**Physical Design**

PA #1 - 2-Way F-M Circuit Partitioning

電機三 b03901016 陳昊

1. **Algorithm flow**



* 1. **Initial Partition**

The first step of F-M algorithm is initial partition, while the simplest way to implement is randomly assign each cell in whether part0 or part1, and make sure it’s under the balance criterion.

* 1. **Iteration**

This part is the main process of F-M algorithm, it contains all operations performed in F-M.

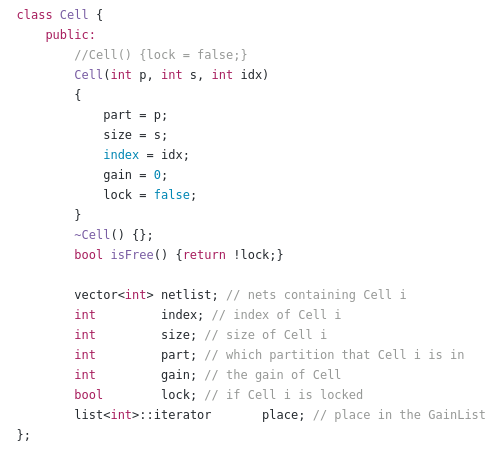
1. Compute the gain of each cell, and pick the highest gain cell which can still satisfy the balance criterion to be the next target.
2. Move the target cell and update the gain of all the cells connected with the target cell. Record the gain of the moved cell and lock it at the same time.
3. After all cells are locked, pick the step which has the highest gain(Gk) and best balance criterion. If Gk <= 0, stop the process and return. Else, set the partition to the step, unlocked all cells, then keep iterating.
4. **Pseodocode**



From the code, we can see that first we have to parse all the input nets and cells, then split them by an initial partition method. After the split operation completed, we’ll build the initial GainList of all cell, which plays the most important role in the program, then we start the iteration.

For each iteration, we’ll try to move the cell with the highest gain and it should satisfy the balance criterion at the same time. After all cells are locked, we pick the best step that we should move to in the iteration, and if the overall gain is greater than 0, we unlocked all cells and keep iterate. If the overall gain is less or equal to 0, we stop the process.

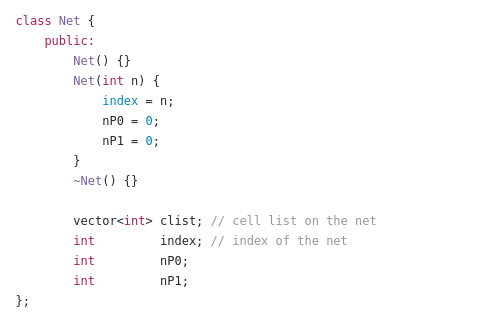
1. **Data structure**
   1. **Cell**



The class “Cell” is for storing the information of each cell, including the index, size, gain, which partition is it in and if it is locked. In addition to these basic information, there’s also an iterator to point to the place in the gainlist so it’s only constant time to access the target cell when finding the maximum gain cell. Besides, each cell has a netlist, which store the index of the nets connecting to it.

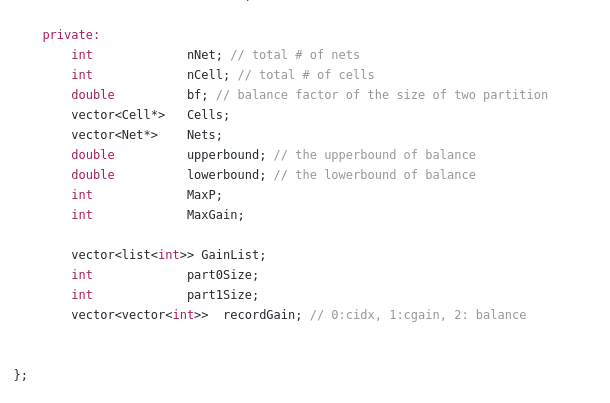
We can see that the index of a cell is store by an integer, because of the optimization of the program, to speed up the runtime. So it cannot deal with cells that their name are not in “cXXX” format.

* 1. **Net**



For each net, there’s an index and a vector storing the indexes of the cells it’s connecting. The special variables nP0 and nP1 are the number of cells from part0 and the number of cells from part1, respectively. By storing these two variables, we do not need to compute if one net is critical when moving a target cell, and it allows us to take constant time while getting these value, efficiently prevent significant growth of runtime when a net is connecting large amount of cells.

* 1. **Manager**



In manager, it contains all pointers of cells and nets in two vectors and in the order of index respectively. Upperbound, lowerbound and bf are used to check the balance criterion. MaxP is the maximum pin number of a cell, and MaxGain is maximum value of gain of all cells in current state, which will be updated every time the operation of updating the gain.

The most important data structure here is the implement of GainList (a vector with size 2 \* MaxP + 1, i.e. gain from -MaxP ~ MaxP), which contains the index of all cells, and is placed by its gain. With this special data structure, we can access the cells in the decreasing order of gain in constant time.

Last, recordGain is actually a vector containing several tuples, which record each move of the target, including the index of the moved target cell, the gain of the move and the balance criterion after the move. So after all cells are locked, we can choose the best step we want in linear time.

With the construction of these data structure, we can obtain an efficient F-M algorithm implementation with speedy runtime.

1. **Special optimization**

First, to place all the cells to the corresponding place to their index. I would read the input file twice, the first time to know the maximum cell index and create a vector of enough size to store all of them. While the second time would place it to its corresponding index in the vector.

Since F-M algorithm is a heuristic algorithm, different initial partition can result in different final cutsize, finding a good way rather than just randomly split the cells is required. One simple way is to put all the cells connecting a lot of nets in the same side, and put those connecting just a few nets in same side. This result in better solution. However, it sometimes gets worse result than randomly split, but still perform well in average.

While F-M algorithm my stuck in the local optimal solution, we can make special effort to prevent it, make it jump out the local optimal and keep going to the global optimal. One simple way is to not move to the best step we pick after an iteration. Then though the total cutsize may be greater after an iteration, it might decrease more after then.