VLSI Circuit Partitioning
Implementation of Fiduccia-Mattheyses Algorithm

Ritu Agarwal, Shashank Rao and Kewal Raul
Stony Brook University

***Abstract*— Partitioning in VLSI System Design is a process of decomposition of a complex system into smaller subsystems.Each subsystem can be designed independently speeding up the design process.It is utilized to minimize interconnections between the subsystems.This paper presents the implementation of Fiduccia-Mattheyses Algorithm in C++, discussing the performance of the code & its resulting output.**

I. INTRODUCTION

VLSI designs of high complexities can be efficiently handled by the scheme of partitioning them into smaller manageable sizes. Subsequently, each subsystem can be designed independently & simultaneously to speed up the design process [1]. The process of decomposition is called partitioning [1]. Partitioning can be used in a hierarchical manner until each subsystem created has a manageable size. Partitioning finds its application in various fields. Divide & Conquer approach is applied to partition various complex problems into smaller problems.

Partitioning plays a pivotal role in the design of computer systems in general, specifically in VLSI chips. A computer system comprises of several transistors which are partitioned into many smaller modules to facilitate the design process. Each block has terminals located at the periphery that are used to connect the blocks [1]. The connection is specified by a netlist, which is a collection of nets [1]. A net can be defined as set of terminals which should be made electrically equivalent. A VLSI is partitioned at multiple levels in order to reduce its complexity. The partitioning of a system into group of PCBs is called system level partitioning. The portioning of a PCB into chips is called board level portioning while the partitioning of a chip into smaller subcircuits is called chip level partitioning.

The partition problem is expressed in terms of a graph G= (V,E) where V represents a set of vertices while E represents a set of hyperedges. A hyperedge is a connection between vertices of a net. The interconnects between partitions are reduced in accordance to the area constraint.

II. RELATED WORK

A. The Kernighan-Lin Algorithm

The KernighanLin algorithm is a heuristic algorithm for finding partitions of graphs. The algorithm has important applications in the layout of digital circuits and components in VLSI.It is a bisection algorithm. It starts by initially partitioning the graph G = (V, E) into two subsets of equal sizes [1]. Vertex pairs are exchanged across the bisection if the exchange improves the cutsize [1].It is carried out iteratively until no further improvement can be achieved.

B. Fiduccia-Mattheyses

In 1982, C.M. Fiduccia and R.M. Mattheyses introduced their partitioning algorithm in a paper, titled A Linear-Time Heuristic for Improving Network Partitions. [2] This algorithm was an improvement on the Kernighan-Lin algorithm in the following ways:

1) The algorithm allows for movement of a single node across partitions, thus, enabling partitions to be present of different sizes as well as speeding up execution since two nodes do not have to be exchanged to facilitate a smaller cut set size.
2) The algorithm implements cuts on hypergraphs, a concept which was not present in the Kernighan-Lin algorithm.

III. PROPOSED SOLUTION

The paper describes the implementation of F-M algorithm using C++.

The input .netD and .are file are parsed creating a structure consisting of an unique key corresponding to each unique node in the netlist & its corresponding area. For simplicity, all further references of the node are made using the above unique key. An adjacency list is implemented to store the graph structure of the parsed netlist.The pads & their corresponding nets from the ibm .netD files are ignored while partitioning.

The initial partition is performed by randomly dividing the list of nodes into two halfs with equal number of nodes. For computing the cutset size the adjacency list is traversed through all the nets & if any of node of the net is in a separate partition the cutset size is incremented by 1.

Inorder to create bucket lists for both the partitions, the maximum possible gain is determined by traversing through the adjacency list, finding for a node with maximum occurrences in the net list. The number of occurrences of this node is the maximum possible gain.

Bucket lists maps with the range (+ maximum possible gain, - maximum possible gain) are initialized with the gain of each nodes. Gain for each node (i) is calculated as;

- If all the nodes in net lie in the same partition then $T(i) = 1$
- If all the nodes except the i^{th}node lie in the other partition then $F(i) = 1$

$$Gain(i) = F(i) - T(i) \quad (1)$$

Node with the highest gain among both the buckets is selected provided it satisfies the area constraint else another optimal node (node of with the same gain if exists or the next highest gain) is selected. The selected node is shifted from its current partition to the other. After, shifting both the bucket lists are updated with new gain values.

The whole process is repeated until the smallest cutset size achieved.

IV. IMPLEMENTATION ISSUES

One of the main issues in our implementation was that the initial partition was taken randomly. Due to this, we faced challenges during incorporating the area constraints since the area of the initial partitions was highly skewed, with one partition having a high total area and the other having a low total area. One of the solutions we had thought of, was to create initial partitions based on area constraints, i.e., maintaining the ratio of the area of the partitions at 1:1 as much as possible, so that later, constraints of 40% and 60% could be applied to the partitions during partitioning. One other solution that we came up with involved creating the same random skewed partition and then moving the gain improving nodes only from the partition of higher area to the partition of lower area. We were unable to implement both solutions due to time constraints on the project, but we hope to include one of these solutions as soon as possible.

One of the other issues faced during our implementation was that of the hardware on which we were working. Initially when we started working on the program, and on running it, our laptops used to start heating and CPU speed slowed down tremendously leading to poor performance. We had to increase the efficiency of our program so that we could get the program to run a little more smoothly. This efficiency improvement included limiting the number and size of variables used and using more efficient data structures for holding the bucket structures and partitions. One other solution that comes to mind is upgrading the hardware, such as the RAM. However, upgrading hardware for improving computational speed and thus reducing the execution time is an expensive and unnecessary practice.

V. EXPERIMENTAL RESULTS

The ISDP Circuit Benchmark Suite consists of 18 .netD files , accommodating around 12,000 to 210,000 cells. We were able to process only the ibm01.netD file, containing 12506 cells.

The code was first tried on a few sample .netD files, for debugging & ensure the correctness of the execution. Some noticeable reduction in the cut size was observed in case of the ibm01 file.Figure 2 mentions the results of the execution of the ibm01 file.

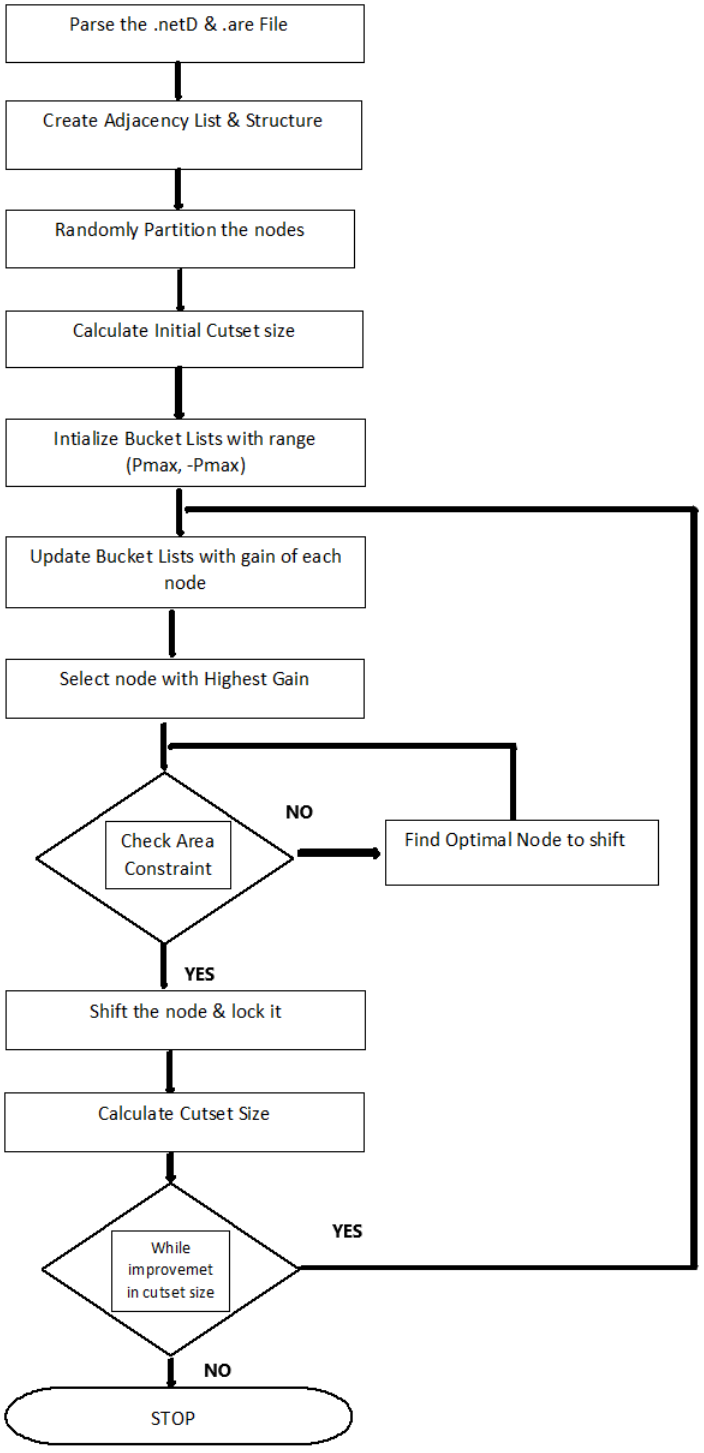

Fig. 1. Flowchart for F-M Implemetation

File Name	Initial Cut Size	Final Cut Size	Execution time (s)
sample.netD	5	1	0.05
ibm01.netD	2636	2056	6400

Fig. 2. Execution Result for .netD Files

VI. CONCLUSIONS

The implementation of Fiduccia-Mattheyses algorithm presented in this paper has been proved to work correctly for small net lists, where manual calculations were possible to verify the correctness of the output. Whether this implementation can correctly calculate the minimum cut size of the ibm.netD files provided by the ISPD98 Circuit Benchmark Suite is currently unknown. Based upon the smaller test cases performed on the program so far, we can conclude that the program should be able to to carry out the F-M algorithm correctly.

We have successfully been able to implement & execute the F-M algorithm; however the time required for processing the ibm01 is on the higher side. Thus, there is a further requirement of optimizing the code inorder to be able to successfully execute the remaining large ibm files.

REFERENCES

[1] N.A Sherwani ”Partitioning,” in Algorithms for VLSI Physical Design Automation, 3rd ed. Norwell, K.A.P., 1999, ch.5

[2] C.M. Fiduccia, R.M. Mattheyses, A Linear-Time Heuristics for Improving Network Partitions, Proceedings of the 19th Design Automation Conference.

[3] C. Alpert. (201, Oct. 14). The ISPD98 Circuit Benchmark Suite.

[4] B.W. Kernighan, S. Lin, ”An Efficient Heuristic Procedure for Partitioning Graphs,” Bell Systems Technical Journal, Vol. 49, No. 2, 1970, pp. 291-307