Clustering Based Acyclic Multi-way Partitioning

Eric S. H. Wong
Department of CSE
The Chinese Univ. of H.K.
shwong@cse.cuhk.edu.hk

Evangeline F. Y. Young
Department of CSE
The Chinese Univ. of H.K.
fyyoung@cse.cuhk.edu.hk

W. K. Mak
Department of CSE
University of S. Florida
wkmak@csee.usf.edu

ABSTRACT

In this paper, we present a clustering based algorithm for acyclic multi-way partitioning. Many existing partitioning algorithms have shown that clustering can effectively improve the solution quality. However, most of them do not consider the signal direction and thus cannot maintain the acyclic property. Our algorithm is based on clustering by computing the modified fan-out free cones. Fan-out free cone clustering can reduce a graph to a smaller and sparser one, and maintain the acyclic property at the same time. Experimental results showed that our algorithm compares favorably with the previous best acyclic multi-way partitioning algorithm in cut-size.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design (CAD)

General Terms

Design, Algorithm

Keywords

CAD, Partitioning, Clustering, Acyclic, Multi-way

1. INTRODUCTION

Circuit partitioning is a critical stage in the VLSI design cycle. Good partitioning techniques can break down a complex system to smaller subsystems such that each subsystem can be designed independently to speed up the design process. The existing partitioning algorithms can be classified into a few categories including the group migration approach [5, 3], the network flow approach [8, 6, 7] and the analytical approach [9, 10]. Some algorithms give two-way partitioning [5, 3] while some give multi-way partitioning [1, 6, 4]. A number of heuristic approaches have been developed to further improve a partitioning solution. They includes logic replication [8], multilevel approach [1, 4] and clustering [1]. In logic replication, some nodes are selected and duplicated into two or more partitions in order to reduce the cut-size. In multilevel approach, a sequence of successively coarser hypergraphs is constructed, and a bisection is obtained from it. The bisection of the original graph is obtained by successively projecting and refining the coarser hypergraph to the original graph. clustering is another approach [1] that groups the nodes in a network into clusters. It can significantly reduce the problem size and give a simpler clustered network before applying a partitioning algorithm.

Besides cut size, the longest delay of a path is also an important issue to be considered. Acyclic partitioning is an effective way to upper bound the largest number of inter-partition delay along any path. The acyclic multi-way partitioning problem was defined in [2], it differs from the general partitioning problem because of the restriction that the edges between different partitions of a solution cannot form a directed cycle. Acyclic multi-way partitioning finds application in pipelining of multi-chip designs, partitioning based logic minimization, and parallel circuit simulation as described in [2].

Many existing clustering based partitioning algorithms have shown that clustering can effectively improve the solution quality. However, most of them do not take the signal direction into account. Therefore, it is not possible to apply them directly to obtain an acyclic partitioning. In [2], an algorithm based on the maximum fanout free cone decomposition followed by a restricted version of the FM algorithm was proposed for the acyclic multi-way partitioning problem. In this paper we show that a simple two-phase clustering process based on a modified fanout free cone decomposition can yield superior acyclic multi-way partitioning than that in [2].

In this paper, we will discuss the acyclic multi-way partitioning problem and present a clustering based partitioning algorithm to solve the problem. The rest of the paper is organized as follows. In section 2, we will formulate the multi-way acyclic partitioning problem. Our clustering based acyclic multi-way partitioning method will be introduced in section 3. The details of our algorithm will be discussed in section 4 to 7. In section 8, we will present some experimental results. Finally, a conclusion will be given in section 9.

2. PROBLEM FORMULATION

In this paper, we want to solve the acyclic multi-way partitioning problem in a combinational network. A combinational circuit is represented by a directed acyclic graph $G(V,E)$ where V is a set of nodes representing the gates and E is a set of directed edges representing the interconnections between the gates. The fan-out of a node is the number of edges incident from it and the fan-in of a node is the number of edges incident to it. Primary input is a node with zero fan-in and primary output is a node with zero fan-out. In the given acyclic graph

GLSVLSI'03, April 28–29, 2003, Washington, DC, USA.

G, each node in V is assigned a unit weight except the primary input and output nodes. The primary input and output nodes are assigned a zero weight each. The weight represents the area occupied by the node. We assume that the areas of all the gates are the same and each has one unit area. Note that an acyclic partitioning of a sequential circuit can be obtained as follows. We can compute an acyclic partitioning of the combinational network obtained by removing all the sequential elements. Then we can put back the sequential elements into the proper partitions.

Definition 1 An Acyclic K-Way Partitioning Problem: Given a directed acyclic graph $G(V, E)$, partition the set of nodes V into k disjoint subsets, V_1, V_2, ... , V_k, such that the sizes of the subsets do not exceed the size constraints, A_1, A_2, ... , A_k, the cut-size is minimized, and the partitioned solution is acyclic, i.e. there is no partitions V_i and V_j such that i≠j, and there are directed paths running from V_i to V_j and from V_j to V_i.

3. CLUSTERING BASED ACYCLIC MULTI-WAY PARTITIONING

In clustering based acyclic multi-way partitioning, the given network is first clustered into a sparser network. We use an idea similar to that of maximum fan-out free cone to cluster the nodes. The maximum fan-out free cone decomposition aims at minimizing the number of edges coming out from a cluster. This is a good strategy since it reduces the total number of edges inside a clustered network globally. After the decomposition, the number of edges is equal to the number of clusters because there is only one edge coming out from each fan-out free cone. This produces a good initial solution for the partitioning process since we aim at minimizing the cut sizes between the resultant partitions. After the clustering phase, the nodes inside a cluster are collapsed to form one node. As a result, the clustered network becomes simpler and sparser. The number of edges and nodes are fewer compared with the original network. It is easier to perform the subsequent partitioning task as the size of the solution space is directly proportional to the number of nodes and edges. In our algorithm, we will use the *modified fan-out free cone decomposition* to perform clustering. Details of the process will be shown in the next section.

In the partitioning phase, we use a method similar to that in the clustering phase because we believe that the modified fan-out free cone decomposition is a good strategy. However, in the partitioning phase, the size constraint is set to the predefined partition size as we want to fill up each partition as much as possible. We will work on the clustered network in this phase. As a clustered node is actually a collection of nodes, the weight of each clustered node can be large. Therefore, it is hard to obtain an ideal fan-out free cone to fit the size of a partition. In such cases, we will first find a maximally fit cone to put into a partition. Then, we will try to fill up the partition as much as possible by taking in smaller cones until no other matches is possible. We will discuss this selection process in details in section 6.

4. MODIFIED FAN-OUT FREE CONE DECOMPOSITION

We use the idea of fan-out free cone to find the clusters and partitions while maintaining the acyclic condition. An input cone of v, $cone(v)$, is a set of nodes consisting of v and a subset of its predecessors such that any path connecting a node in $cone(v)$ to v lies entirely within $cone(v)$. We can observe that there exists many input cones for a specific node (see Figure 1). A fan-out free cone of v, $FFC(v)$, is an input cone of v such that any fan-out of a node in $FFC(v)$, except that of node v, must also be in $FFC(v)$. Fan-out free cone of a node is again not unique (see Figure 2). But the number of fan-out edge from any fan-out free cone must be equal to one. We have made use of this size flexibilities to match the size constraints in the clustering and partitioning phases.

In our modified fan-out free cone decomposition, after we locate a cone of a node, we will remove the nodes inside the cone and the edges connected to this cone from the network before proceeding to form the next cluster. As a result, some nodes connected to the cone will become a primary output node after the removal. This step gives a higher probability for the later steps to form larger clusters or partitions. A simple example is shown in Figure 3. Note that the number of fan-out edges from a cone may be larger than one in this modified decomposition method. Experimental results have shown that it is valuable to do such a modification.

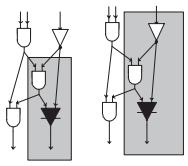

Figure 1: Two Different Cones of a Node

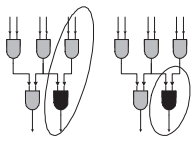

Figure 2: Two Different Fan-out Free Cones of a Node

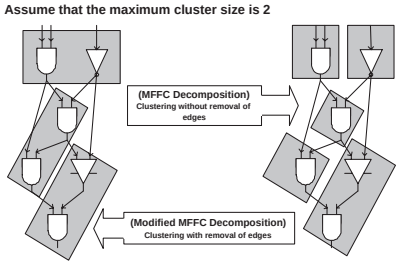

Figure 3: Results Obtained from Modified MFFC Decomposition and MFFC Decomposition

5. CLUSTERING PHASE

In the clustering phase, the given network is clustered to form a sparser network using the modified maximum fan-out free cone decomposition. Initially, a primary output is selected randomly. It acts as the starting point of the clustering process. We denote this selected node as v and assign it to a cluster C. We will try to fill up the cluster by taking in nodes

step by step until the predefined cluster size is reached. We select a fan-in node of the nodes in C randomly and denote it as u. A testing process is then performed on u to ensure that the newly clustered nodes do not violate the property of a modified fan-out free cone. If we cluster node u into C, the node that can be reached from node u must also be clustered into C. Otherwise, the fan-out free property cannot be maintained. So, we need to find the number of nodes that can be reached from node u. If the cluster C can take in the whole set of nodes that can be reached from u, we will cluster all these nodes into C. Otherwise, we will reject this fan-in node u and try another one. If no nodes can be selected, the process will stop and one cluster is formed. An example of the selection process is given in Figure 4. The resultant cluster will have a size equal to or smaller than the predefined cluster size. After one cluster is formed, we will remove the clustered nodes and the edges connecting this new cluster from the network. We will then work on the remaining network similarly as above. We will build another cluster by starting from a randomly picked primary output node of the remaining network and repeat the above process until all the nodes are clustered. An example of the whole process is shown in Figure 5 and the algorithm is given below.

Clustering Algorithm

```
Clustering(Network)
 1. dolist = 0
 2. Do
 3.       If dolist>0
 4.            i = a randomly selected node in dolist
 5.       Else if there is a primary output node
 6.            i = a randomly selected primary output node
 7.       Else exit
 8.       cluster = cluster + 1
 9.       Find_Cluster(Network, dolist, i)
10.      Remove the clustered nodes and their edges from
           the network
11. End_DO

Find_Cluster(Network, dolist, i)
 1. Add the fan-in nodes of i to local_dolist
 2. While local_dolist are tested is not empty
 3.       If local_dolist>0
 4.            i = a randomly selected node from local_dolist
 5.            j = numbers of node that can be reached by i
 6.            If current_cluster_size + j <max_cluster_size
 7.                 Assign the set S of nodes that can be reached
                      by i to the current cluster
 8.                 Add all the fan-in nodes of S to dolist and
                      local_dolist
 9.                 Remove the clustered nodes from dolist and
                      local_dolist
10.       Else exit
```

6. PARTITIONING PHASE

In the partitioning phase, the clustered network is partitioned into the desired size. Basically, the approach used in partitioning is the same as that in clustering. The main difference is that we work on the clustered network and the cluster size constraint is set to the partition size constraint. There is no limit for the number of clusters in the clustering phase. In the partitioning phase, we must keep the number of partitions to a predefined value. Therefore, we cannot treat the size of each partition as loosely as in clustering. We must match each partition as much as possible. In the clustering phase, if no fan-in nodes can be selected to be put into a cluster anymore, we will close the cluster and continue the clustering process with a new empty cluster. But in the partitioning phase, if

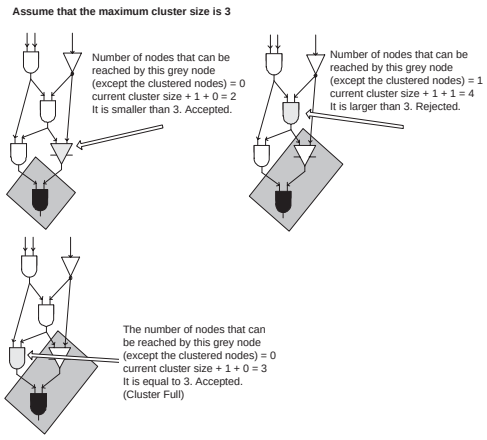

Figure 4: Fan-in Node Selection in the Clustering Phase

no fan-in clusters can be selected to put into a partition anymore, we will first remove the currently partitioned clusters together with their incoming edges from the network and then continue with another cluster that is a primary output of the remaining network to put into the partition until the partition size is reached or no clusters can fit into it. An example of the whole partitioning process is shown in Figure 6 and the algorithm is given below.

Partitioning Algorithm

```
Partitioning(Clustered_Network)
 1. dolist = 0
 2. Do
 3.       If dolist>0
 4.            i = a randomly selected cluster from dolist
 5.       Else if there are primary output clusters
 6.            i = a randomly selected primary output cluster
 7.       Else exit
 8.       partition = partition + 1
 9.       Find_Partition(Network, dolist, i)
10.       If current partition is not full yet
11.            partition = partition - 1
12.       Remove the partitioned clusters and their edges from
            the network
13. End_Do
```

The algorithm of *Find_Partition()* is exactly the same as that of *Find_Cluster()*, so we do not repeat it in this section again.

7. THE ACYCLIC CONSTRAINT

As we are performing acyclic partitioning, we must ensure that the acyclic property is not destroyed in each clustering and partitioning step. We have the following lemmas about the correctness of our algorithm. Since the length of the paper is limited, the proofs of the lemmas are not included here.

Lemma 1: The modified maximum fan-out free cone clustering process produces acyclic clustered network only.
Lemma 2: The partitioning process produces an acyclic partitioned network only.

8. EXPERIMENTAL RESULTS

In order to evaluate the performance of our algorithm, we implemented our clustering based partitioning method using

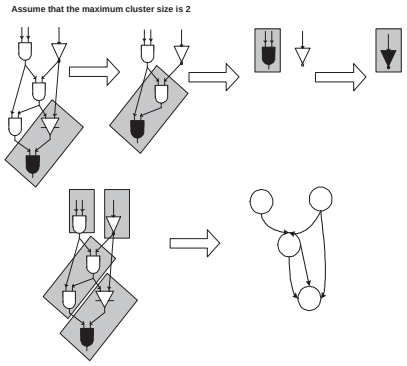

Figure 5: An Example of the Clustering Phase

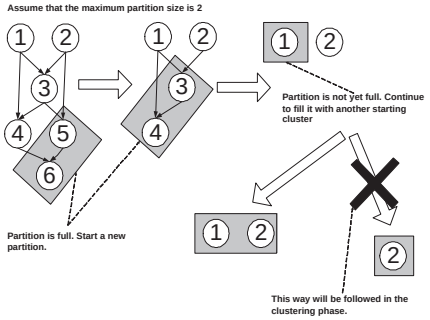

Figure 6: An Example of the Partitioning Phase

C language. The testing platform is Sun Ultra 5/270. The benchmarks are obtained from ISCA85. These data sets contain information of the signal direction. Therefore, we can use it to test our algorithm. Moreover, we can compare our results with that of another clustering based algorithm [2] using the same data suit which is the lastest best results of this problem. We compared our results with two algorithms, K-AFM and K-MAFM. The experimental results of K-AFM and K-MAFM are obtained from [2]. For all the experiments, the number of partitions is 8 and the maximum cluster size is 1/2 of the target partition size. Each partition allows ±5% deviation from its target size. The results of K-AFM and K-MAFM shown in Table 2 are the best partitioning result obtained by running the program ten times. The results of our algorithm are also the best results obtained by running the program until ten partitioning results are generated. Table 1 shows the characteristics of the benchmarks. Table 2 shows the cut-size results of different partitioning algorithms and the runtime of our algorithm. Note that the runtimes of K-AFM and K-MAFM are not shown because they are not reported in [2].

In comparisons with K-AFM and K-MAFM, our algorithm gives better performance in most of the cases. For smaller circuits, our algorithm is not as good as K-AFM algorithm. However, the results of our algorithm are better for large size circuits. The average improvement to the K-MAFM algorithm is 30%. Our method out-performs K-MAFM when the circuit size increases. This result suggests that the performance of the FM algorithm drops when the size of the circuit increases.

Circuit	No. of Gates	No. of PIs	No. of Edges
c880	383	60	729
c1355	546	41	1064
c1908	880	32	1064
c2670	1193	233	2076
c3540	1669	50	2939
c5315	2307	178	4386
c6288	2416	32	4800

Table 1: Characteristics of the Benchmarks

Circuit K=8	Cluster size	K-AFM (net-cut)	K-MAFM (net-cut)	Our Algorithm (net-cut)	Our Algorithm Runtime (Sec)
c880	28	156	52	68	23.6
c1355	37	184	23	80	34.3
c1908	57	327	112	84	26.7
c2670	89	443	246	115	41.3
c3540	107	575	232	137	61.8
c5315	155	866	238	218	114.4
c6288	153	491	487	373	85.2

Table 2: Results of Different Partitioning Algorithms

9. CONCLUSIONS

In this paper, we presented a new acyclic multi-way partitioning algorithm. We first use the modified fanout-free cone decomposition to cluster a given network. This decomposition effectively reduces the given network to a smaller and sparser one while maintaining the acyclic property of the network. Then we use this decomposition again to further partition the clustered network into the desired number of partitions. Our algorithm is able to obtain acyclic multi-way partitioning solutions with smaller cut-sizes compared to the best algorithm reported previously [2].

10. REFERENCES

[1] C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. *DAC*, pages 530–533, 1997.

[2] J. Cong, Z. Li, and R. Bagrodia. Acyclic multi-way partitioning of boolean networks. *Proceeding ACM/IEEE 31st Design Automation Conference*, pages 670–675, June 1994.

[3] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. *Design Automation Conference*, pages 241–247, 1982.

[4] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *ACM/IEEE Design Automation Conference*, pages 343–348, 1999.

[5] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–307, 1970.

[6] H. Liu and D. F. Wong. Network flow based circuit partitioning for time-multiplexed fpga. *ICCAD*, pages 497–504, 1998.

[7] H. Yang and D. F. Wong. Efficient network flow based min-cut balanced partitioning. *IEEE ICCAD*, pages 50–55, November 1994.

[8] H. Yang and D. F. Wong. New algorithms for min-cut replication in partitioned circuits. *ICCAD*, pages 216–222, 1995.

[9] R. Boppana. Eigenvalues and Graph Bisection: An Average-Case Analysis. *IEEE Symp. on Foundations of Computer Science*, pages 280–285, 1987.

[10] P. K. Chan and D. F. Schlag and J. Zien. Spectral K-way Ratio-Cut Partitioning and Clustering. *Proceeding ACM/IEEE Design Automation Conference*, pages 749–754, 1993.