Implementation of Fidducia – Mattheyses Algorithm for Circuit Partitioning

By Tannu Sharma

Abstract: In this paper we will discuss the circuit-partitioning problem. The goal is to minimize the cost of all the cut nets while satisfying the partition size constraints up to 45-55% of area range. We will implement this using Fidducia-Mattheyses Algorithm to find an optimal bi-partition of the circuit.

Fidducia Mattheyses (FM) is an extension of *Kernighan-Lin* (KL) algorithm implemented on a graph with hyper-edges. Where hyper-edges are edges of a graph that connect any number of nodes. In KL algorithm, a pair of nodes is swapped to maximize the gain. Whereas, in FM based approach single cell is moved from one partition to another to minimize the cost.

Definition:

We consider a circuit represent as a set of cell connected to different nets through multiple pins. Let P be the number of pins of a unit cell (c), where c = 1, 2, 3…c cells. So, if V is the set of the C cells, then,

$$area(V) = \sum_{c=V} area(c)$$

In this graph *Hyper-edges* are the nets connecting two or more cells. A *cell* (c) is a logical or functional component unit, which will be considered as a node of Graph during the discussion of this paper.

A *net* connects many cells together. A *partition* is a grouped collection of cells. *Bi-partition* means the circuit will be grouped in two parts, 0 and 1.

We begin by assuming that some partition of nodes already exist, but it's not optimal and we have to take the graph to optimal partition

Our algorithm will try to move around the cell from one partition to another.

A *base cell* is the cell selected for movement from one partition to another. Its selection is based on its gain constraint by not affecting the balance criterion (both of which we will explain ahead).

The optimization goal is to reduce the Cut-Edges without compromising on balance.

An edge is considered to be *uncut*, if all its connected cells are occupied within a partition. However, if a net is shared among multiple partitions (in this case two partitions) it is a *cut edge.*

FM heuristic:

FM proposes an iterative technique to partition a multi-pin circuit into blocks such that the number of nets having cells in both the partition is minimized and the balance of each partition is maintained.

1. A single cell is moved from any one of the partitions.
2. Cost of the nets cut by the partition is reduced.
3. Gain is updated for connected cells after every move of move of cell c.
4. After the move, the cell is locked for remainder of the pass.
5. Cells belonging to critical nets are not moved across the partition, as they will change the cut-state.

Cut-state is a state of a cut-edge (or cut net). *Critical net is a net if it has a cell which if moved will change the cut-state.*

Example for a net1 if number of cells in each partition 1 is 0 or 1, or in partition 0 is 0 or 1 then, net 1 is a critical net.

Gain:

The gain $\Delta g(c)$ for cell c is the change in the cut set size if c moves.

$$\Delta g(c) = FS(c) - TE(c)$$

FC(c) = number of cut nets, i.e. number of cut nets connected only to cell c of a partition. It can be considered as a moving force applied on a cell c by the other partition. Higher the value of moving force (FC), more likely is the cell (c) to move.

RU(c) = number of uncut nets connected to cell c within a partition. It can be considered as a retention force of c to remain in the existing partition.

After the cell (c) move, gain for all the cells connected to c will be updated.

In one pass after moving m cells we calculate the total Gain of the pass.
The maximum positive gain Gm of a pass is the cumulative gain for all the moves.

$$Gm = \sum_{i=1}^{m} \Delta gi$$

We then select a sequence that maximizes the total gain. This sequence is the moves that we finalize and start the process again.

Balance Criterion:
FM produces balanced partition with respect to the size of a partition.
The ratio factor (r) is used to prevent the clustering of all the cells in one partition. It is also known as balance factor.
If A is the area of partition 0 and B is the area of partition 1, with unit cells.
Total area (V) of all the unit cells.
So,

$$area(A) + area\ (B) = area(V)$$

Then,

$$r = area(A)/(area(A) + area(B))$$

And a partition is balanced if:

$$r.area(V) - area_{\max}\ \ (V) \le area(A) \le r.area(V) +\ area_{\max}\ \ (V)$$

My Implementation:
I have implemented the algorithm using data structures.

1. I have created a C++ Class to represent a hypergraph G composed of nodes (V) and hyper-edges (E). It contains mapping from node label to node object and edge label to edge object. It contains functions to populate the graph and run the algorithm
2. An Edge Class is defined which holds list of nodes as string label. It contains functions that update and edge, decide if it is cut or critical.
3. A Node Class is defined which holds list of all the edges connected to a node as integer labels. It contains function that calculates the Gain etc.
   I can traverse on every edge using its connected node and on every node using its connected edge. The maps in hyper graph would furnish the required object as I pass the labels
4. When a Node Object is created, I am assigning partition to the Node randomly using a C++ random generator function.
5. Update Edge function is called to check the cut state of for each edge. Cut state and criticality is registered within the Edge object.

For each pass:

6. Calculate Gain of all Nodes in the graph.
7. To calculate gain, Iterate over all edges
   a. If the edge is critical, do nothing
   b. Calculate FS and TE and the gain, for the node
   c. Update the temporary gain with the max gain value. Temp gain will be finalized at the end of this pass
8. Sort all the nodes as per their gain in a sorted bucket list (map<gain, Node>).
9. Get the node which is the max from the sorted list.
10. To check the validity of the move, check the balance condition for the selected node. Repeat 8-9 until a node is found and this is the Base Cell.
11. Move Cell, this is temporary move, we decided to finalize at the end. Lock this cell for this pass. Remove from sorted list. Add it to the change Sequence list.
12. Update all edges connected to this base cell. Update gain for all the nodes connected to the edges.
13. Update Gain is similar to calculate gain but run only over the affected nodes. This affects the sorted list of nodes as gain changes
14. Repeat 8-13 until the sort list is empty.
15. We have a change sequence list, we calculate starting from i=0. We select a k such that 0->k elements such that Total Gain is maximized. Only cells given by the best sequence will be permanently moved to the second partition.
16. Finalize the moves of 0-K nodes from Sequence list. Restore the remaining nodes
17. Unlock all Nodes
18. Update gain for all cells with respect to the base cell
19. For program to terminate I have added a convergence condition, which will check the convergence of the code for 10 consecutive passes. i.e. if number of cuts and other parameters do not change for 10 passes we break out.
20. If after 10 consecutive passes, the gain is same and we are not able to minimize

the cut-state of a partition, we will exit then.

Randomness:
Why I am choosing random assignment:
The random assignment is needed to ensure that convergence of our code and to find the Global optimal solution. Random Start helps in optimization problem as every time we start from a new point and converge toward the solution. We avoid getting stuck in local optimal, in multiple runs. As we find a different solution after each run.
Also balance is assured as we assume uniform distribution and hence for large sample space, the initial balance of design is close to 50 %. However in smaller design like with 9 nodes this may not be true a random assignment might result in 3 and 6, which is unbalanced, however algorithm after several passes will try to bring it back to balance.
I tried with a fixed alternate assignment where every second cell is in the second partition. This proved counterproductive as with 5 nodes we still end up with unbalance.
Example:
On small designs like the problem with 9 nodes, we are trying to achieve the balance of 5, when any alternate assignment will give us 44.44 (4 nodes in one partition out of 9) and 55.66 (5 in the other partition out of 9), violating our balance condition.
Hence random start is a good idea even if it leads to initial imbalance.

Design comparison:
I have compared the results obtained from my code with the obtained with the **shmetis** tool.

The shmetis tool is implemented in different phases, where coarsening is done based on KL algorithm, moving all the nodes in a single partition to maximize the gain.
Later the Shmetis tool does initial partitioning of the design into smaller circuits using FM algorithm. Later the already partitioned design is improved using un-coarsening and refined to achieve the desired number of partition. Since the tool allows partitioning a design into multiple partitions during its initial partitioning phase, we can use the tool to get k-partitions.

In my implementation, I have used FM algorithm to bi-partition a design with balance factor of 5 i.e. in the range of 44-55% is maintained

Inputs for Shmetis tool:
shmetis *HGraphFile Nparts UBfactor*

shmetis is the command

HGraphFile (*.hgr) file is the graph file with the total number of edges and nodes in the first line. The subsequent lines have a list of nodes in each line connecting a net. i.e. line is an edge and the values are the list of nodes in that edge.

Nparts is the number of desired partitions for a circuit.

UBfactor is the balancing criterion for a design.

Input for my implementation:
ts_fm HGraphFile

ts_fm is the command
HGraphFile is the graph file with the list of nodes for a circuit.

Results Obtained:
Q1) ***For problem 1 with Shmetis tool:***

```
HyperGraph Information -------------------------
 Name: s13207p.hgr, #Vtxs:  8772,
#Hedges:  8651, #Parts: 2, UBfactor: 0.05
 Options: HFC, FM, Reconst-False, V-cycles
@ End, No Fixed Vertices
Recursive Partitioning... -------------------------
  Bisecting a hgraph of size [vertices=8772,
hedges=8651, balance=0.50]
    The mincut for this bisection = 54,
(average = 64.8) (balance = 0.47)
 -------------------------------------------------------
  Summary for the 2-way partition:
               Hyperedge Cut:        54
            (minimize)
      Sum of External Degrees:       108
            (minimize)
                 Scaled Cost:  2.82e-06
            (minimize)
```

```
        Absorption:   8630.69
    (maximize)

    Partition Sizes & External Degrees:
        4116[  54]   4656[  54]
```

For problem 1 with my tool:

```
Initial Balance:      0.485522
Number of HyperEdges:  4713

Converged FM is fully tuned!

Max k:         0       max total gain:       0
Balance:      0.466484
Number of cut:        275
Pass: 853
**** Input Parameters ****

The Number of Nodes:    8772
The number of Edges:    8651
***** My Results *****
Number of Minimum Cutset:     275
The number of element in each cut:
Set 0: 4092  Set1:  4680
Balance:      0.466484
******************************************
Time taken: 67.22s
```

Q2)   Contents of *.hgr file:

```
8    9
1    6    5
2    6    5
3    7
4    9
5    8
6    7
7    8
8    9
```

Output with My tool:

```
Initial Balance:      0.777778
Number of HyperEdges:  2

**** Input Parameters ****

The Number of Nodes:    9
The number of Edges:    8
***** My Results *****
Number of Minimum Cutset:     5
The number of element in each cut:
Set 0: 5       Set1:  4
Balance:      0.555556
******************************************
Time taken: 0.00s
```

Output for ckt_17.txt netlist file.

```
Initial Balance:      0.444444
Number of HyperEdges:  4

**** Input Parameters ****

The Number of Nodes:    11
The number of Edges:    6
***** My Results *****
Number of Minimum Cutset:     4
The number of element in each cut:
Set 0: 5       Set1:  4
Balance:      0.555556
******************************************
Time taken: 0.00s
```

Output for ckt_432.txt file.

```
Initial Balance:      0.45935
Number of HyperEdges:  95

Converged FM is fully tuned!

Max k:         0       max total gain:       0
Balance:      0.455285Number of cut:   0
Pass: 30
**** Input Parameters ****

The Number of Nodes:    269
The number of Edges:    174
***** My Results *****
Number of Minimum Cutset:     0
The number of element in each cut:
Set 0: 112     Set1:  134
Balance:      0.455285
******************************************
Time taken: 0.02s
```

References:

1. http://users.ece.utexas.edu/~dpan/EE382V_PDA/notes/sait/chapter2.slides.pdf
2. The C++ Programming Language by Bjarne Stroustrup.
3. http://vlsicad.eecs.umich.edu/KLMH/
4. Circuit Partitioning by Prof. David Pan
5. Data Structures Using C and C++ Tanenbaum.