062682:
 checking item          | status
------------------------|--------
 correct tar.gz         | yes
 correct file structure | yes
 have README            | yes
 have Makefile          | yes
 correct make clean     | yes
 correct make           | yes

  testcase |   cut size |    runtime | status
-----------|------------|------------|--------
   public1 |       1833 |       0.83 | success
   public2 |        168 |       0.85 | success
   public3 |      33310 |       1.29 | success
   public4 |     139923 |       2.89 | success
   public5 |     240339 |      19.87 | success
   public6 |     186880 |     149.23 | success
+-------------------------------------------------+
|                                                 |
|   Successfully write grades to HW2_grade.csv    |
|                                                 |
+-------------------------------------------------+
```

3. The details of my algorithm :

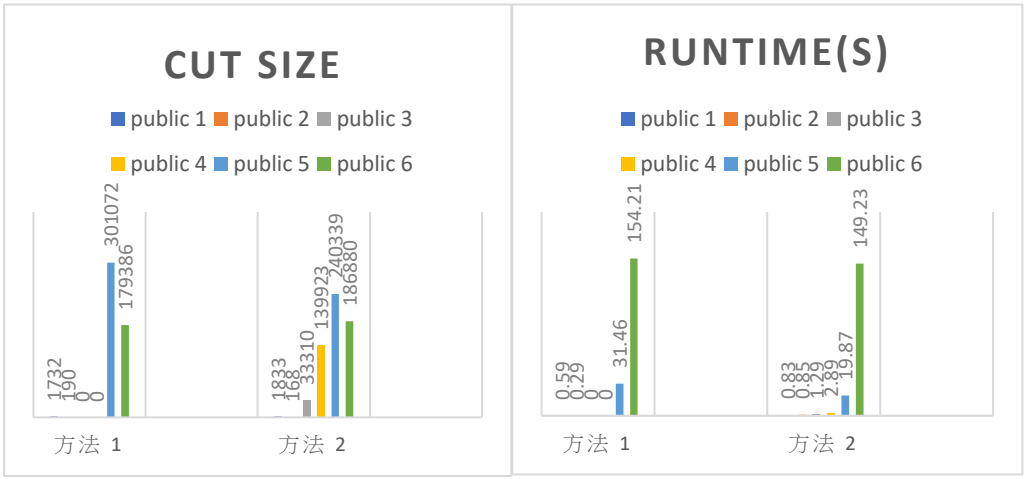
```
main(inFile, outFile):
    # === 1. 讀取並解析輸入檔 ===
    # parseInput 會依序讀取 Tech/LibCell、DieA/DieB、Cells、Nets
    parseInput(inFile, techLibCells, cellMap, dieA, dieB, nets)

    # === 2. 計算每顆 Cell 在 DieA/DieB 的面積 ===
    # 利用 cellMap 中每個 Cell 的 libCellName，
    # 從 techLibCells 中找到對應的 (width,height)，算出 areaA/areaB
    computeCellAreas(cellMap, techLibCells, dieA, dieB)

    # === 3. 計算 Pmax ===
    # Pmax 用來決定 Bucket 陣列的長度 (2*Pmax+1)，
    # 取所有 cell 連接之 net 的權重加總中之最大值
    Pmax = 0
    for each cell in cellMap:
        sumWeight = 0
        for each netName in cell.nets:
            netID = parseNetID(netName) # 例如 netName="N10" -> netID=9, 減一, 因為陣列從 0 開始
            sumWeight += nets[netID].weight
        Pmax = max(Pmax, sumWeight)

    # === 4. 準備 buckets (A, B) 結構 ===
    # 每個 bucket 內都有:
    #     - bucketList: 用 (gain + Pmax) 當索引的鏈結串列
    #     - bucketSet : 紀錄目前在該 partition 的 cell (用於最後輸出)
    #     - cnt       : 此 partition 中的 cell 數量
    #     - size      : 此 partition 目前的總面積
    buckets = [ Bucket("A", Pmax), Bucket("B", Pmax) ]

    iteration = 0
    initTime = currentTime()

    # 這兩個變數只是用來控制是否繼續迭代
    maxPartialSum = 0
    totalTimeUsed = 0
```

```
# === 5. 進入 FM 演算法的迭代迴圈 ===
while ( maxPartialSum > 0 or iteration==0 ) and ( totalTimeUsed < 180 ):
    iteration += 1
    partialSum = 0          # 用於記錄本 iteration 內所有 move 的 gain 總和
    maxPartialSum = 0       # 用於判斷此輪最佳是否 gain 總和 >  0

    # --- (a) 依迭代次數初始化 partition 狀態 ---
    if iteration == 1:
        # 第一次: 先將所有 cell 放到 DieA，若超出 DieA 最大可用面積，
        # 再依 cell 的 (areaA - areaB)大小作排序, 將部分 cell 搬到 DieB
        initSolution(cellMap, dieA, dieB, buckets)
    else:
        # 後續 iteration: 先把 partition 狀態重設回上一輪 iteration 的 buckets
        # 以便重新計算初始 gain
        revertPartitionFromBuckets(cellMap, buckets)

    # --- (b) 初始化 gain 和 buckets，並計算 cutSize ---
    # initGainAndBuckets 會:
    #     1) 根據 cell.partition 更新 Net 的 cntBucket[0/1] (有幾顆 cell 在 A 或 B)
    #     2) 判斷每個 Net 是否 cut/uncut，以及對每顆 cell 計算初始 gain
    #     3) 將 cell 放到對應的 BucketList，並更新 bucketSet
    #     4) 回傳當前分割的 cutSize (根據 nets 中被切割的 net 權重總和)
    cutSize = 0
    initGainAndBuckets(nets, cellMap, buckets, cutSize, Pmax)

    printIterationInfo(iteration, cutSize, buckets) # 印出初始狀態

    # pass: 代表在一次 iteration 內，連續移動 cell 的回合數
    pass = 0
    bestPass = 0
    minCutSize = cutSize
    moveList = []  # 用來記錄本次 iteration , 不同 pass 的移動 (cellName, fromPartition)

    # --- (c) 通過 cellSelect 選出可移動的 cell，不斷移動直到無可動 cell ---
    target = cellSelect(buckets, dieA.maxUsableArea(), dieB.maxUsableArea())
    while target != NULL:
        pass += 1
        partialSum += target.gain    # 累計此步的 gain
```

```
        fromSide = target.partition # cell 移動前在哪一側

        # 記錄移動行為: (cellName, fromSide)
        moveList.push( (target.name, fromSide) )

        # 更新分割(partition)及 cutSize
        #       1) target.partition = !fromSide
        #       2) cutSize -= target.gain
        #       3) 更新 buckets[fromSide] / buckets[!fromSide] 的 cnt, size
        #       4) lock 該 cell
        updatePartition(target, buckets, fromSide, cutSize)

        # 若移動後 cutSize 有進步，更新此次 iteration 的最佳記錄
        if cutSize < minCutSize:
            minCutSize = cutSize
            maxPartialSum = partialSum
            bestPass = pass

        # 更新此 cell 連接的每條 net 的 gain
        #     - 減少 net.cntBucket[fromSide], 增加 net.cntBucket[!fromSide]
        #     - 依 net 是否變成 critical net 或由 critical net 消失,
        #         調整其他 cell 的 gain
        for each netName in target.nets:
            netID = parseNetID(netName)
            updateGain(target, nets[netID], cellMap, buckets, fromSide, Pmax)

        # 選下一顆可移動的 cell
        target = cellSelect(buckets, dieA.maxUsableArea(), dieB.maxUsableArea())

    # --- (d) 回溯到本 iteration 裡「最佳移動步數 (bestPass)」的狀態 ---
    # 將 moveList 中超過 bestPass 的移動撤銷 (恢復 partition)，
    # 同時更新 buckets[0]/buckets[1] 的 bucketSet
    updateBucketSet(moveList, buckets, bestPass)

    # --- (e) 重新計算回溯後的 cutSize ---
    # 對所有 net 重新統計 cntBucket, 若 net 兩端都有 cell 則為 cut，累加權重
    finalCutSize = recalcAfterBestPass(nets, cellMap, buckets)
    minCutSize = finalCutSize
```

```
        #  印出此 iteration 的最佳結果
        printBestInfo(bestPass, maxPartialSum, minCutSize, buckets)

        # (更新 totalTimeUsed 以檢查是否超過 180 秒等條件)
        totalTimeUsed = currentTime() - initTime

# === 6. 輸出結果 ===
# 最後一次迭代結束後，buckets 保存了目前(或最後一次)最好的 partition 狀態
# writeOutput 會寫:
#       CutSize <minCutSize>
#       DieA <#cellsInA> + cells 清單
#       DieB <#cellsInB> + cells 清單
writeOutput(outFile, minCutSize, buckets)
print("Write output file done.")

# === 7. 釋放記憶體 (delete cell pointers) ===
cleanupCellMap(cellMap)

end
```

4. The techniques I used to improve my solution’ s quality :




主要是針對 initSolution 做改善。

剛開始先將所有的 cell 先放到 Die A。

- 方法一 ：

  將所有的 cell 做排序，優先將在 Die A 面積大的 cell，放到 Die B。

  在 public 3、public 4 ，Die A 會違反 utilization。

- 方法二：

  將所有的 cell 做排序，依照 cell 的 (area A – area B)大小作排序, 將 cell 從 Die A 搬到 Die B。每一個測資都能通過。

5. What have you learned from this homework ? What problem(s) have you encountered in this homework ?

   - 在這次作業中，我學會了 FM algorithm 的實作流程，包括如何計算初始 gain、如何利用 buckets 來選擇需要移動的 cell，並在每次移動後動態更新整個系統的 gain。

   - 在整個過程中，initSolution 函式的設計與實作對我來說是最棘手的部分。這個函式的目標，是要先把所有的 Cell 放在 Die A，如果 Die A 的使用面積超過限制，就必須依照一定的準則，將部分 Cell 移動到 Die B；同時還要確保 Die B 放下這些 Cell 後也不會超出自己的可用面積。