CS6135 VLSI Physical Design Automation

Homework 3: Fixed-outline Slicing Floorplan Design

Due: 23:59, November 19, 2019

---

1. Introduction

In this homework, you are asked to implement an existing algorithm, published in the DAC-86 paper "**A New Algorithm for Floorplan Design**" by Wong and Liu, to solve the fixed-outline floorplan design problem with a set of hard blocks.

2. Problem Description

(1) Input:

- A set B of hard blocks, where each block b_i in B has a rectangular shape specified by 4 corners.
- A netlist E
- The dead space ratio, which is predefined and passed by the argument. The aspect ratio of the floorplan region is 1, so you can calculate the width w_{fl} and height h_{fl} of the floorplan region as follows:

$$w_{fl} = h_{fl} = \sqrt{(total\ block\ area) * \big(1 + (dead\ space\ ratio)\big)}$$

For example, if the total block area is 1200000 and the dead space ratio is 0.2, the width w_{fl} and height h_{fl} of the floorplanning region are as follows:

$$w_{fl} = h_{fl} = \sqrt{1200000 * 1.2} = 1200$$

Then, the coordinates of the lower-left corner and upper-right corner of the floorplan region are (0, 0) and $(w_{fl},\ h_{fl})$, respectively.

(2) Output:

- The total wirelength of all nets, where the wirelength for each net is defined as the half-perimeter wirelength (HPWL) of the minimum bounding box of pins of the net. Each pin of block b_i is located at the center of b_i. Note that the x- or y- coordinate, say i, of each block center is rounded down to an integer k such that $k \leq i < k + 1$.
- The coordinates $(x_i,\ y_i)$ of the lower-left corner of each block b_i, as well as the rotation status (1 for rotated, and 0 for un-rotated).

(3) Objective:

By assuming each block can be rotated by 90 degrees, the total wirelength of the floorplanning result and the runtime of you program are minimized subject to the following constraints.

1. Fixed-outline constraint: Each block must be entirely inside the floorplan region.
2. Non-overlapping constraint: No two blocks overlap with each other.

3. Input File

(1) The *.hardblocks* file:

The .hardblocks file specifies the name and the other information about each block/terminal node in the floorplan. Each line specifies a single block/terminal node. Here is an example:

```
NumHardRectilinearBlocks : 10
//NumHardRectilinearBlocks : number of hard rectilinear block nodes
NumTerminals : 69
//NumTerminals : number of terminal (pad etc.) nodes
sb0 hardrectilinear 4 (0, 0) (0, 82) (199, 82) (199, 0)
//nodeName hardrectilinear vertexNumber vertex1, vertex2, ..., vertexN
      ⋮
p1 terminal
//nodeName terminal
      ⋮
```

- *nodeName* is an arbitrary-length alpha-numeric string, and is case-sensitive.
- *hardrectilinear* is a literal which declares that the node is a hard rectilinear block.
- *vertexNumber* is the number of vertices of the corresponding hard rectilinear block.
- *vertex1, vertex2, ..., vertexN* are a list of all vertices of the corresponding hard rectilinear block in a clockwise order, *vertex1* != *vertexN*. Each vertex is a pair of parentheses-enclosed and comma-separated intergers indicating the X-, then the Y- coordinate of the vertex, relative to the lower-left corner of the corresponding hard rectilinear block's bounding box.
- *terminal* is a literal which indicates that the node is a fixed pin.

(2) The *.nets* file:

The .nets file specifies the netlist. Here is an example:

```
NumNets : 118
//NumNets : number of nets
NumPins : 248
//NumPins : number of pins
NetDegree : 2
//NetDegree : number of pins on the net
p1
sb6
//nodeName
      ⋮
```

(3) The *.pl* file:

The .pl file specifies the pin coordinates of each terminal node in the floorplan. Here is an example:

```
p1 0 0
//nodeName XY-coordinate
      ⋮
```

4. Output File

(1) The *.floorplan* file:

The .floorplan file specifies the floorplanning result including the total wirelength of all nets and the coordinates of the lower-left corner of each block with/without rotation.

```
Wirelength 75563
Blocks
sb0 152 284 1
//nodeName lower-left corner coordinates (x,y) Rotated
sb1 126 179 0
//nodeName lower-left corner coordinates (x,y) Unrotated
      ⋮
```

5. Language/Platform

(1) Language: C/C++

(2) Platform: Unix/Linux

6. Report

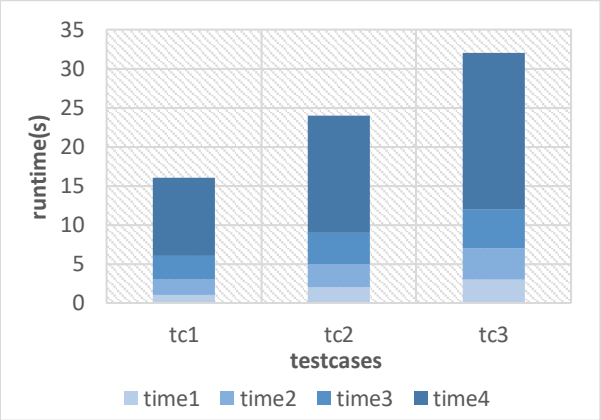
Your report must contain the following contents, and you can add more as you wish.

(1) Your name and student ID

(2) How to compile and execute your program and give an execution example.

(3) The wirelength and the runtime of each testcase with the dead space ratios 0.15 and 0.2, respectively.

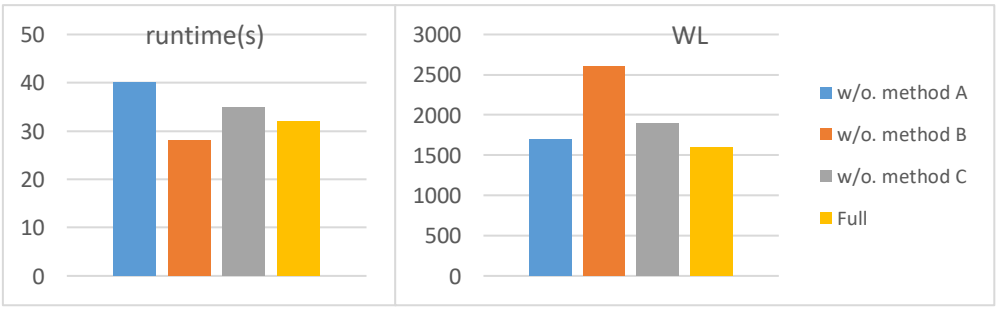
Notice that the runtime contains I/O, constructing data structures, initial floorplanning, computing (perturbation) parts, etc. The more details your experiments have, the more clearly you will know where the runtime bottlenecks are. You can plot your results like the one shown below.



(4) Please show that how small the dead space ratio could be for your program to produce a legal result in 20 minutes.

(5) The details of your implementation. If there is anything different between your implementation and the algorithm in the DAC-86 paper, please reveal the difference(s) and explain the reasons.

(6) What tricks did you do to speed up your program or to enhance your solution quality? Also plot the effects of those different settings like the ones shown below.



(7) Please compare your results with the top 5 students' results last year for the case where the dead space ratio is set to 0.15, and show your advantage either in runtime or in solution quality. Are your results better than theirs?
- ✓ If so, please express your advantages to beat them.
- ✓ If not, it's fine. If your program is too slow, then what could be the bottleneck of your program? If your solution quality is inferior, what do you think that you could do to improve the result in the future?

Top 5 students' results last year (white space ratio = 0.15)

	n100		n200		n300	
	RT	WL	RT	WL	RT	WL
1	35.877	209584	72.306	375829	105.001	522036
2	2.142	214282	12.651	395941	32.083	556247
3	21.121	210292	71.424	377553	182.017	545774
4	25.017	218352	100.144	402538	225.46	553960
5	87.674	204470	159.531	378612	220.383	562069

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

7. Required Items

Please compress `HW3/` (using tar) into one with the name `CS6135_HW3_${StudentID}.tar.gz` before uploading it to iLMS.

(1) `src/` contains all your source code, your `Makefile` and `README`.
- ➢ `README` must contain how to compile and execute your program. An example is like the one shown in HW2.

(2) `output/` contains all your outputs of testcases for the TA to verify.

(3) `bin/` contains your executable file.

(4) `CS6135_HW3_${STUDENT_ID}_report.pdf` contains your report.

You can use the following command to compress your directory on a workstation:

```
$ tar -zcvf CS6135_HW3_{StudentID}.tar.gz <directory>
```

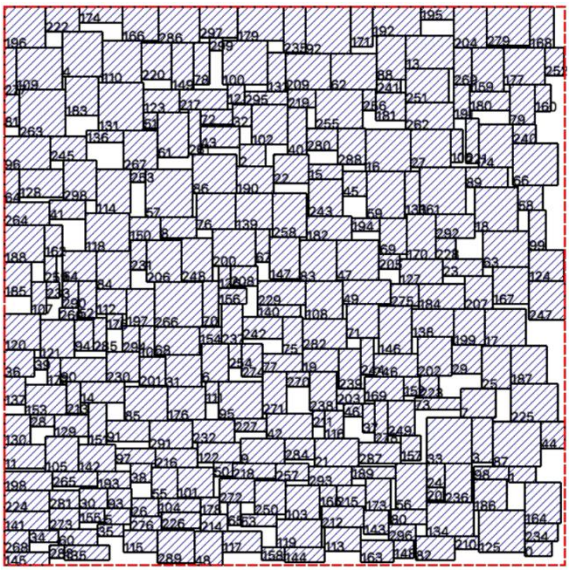
For example:

```
$ tar -zcvf CS6135_HW3_105062901.tar.gz HW3/
```

8. Grading

- ✓ 70 ~ 80%: The solution quality (wirelength) and the runtime of each testcase, hidden testcases included.
- ✓ 20 ~ 30%: The completeness of your report

- ✓ **5% Bonus**: Parallelization. Please specify your system specification.
- ✓ **10% Bonus**: For each testcase, you could use any graphics (programming) libraries/packages/tools such as Xwindow, OpenGL, Qt, Swing, Cairo, Processing etc., to draw your result like the following figure and put each of them in your report.



Notes:

- The executable file name must be named as *hw3*.
- Please use the following command format to run your program.

  `$ hw3 *.hardblocks *.nets *.pl *.floorplan white_space_ratio`

  E.g.: `$ hw3 n100.hardblocks n100.nets n100.pl n100.floorplan 0.1`
- Program must be terminated within 20 minutes for each testcase.
- Grading is based on the total wirelength (primary) and runtime (secondary).