Assignment Report:

A3: Partitioning using Kernighan-Lin / Fiduccia-Matheyses algorithm

EECE 583: CAD Algorithms for Integrated Circuits

Prof. Steven Wilton

M.Eng. Peixu Ren

1. Basics of FM Algorithm

1.1. Data structure

The main data structure is array.
There are four types of classes in FPGA block placement.
(1) Blocks. Every block records (a) its partition mark, 0 for part A and 1 for B, (b) the net number which indicates a net that contains this cell.
(2) Edges. Every edge has three numbers, corresponding to its both ends and which net it belongs to. Also, a flag 'crossing' shows that whether it is across the cut.
(3) Nets. The member data includes (a) the number of this net. This number does not mean anything, but only an index for data storing searching. (b) All the cells in this net, and the number of cells. (c) All the edges in this net. (d) The number of cell in part A and part B. (e) The cut-size of this net. (f) The cost of the net.
(4) The whole placement. Contains all the nets and all the cells (stored as two arrays). And the algorithm actually operates on this level.

1.2. Important values

The calculation method for cells' gain, nets' cutsize and cost is:
(1) Calculation for cell's gain is quite complicated. This is the main concern of my initiative algorithm versions. It will be introduced in section 2.
(2) Assume a net has *n* cells, then its cutsize is cs=1/n;
(3) If a net has more than 1 cell in either part A or B, its cost is 1. If NO cells in part A or B, its cost is 0.

1.3. Basic algorithm

Calculate all the cells' gain. Pick the larges one and move it. The moved cell is immediately locked, which means it will not be moved again (for this pass). Recalculate all the cells' gain and do the similar move repeatedly until all the cells are locked.
One important thing in my algorithm is the arrangement of the array. There are two arrays for part A and B respectively. Two 'pointers' (max_A and max_B) point to the last cells. In short, the 'max' pointers always point to the top of unvisited cells, and the top of unvisited cells is always the 'maximum' cell.

1.4 Basic operations

(1) Find the max cell. Traverse the cell array and find the cell with maximum gain (naïve method).
(2) Move cell. (a) Change the cell's partition mark (A->B or B->A). (b) Decrease 'max' pointer of its 'from' part array. (c) For all the nets that this cell contains, change cell number in part A and B. (d) Change all the edges' 'crossing' status.
(3) Calculate all cells' gain.
(4) Calculate all nets' cost.
(5) Re-initialize (roll back). This includes (a) using the temporary array which is mentioned before to determine whether a cell should be in part A or B; (b) calculate cell number in part A

or B for all nets; (c) set crossing status for all edges; (d) reset the max pointers and find the new top cell.

2. Three versions of FM Algorithm

The method that Fiduccia and Mattheyses proposed for partition improvement is a heuristic algorithm. It gives some ideas about how to optimizing the partition results and how to make the code more efficient.
My algorithm focuses on two aspects: the scheme to calculate cells' gain and reduce complexity of algorithm. Main benchmark for test is 'paira'.
Due to report limitation, version 1 and 2 are introduced briefly.

2.1. Version 1: weighted cutsize

(1) With cutsize=1/n, a cell's gain is calculated as:

$$g = \sum_{i=0}^{n_1} cs_i - \sum_{j=0}^{n_2} cs_j$$

where n_1 is the number of "crossing" edges and n_2 is the number of "uncrossing" edges that connect to this cell.
(2) Move of cells is pairwise. Every time a cell is moved from A to B corresponds to a cell is moved from B to A.
(3) The partition final cost of this algorithm does not converge, with an average of 80.

2.2. Version 2: Unweighted cutsize using nets' cost

(1) Cells' gain is calculated based on the paper of Fiduccia and Mattheyses. A cell's gain is based on its status in one net. It may have five kinds of status. If the cell is the only one in current net, gain increases 1. If its opposite net has no cell, gain minus 1. Otherwise, net makes no contribution to gain. And

$$g = \sum_{i=0}^{n} gain_i$$

where n is the number of nets which a cell is on.
(2) The picking scheme should also consider the balance criterion. If two parts have unequal free cell number, move the max cell of the part with more cells. If two parts have equal free cell number, compare two max cells and pick the bigger one.
(3) The partition final cost of this algorithm converges at around 50.

2.3. Version 3

It is quite understandable that the method of gain calculation will determine the 'motivation' for moving a cell. In version 1, the motivation is to minimize the cutsize of a single cell. This is a wrong optimization goal to some extent while it may quickly lead nets to get fewer cells in one part. In version 2, the motivation is to reduce the net that is crossing partition, which is

definitely our optimization aim. But it may spend long time making random moves until a bulk of nets have their cost contribution.
Hence, I combine these two methods. Cell's gain is determined by weighted cutsize and nets' cost simultaneously. To make the contribution of nets' cost more important, it is always positive. Gain for weighted cutsize (g_1) is the same as version 1. Gain for nets' cost (g_2) is changed to weighted cost as well.

$$g_2 = wc/part\_num$$

where wc is a parameter to determine the weight of nets' cost. Part_num is the cell number of the net's current part that the cell is in. It is clear that when a net's cells get fewer, g_2 is larger. However, this method does not guarantee a convergence. Final cost may be form 2 to 100.

2.4. Other attempts for efficiency

(1) At the beginning, every time when a cell moves, I need to find 'involved' nets. But in the FM paper, it tells a data structure that every cell contains an array for all net number. If the involved nets are found only once, the finding operation with O(n) will be O(1).
(2) When calculate a source cell's gain, if traversing all the involved nets and find whether edges are acrossing, the complexity is $O(n^2)$. However, a source cell's gain can be calculated as

$$\mathrm{g1}+= (\mathrm{parta\_num} - \mathrm{partb\_num}) \times \mathrm{cutsize}$$

for all nets. This leads to complexity of O(n).
(3) I used to find the max cell with a sorted array. But actually, the elements in this array changes continuously. So, I only need to put the max cell on the top of the array. Traversing the array is O(n), which is much faster than binary insertion sort of O(nlog(n)).
In short, (1) leads to a time saving of 380s->279s. (2) results in 279s->3s. And (3) makes final time consuming of 1s.

3. Results

I only choose the minimum of every benchmarks that I observed. The average is:

$$\mathrm{AVE} = \frac{\text{sum of minimum cost}}{\text{number of benchmarks } (11)} = 49.72$$

4. User Instructions

As I use EasyGL for UI drawing, users cannot open a file through graphical window. But users can type the name of the benchmark in command line. The file 'exefile' is the executable program. Use command "make" to make execution file. Library and header link need to be modified for Linux and Mac.

5. Reference

[1] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," Bell System Technical Journal, Vol. 49, Feb. 1970, pp. 291-307.
[2] C.M. Fiduccia and R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", Design Automation, 1982. 19th Conference.