

Implementation of KL Algorithm for Partitioning using Perl

Swapnil D. Ninawe¹, Pranali D. Surkar²

¹Assistant Professor, E&C Department, PIET, Nagpur, Maharashtra, India

²Assistant Professor, E&C Department, PIET, Nagpur, Maharashtra, India

Abstract: *This paper presents the implementation of KL algorithm for circuit partitioning using PERL. Circuit partitioning is NP hard problem. To solve the circuit partitioning problem, heuristic iterative algorithm is used. The algorithm treats the circuit as a graph, logic gates as nodes and interconnecting nets as edges. It randomly generates the initial partition of the circuit. The text file is used as an input which contains list of nodes and adjacency matrix of edges. The complete algorithm is implemented using arrays and associative arrays in PERL. Associative arrays are used to store the initial partition, improvement values, gain values and final partition. Nodes are stored as keys of associative arrays. The output file is generated which contains final partitions. This process helps to reduce the computation time of this iterative algorithm.*

Keywords: Circuit partitioning, KL Algorithm, two-way partitions, improvement values, gain values, associative arrays and adjacency matrix.

1. Introduction

From last 50 years, Moore's law [1] – [2], at every two year double the number of transistors could be fitted on a single chip of silicon has held true. Due to exponential growth in technology, millions of transistors are now present in an ASIC Design. It is an impractical approach to build such design manually, so automation of design process by using computer-aided design tools is the better solution. Such design tools can evaluate complex design conditions and also tries to find the bottleneck of the problem. Each and every steps in the ASIC Design flow is equally important to get the specific speed, area and power. To deal with such larger design, determining the hierarchy of the design and dividing it into smaller parts plays an important role. The speed of the circuit, its power consumption, its area and its reliability depend severely on the way the circuit is laid out. The complete physical design flow is important; this paper covers the first step in the process called Circuit Partitioning. An efficient partitioning algorithm is important to solve the large and complex design problem. There are different „heuristic“ techniques [3] are available to generate approximate solution to the partitioning problem. In this paper, Kernighan-Lin algorithm is implemented for partitioning. In this paper, the scripting language Perl is used to implement KL algorithm. As Perl does not need a compiler and linker to run the working code; it gives the quick solution to the partitioning problem. Hence this paper gives faster solution to partitioning problem even if it is NP hard.

In this paper, we present the implementation of KL Algorithm using data structures like arrays and hashes. For this, we use Perl. Section II describes the concept of partitioning and its importance in the design flow. Section III explains the KL algorithm for two way partitioning. Section IV describes the data structures used to implement the algorithm. There are some future scope like k-way partitioning and weighted edges partitioning is possible using KL algorithm has been discussed in Section IV.

2. Circuit Partitioning

The size and complexity of the ASIC designs increasing rapidly as millions of transistors in a single chip increasing each years, efficient partitioning techniques are better solution to solve these problems. Circuit partitioning divides the given circuit into two or more parts; such that the weights associated with the signal interconnects is minimum while maintaining a given balance criterion among the part sizes [4]. It is normally composed as the graph partitioning problem. These circuits can be properly modeled by hypergraphs. The partitioning problem is an NP-hard problem; this means use of polynomial-time algorithm for solving such problem is not preferable. Heuristic techniques has been used to generate approximate solutions for hypergraphs partitioning problem.

In the ASIC design flow, partitioning is the first step. Hence it means it is most important step as rest of ASIC Design flow steps directly dependent on the circuit partitioning. If circuit partitioning is not proper; then design may have less area but it will definitely have too much interconnection wire. As technology is continuously shrinking; interconnection delays is major contributing element in total delays. If we try to minimize the interconnect delays then it may have uneven partitions in size. To avoid such kind of undesirable partitioning results; various partitioning algorithms have been created. Generally, it can be expected to get good partitions results; which having minimum wire delays and maintaining area constraints with all partitioning algorithms. But time is an important factor in deciding the specific partitioning algorithm. Basically, the main task of circuit partitioning algorithm is to minimize the number of cuts and to minimize the deviation of nodes (which can be inputs, outputs, logic gates etc.) assigned to each partitions. There are various heuristic techniques available to solve the circuit partitioning problem. These can be deterministic or stochastic algorithms based on decision making solutions. Deterministic decisions have been made to get the solution using deterministic algorithm while random decisions have

been made in the search of solution in stochastic algorithm. Even heuristic algorithm can also be constructive or Iterative. In constructive heuristic; a seed component is used initially and other components are added to obtain the complete solution in the partial solution. In iterative heuristic; two inputs including problem instance description and its initial solution has been received. Then it tries to modify the current solution by improving the cost function. If the new cost is better than old one; then new one is used otherwise the solution may or may not be accepted, it depends on exact search method used in algorithm.

3. KL Algorithm

Generally, partitioning problem consider the circuit as a graph. The graph bisection problem is NP-hard, and to get good partitioning results of the problem, creation of effective heuristic algorithms must be required [5]. In this paper Kernighan-Lin algorithm is applied to the circuit partitioning problem. This algorithm first convert the circuit into its corresponding graph by treating logic gates as vertices and their interconnection is replaced by edges between the corresponding vertices of the graph. The K-L heuristic algorithm is an iterative improvement procedure. First initial partition is carried out and then it tries to improve the bisection further. To find a minimal-cost partition of a given graph of $2n$ vertices (of equal size) into two subsets of n vertices each [6]. This section provides the solution of the two-way partitioning problem.

KL algorithm randomly generates the initial partition and two sub-circuits P1 and P2 is created. If the given graph has $2n$ vertices (circuit has $2n$ gates), the first n vertices are taken into subset P1, and the remaining n vertices are taken into subset P2. This paper deals with only even number of vertices. Then only it can be divided into same number of vertices for each partition P1 and P2. If the circuit has odd number of vertices then we have to twist the algorithm by assuming one extra dummy node. Here the objective of this section is to minimize the number of edges which are cut by the partition. Suppose there is an edge exists between two nodes A and B in the graph. It means A and B are some logic gates and there is an interconnection between them. Now, the first thing to check that these two nodes are in same sub-circuits or in different sub-circuits. If nodes A and B are in same sub-circuits then that edge between them is cut; otherwise it can be considered as a cut. To get the better and improved new solutions from the current one; subset of nodes from sub-circuit P1 must be swap with subset of nodes from sub-circuit P2. These two subsets must be of same size otherwise it will create an imbalanced two-way partitioning.

KL algorithm basically concentrates on selecting vertices at a time; but it does not swap one pair of nodes at a time. This algorithm temporarily swap a single pair of nodes from different sub-circuits at a time; but the main motive of these swapping is to get a subset of nodes to be swapped on a permanent basis. First step is to find the cut and uncut edges for each node in the graph. For each node say A in the circuit; we have to find out two cost values, E_A and I_A . E_A is the set of A's incident edges that are cut by the cut line, and I_A is the set of A's incident edges that are uncut by the cut

line. We can say that, the total number of edges attached to A is the sum of E_A and I_A ; which is also called as the degree of node A. Suppose that C number of edges are cut due to initial partition. Assume node A is in partition P1 initially; and we are moving it from partition P1 to P2, the change in C will be

$$D_A = E_A - I_A \quad (1)$$

So by moving node A from one partition P1 to other partition P2, we will get new number of cut edges and that will be $C - (E_A - I_A)$. This happens due to movement of node A from one to another, edges attached to node A that was cut before will be uncut edges now and uncut edges will become cut edges. D_A is an improvement for node A; if it is positive improvement then it will reduce the cut size otherwise it will increase the cut size.

To find the set of nodes in the graph to be swapped between sub-circuits, a series of node pair-swaps has been used by the KL algorithm. Let us assume that nodes A and B are present in partition P1 and partition P2 respectively. The gain of swapping a pair of nodes A and B is

$$\Delta g = D_A + D_B - 2 * c(A,B) \quad (2)$$

where $c(A,B)$ is the connection weight between A and B; if an edge exists between nodes A and B, then $c(A,B)$ = edge weight (here 1), otherwise, $c(A,B) = 0$. This equation shows how the edge between A and B is taken into consideration. The gain considers the interconnection edges twice in improvement equation, once for each node. Even if we swap the nodes A and B, the interconnection edge is going to be cut in both situation and hence we have to consider it in the gain equation.

Once all the improvement D_{node} values for all nodes are calculated; we can calculate the gain values for all swapping pairs of node. But the important thing is this swapping is not permanent, it's just tentative swapping and actual movement of nodes between partitions does not takes place. Before proceeding with the process, the total number of cut edges which can be termed as cutsize must be recorded as initial cutsize. Now we have the list of all gain values; now sort the list of nodes pairs into the descending order for both the partitions P1 and P2 and then select the pair with the greatest improvement. Then tentative swapping of these two nodes A and B are performed and calculate the reduction in cutsize. Here it is important to update all the improvement values for any node which is attached to node A or node B. If the node A was previously attached with node X and that interconnecting edge was a cut edge, improvement of node X has to be reduced by one; and if it was uncut edge then improvement has to be increased by one. The gain values of any pair of nodes which also contains node X has been updated and the sorted list in descending order is also updated.

Now we have to mark nodes A and B has been swapped; actually it's not swapped yet but temporal swapping also we have to note down and deleting the pair of nodes in the sorted list which contains these two nodes as the node which has been swapped with another node, cannot swap it again

for the second time. And make record of this pair of nodes along with new total number of cutsizes obtained after the temporal swapping of these two nodes. After updating the improvement values along with the gain values of remaining pairs and sorted list of pairs of nodes; select the pair of nodes with the largest gain value and repeat the previous steps again. Again do the temporal swapping of two nodes and record the required results. This nodes pair selection process is continued till all the available pairs are taken into consideration. This process causes each node to be swapped tentatively with another node in the other partition. If actual swapping of nodes pairs had been done, then we would get the same partition from where we started. If we have $2n$ nodes in the graph (means $2n$ logic gates in the circuit); a list of n tentative swaps has been recorded along with its cutsizes for each pair. To find out the minimum cutsizes; we have to determine the set of nodes to be swapped from the tentative nodes swaps list. Compare the cutsizes recorded at each swaps with the initial cutsizes. The maximum positive gain G_m corresponds to the best prefix of m swaps within the swap sequence of a given pass. These m swaps lead to the partition with the minimum cut cost encountered during the pass. G_m is computed as the sum of Δg values over the first m swaps of the pass, with m chosen such that G_m is maximized.

$$G_m = \sum \Delta g \quad (3)$$

More positive G_m means minimum value of cutsizes. So if the cutsizes is minimum as compared to initial cutsizes, then actual swapping has been performed from the start of the list of tentative m swaps and all the nodes pairs has been included which gives the minimum cutsizes. If the minimum cutsizes is not less than or equal to the initial one; the algorithm has been terminated.

All this process discussed previously represent just one iteration of the Kernighan-Lin heuristic iterative algorithm. After getting the set of nodes to be swapped, the algorithm do the complete process again, re-computing all the improvement and gain values and calculate again the maximum positive gain and minimum cutsizes. This iteration is continued till no improvement is possible and then the entire algorithm stops. In this way, the KL algorithm is typically an iterative-improvement algorithm.

4. Data Structures

The choice of data structure strongly depends on the cost functions, gains, and the characteristic of VLSI circuitry [7]. In this paper, Kernighan-Lin algorithm is implemented using PERL (Practical Extraction and Report Language). It is very powerful and flexible language which supports so many functions of a high-level programming language such as C. Even we can say that PERL borrowed so many features from C itself. The main reason of using PERL is that unlike other languages, it does not require special compiler and linker to turn the programs written into working code [8]. In PERL, we have to write the program and run it. So it produces the quick solutions and the computation time is less and hence in this paper to get the quick response, we use data structures in PERL to implement the KL algorithm. Even in VLSI industries PERL language is used for report generation and data handling. So it will help to resolve the compatibility issues. As KL algorithm is a heuristic iterative algorithm,

PERL provides flexibility and quick response which reduces the computation time of the algorithm.

KL algorithm takes a text file as an input which includes information like total number of nodes in the graph (means total number of logic gates in the circuit), list of nodes and the elements of adjacency matrix representing the graph to be partition. These elements of adjacency matrix tells there is interconnecting net (edge) available between the nodes or not. This algorithm also produces output in the text file only. So it helps to handle the results if it is required in other process and it is recorded permanently. To read and write to files we should create something called handles which refer to the files and to create the handles we use the OPEN command. This handle will be used for reading and writing. Each line from the input file is being read and processed by storing in the scalar variable. The contents of input file for 8 nodes are as follows:

Number of nodes - 8

The list of nodes - {1 2 3 4 5 6 7 8}

Two dimensional Adjacency graph -

```
0 1 0 0 1 1 0 0
1 0 0 0 1 1 0 0
0 0 0 1 0 1 1 1
0 0 1 0 0 0 1 1
1 1 0 0 0 1 0 0
1 1 1 0 1 0 0 0
0 0 1 1 0 0 0 1
0 0 1 1 0 0 1 0
```

The main data structures used in implementing KL algorithm are arrays and hashes in PERL. An array holds a list of scalar values of different types. When a list is assigned to an array variable, an element of that array variable can be accessed as a scalar variable. In KL algorithm, arrays are mostly used to store the temporary values while finding the maximum values of improvement and gain. But using arrays in every situation is complicated as it is important to store the information of each elements. Hence there is different kind of array is available in PERL, which can access array variables by using any scalar value. Such arrays are called associative arrays (hashes). In associative arrays, the index is referred to as a key and the corresponding element as a value. This key-value pair makes the task easier as index also stores the information of elements type. Most of the processes are handled by associative arrays. Initial partition, improvement values and gain values are stored in hashes only. Nodes are stored as keys and its respective values are stores as values. Even updation of improvement values after completing iteration is also stored in hashes. Hashes to arrays and arrays to hashes conversion and the „reverse“ function in hashes which reverse the roles of keys and values provides flexibility to the implementation.

Output file contains the initial partition of the nodes, the final partition of nodes, the adjacency matrices of new generated partitions and the cutset (the set of edges to be cut). The contents of output file after implementing KL algorithm on the input given above are as follows:

Number of nodes in graph - 8

The list of nodes – {1 2 3 4 5 6 7 8}
*****INITIAL PARTITION*****
First partition: {1 2 3 4}
Second partition: {5 6 7 8}
*****FINAL PARTITION*****
First partition: {6 1 2 5}
Second partition: {8 4 3 7}
Adjacency matrix for partition 1 is:
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
Adjacency matrix for partition 2 is:
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
The cutset is: {{6, 3}}

5. Future Scope

This paper discussed the solution of two-way circuit partitioning problem on a set of $2n$ nodes (logic gates). But the KL algorithm using PERL can also be extended to perform k -way partitions on a set of kn objects, using the same two-way process as a basic tool. There is a chance to partition the graph into k sets of size n ; as we have KL heuristic algorithm for partitioning bisection. Firstly the graph is divided into two partitions P_1 and P_2 by our algorithm. There is a scope to consider each partition as a complete graph and apply the two-way partitioning KL heuristic algorithm to get 4-way partitioning. Again we can assume each partition as a complete graph and do the same to get the required k -way partitions. Even in this paper, we assume the weights of the edges are same. But it is not necessary that the cost of all interconnection nets are same; so the algorithm is also extended to solve the different weighted edges partitioning problems. It can be possible by considering the weights in computing improvement and gain values.

6. Result and Conclusion

The KL algorithm for two-way partitioning is implemented using data structures like arrays and associative arrays (hashes) in Perl. List of nodes and interconnecting nets is stored in the text file. This text file is taken as input file and the new partition is stored in output file. Text files are handled by filehandles. This implemented KL algorithm is used to solve various circuit partitioning problems. The partition of odd number of nodes is also handled by this implemented KL algorithm using dummy nodes. As the KL algorithm is implemented using PERL, the computation time is less and information is easily available and stored for other VLSI applications. The number of edges for the circuit consider in this paper has been reduced from 9 to 1 using our implemented KL algorithm.

References

[1] G. Moore, "Cramming more components onto integrated circuits" Electron. Mag., vol. 38, no. 8, pp.

- 114–117, Apr. 19, 1965.
- [2] R. R. Schaller, "Moore's law: Past, present, and future," IEEE Spectrum., pp. 52–59, Jun. 1997.
- [3] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," IEEE Transactions on Computers," C-33, 5, pp.438-446, 1984.
- [4] Ali Dasdan and Cevdet Aykanat, "Two Novel Multiway Circuit Partitioning Algorithms Using Relaxed Locking", IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 16, No. 2, February 1997.
- [5] Shin'ichi Wakabayashi, Kazunori Isomoto, Tetsushi Koide, Noriyoshi Yoshida, "A Systolic Graph Partitioning Algorithm for VLSI Design", IEEE International Symposium on Circuits and Systems, vol. 1, pp. 225-228, 1994.
- [6] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", The BELL System Technical Journal, February 1970.
- [7] Sao-Jie Chen and Chung-Kuan Cheng, "Tutorial on VLSI Partitioning", the Gordon and Breach Science Publishers (Overseas Publishers Association), VLSI Design, Vol. 00, No. 00, pp. 1 – 43, 2000.
- [8] Larry Wall, Tom Christiansen, Jon Orwant "Programming Perl, 3rd Edition", July 2000.