National Taiwan University
Graduate Institute of Electronics Engineering
Department of Electrical Engineering
Physical Design for Nanometer ICs, Spring 2020

May 2, 2020
Handout #12
Yao-Wen Chang

Programming Assignment #3: Global Placement (due 6pm, May 24, 2020 on-line)

Submission URL & Online Resources:

https://cool.ntu.edu.tw/courses/1331/assignments/11216

1. Problem Statement

This programming assignment asks you to write a global placer that can assign cells to desired positions on a chip. Given a set of modules, a set of nets, and a set of pins for each module, the global placer places all modules within a rectangular chip. In the global placement stage, overlaps between modules are allowed; however, modules are expected to be distributed appropriately such that they can easily be legalized (placed without any overlaps) in the later stage. As illustrated in Figure 1, for a global placement result with modules not distributed appropriately (Figure 1(a)), the modules cannot be legalized after legalization and detailed placement (modules in the middle bin of Figure 1(b) are overlapped). In Figure 1(c), a more appropriately distributed global placement result is provided, which can be legalized as illustrated in Figure 1(d).

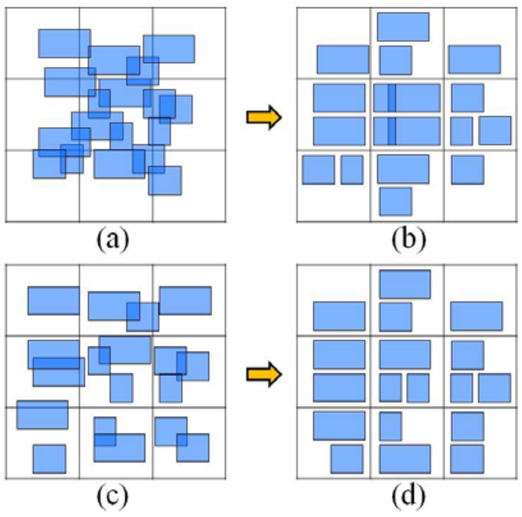


Figure 1. Global placement effects.

In addition to placing modules appropriately, the objective of global placement is to minimize the total net wirelength. The total wirelength *W* of a set of *N* can be computed by

$$W = \sum_{n_i \in N} HPWL(n_i)$$

where n_i denotes a net in N, and $HPWL(n_i)$ denotes the half-parameter wirelength of n_i. Note that a global placement result which cannot be legalized is not acceptable, and any module placed out of the chip boundary would lead to a failed result.

2. Required Data Structure

```
class Module
{
  public:
  /*get functions*/
    string name();
    double x(); // coordinate of the bottom-left corner
    double y();
    double centerX();
    double centerY();
    double width();
    double height();
    double area();
    unsigned numPins();
    Pin& pin(unsigned index); // index: 0 ~ numPins()-1
  /*set functions*/
    // use this function to set the module position
    Void setPosition(double x, double y);
};
class Net
{
  public:
    unsigned numPins();
    Pin& pin(unsigned index); // index: 0 ~ numPins()-1
};
class Pin
{
  public:
    double x();
    double y();
    unsigned moduleId();
    unsigned netId();
};
class Placement
```

```
{
  public:
    double boundaryTop();
    double boundaryLeft();
    double boundaryBottom();
    double boundaryRight();
    Module& module(unsigned moduleId);
    Net& net(unsigned netId);
    Pin& pin(unsigned pinId);
    unsigned numModules();
    unsigned numNets();
    unsigned numPins();
    double computeHpwl();// compute total wirelength
    // output global placement result file for later stages
    outputBookshelfFormat(string fileName);
};
```

The above functions are used to access the data required by the global placement stage. You should use the function `setPosition(double x, double y)` to update the coordinates of modules (to move blocks). At the same time, the coordinates of pins on modules will be updated accordingly and automatically. Note that modules cannot be placed out of the chip boundaries, or the program may not be executed normally.

3. Compile & Execution

To compile the program, simply type:

```
make
```

Please use the following command line to execute the program:

```
./place -aux inputFile.aux
```

For example:

```
./place -aux ibm01-cu85.aux
```

4. Language/Platform

- Language: C/C++
- Platform: Linux. **Please develop your programs on servers in the EDA Union Lab**

5. Submission

You need to create a directory named <student id>_pa3/ (e.g. f08943000_pa3/) which must contain the following materials:

- A directory named src/ containing your source codes: only *.h, *.hpp, *.c, *.cpp are allowed in src/, and no directories are allowed in src/;
- A directory named bin/ containing your executable binary named **place**;
- A makefile name makefile or Makefile that produces an executable binary from your source codes by simply typing "make": the binary should be generated under the directory bin/;
- A text readme file named readme.txt describing how to compile and run your program.
- A report named report.pdf on the data structures used in your program and your findings in this programming assignment.

6. Evaluation

This programming assignment will be graded based on the (1) correctness of the program, (2) solution quality, (3) running time (restricted to 2 hours for each case), (4) report.doc and (5) readme.txt. Please check these items before your submission.

If the global placement result can be legalized, the final solution quality is judged by the total HPWL after the detailed placement stage, which will be shown on the screen as follows:

```
Benchmark: ibm01

Global HPWL: 2349680    Time:     13.0 sec (0.2 min)
Legal HPWL: 2531684     Time:      1.0 sec (0.0 min)
Detailed HPWL: 2482319     Time:      0.0 sec (0.0 min)
==================================================
        HPWL: 2482319     Time:     14.0 sec (0.2 min)
```

However, if the global placement result cannot be legalized, the solution quality will be judged by the following cost function:

$$W \cdot (1 + scaled\_overflow\_per\_bin)$$

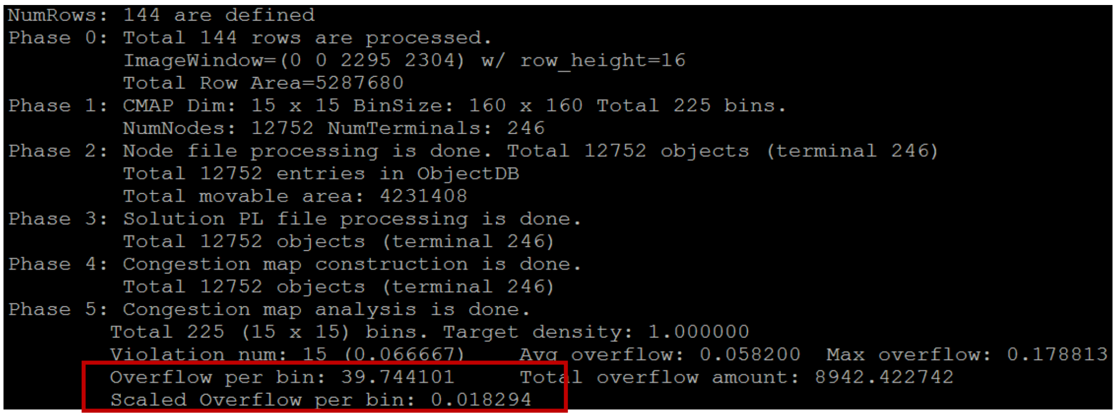
where W is the total wirelength derived from the global placement stage. The "scaled overflow per bin" can be found by using the following script:

```
perl check_density_target.pl <input.nodes> <Solution PL
file> < input.scl >
```

For example:

```
perl check_density_target.pl ibm01.nodes ibm01-cu85.gp.pl
ibm01-cu85.scl
```

Then “scaled overflow per bin” can be checked on the screen as follows:



```
NumRows: 144 are defined
Phase 0: Total 144 rows are processed.
         ImageWindow=(0 0 2295 2304) w/ row_height=16
         Total Row Area=5287680
Phase 1: CMAP Dim: 15 x 15 BinSize: 160 x 160 Total 225 bins.
         NumNodes: 12752 NumTerminals: 246
Phase 2: Node file processing is done. Total 12752 objects (terminal 246)
         Total 12752 entries in ObjectDB
         Total movable area: 4231408
Phase 3: Solution PL file processing is done.
         Total 12752 objects (terminal 246)
Phase 4: Congestion map construction is done.
         Total 12752 objects (terminal 246)
Phase 5: Congestion map analysis is done.
        Total 225 (15 x 15) bins. Target density: 1.000000
        Violation num: 15 (0.066667)     Avg overflow: 0.058200  Max overflow: 0.178813
        Overflow per bin: 39.744101      Total overflow amount: 8942.422742
        Scaled Overflow per bin: 0.018294
```

Note that solutions which can be legalized will get better scores than those that cannot be legalized.

7. Online Resources

Sample codes, benchmarks and checkers can be found at the submission website.

8. Hints

Both combinatorial and analytical methods are acceptable for this PA. If an analytical method is chosen, you may refer to the sample code in GlobalPlacer.cpp and ExampleFunction.cpp, where a conjugate gradient solver is provided.