Network Partitioning into Tree Hierarchies[*]

Ming-Ter Kuo, Lung-Tien Liu[†], and Chung-Kuan Cheng

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093

[†]AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

This paper addresses the problem of partitioning a circuit into a tree hierarchy with an objective of minimizing a global interconnection cost. An efficient and effective algorithm is necessary when the circuit is huge and the tree has many levels of hierarchy. We propose a heuristic algorithm for improving a partition with respect to a given tree structure. The algorithm utilizes the tree hierarchy as an efficient mechanism for iterative improvement. We also extend the tree hierarchy to apply a multi-phase partitioning approach. Experimental results show that the algorithm significantly improves the initial partitions produced by multiway partitioning and by recursive partitioning.

1. Introduction

In the design of large scale circuit systems, hierarchical approaches are widely applied in order to deal with the high complexity of various design problems. Physically, the circuit system is also organized in multiple levels of hierarchy and divided into system level, board level, chip level, macrocell level, and so on. At each level, the circuit is partitioned into subcircuits that are implemented in separate components with limited resources. Therefore, a partition can be viewed as a mapping from a given design hierarchy into a hardware tree structure that satisfies certain constraints. Practically, there are many hierarchies into which we can partition a circuit. The problem is how to find a hierarchy and a partition so that the interconnection cost between components in the entire system is minimized. In this paper, we call this the hierarchical tree partitioning problem.

In the implementation of a circuit, the wiring cost of a net that connects different components may vary at different levels. Usually, the interconnection cost is greater at higher levels. To formulate the cost of a multipin net, we use a cost weighting factor for the I/O pins consumed on a component

[*]This work was supported in part by grants from the NSF project MIP-9315794 and the California MICRO program.

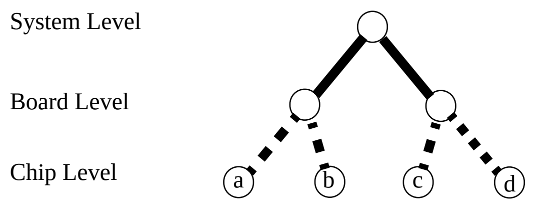

Fig. 1. A system hierarchy.

at each level. For example, Fig. 1 shows a system hierarchy with two boards containing two chips each. Let the cost weighting factor be 2 for I/O pins on each board and 1 for I/O pins on each chip. Then a net that crosses chips **a**, **b** and **c** will consume 3 I/O pins on the chips and 2 I/O pins on the boards. The total interconnection cost of this net will be 7 since each I/O pin on the board has a cost weighting factor of 2.

Conventionally, two approaches may be used for mapping the circuit into a tree hierarchy: recursive top-down partitioning and bottom-up clustering. Both approaches optimize the partitioning one level at a time. However, a good partitioning result at one level may sacrifice the quality of the entire hierarchical tree. Suppose we want to partition the circuit in Fig. 2 into the tree structure in Fig. 1 with an objective of minimizing the total interconnection cost. In Fig. 2, each node is labeled with its size and each net is labeled with a weight that denotes the number of connections. Using recursive two-way partitioning, the circuit in Fig. 2(a) is first partitioned into two boards with a minimum cut of 5 (denoted by a bold line) if the capacity of each board is 20. To connect the four nets in this cut, an interconnection cost of 20 (5×2 for each part) between boards and an interconnection cost of 10 between chips (5 for each part) are required. At the chip level, each part of the circuit is further partitioned into two chips with a minimum cut of 12 (denoted by a dashed line) if the capacity of each chip is 10. These two cuts totally create interconnection cost of 48 between chips, giving a total cost of 78. However, if the same circuit is partitioned as shown in Fig. 2(b), it has a smaller cost of 72 even though the cut at the board level is not a minimum cut. This example shows that the recursive partitioning approach may not obtain the minimum global cost when mapping the circuit into a tree hierarchy.

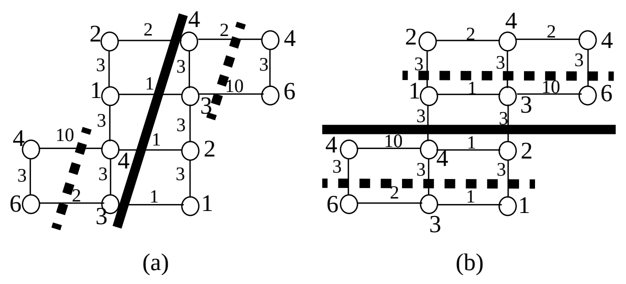
Fig. 2. A hierarchical tree partitioning example.
(a) A partition with an interconnection cost of 78.
(b) A partition with an interconnection cost of 72.

In considering the global cost, Vijayan proposed a tree partitioning of hypergraphs for VLSI design applications [5]. He formulated a cost function that minimizes the routing cost of hyperedges. However, the tree in [5] has no levels of hierarchy and each vertex of the tree represents a block of the partition in a floorplan. As an extension, vertices in our hierarchical tree partitioning problem denote blocks of partitions at different levels in the system hierarchy. Moreover, the tree structure in [5] is fixed with a given floorplan. In the hierarchical tree partitioning problem, the tree can be dynamically changed to fit into an optimal hierarchical structure of a design.

As it will be shown in Section 2.3 of this paper, the hierarchical tree partitioning problem is an NP-hard problem. It is still NP-hard even if the tree structure is specified. For partitioning a hypergraph into a specified tree structure, we propose an iterative improvement algorithm to minimize the global interconnection cost. The algorithm utilizes the tree hierarchy as a sorting tree to efficiently retrieve the next move for improvement in each iteration. In our implementation, we extend the tree structure to represent the netlist and apply a multi-phase approach to further improve the partition. Experimental results show that our algorithm can effectively reduce the interconnection cost of partitions.

The rest of the paper is organized as follows. In Section 2, we formally define the hierarchical tree partitioning problem and discuss the NP-hardness of the problem. The proposed heuristic algorithm and the multi-phase approach are described in Section 3 and Section 4, respectively. We present the experimental results in Section 5 and conclude the paper in Section 6.

2. Hierarchical Tree Partitioning

2.1. Definitions

Given a circuit system, we use a hypergraph $H=(V, E)$ to represent its netlist, where V ($|V| = n$) is the set of nodes and E ($|E| = m$) is the set of nets. Each node v in V has size $s(v)$; each net e in E is a subset of V with cardinality $|e| \geq 2$ and has associated weight $c(e)$.

The hierarchy of the circuit system in partitioning is represented by a rooted tree. The height of the tree, L, is the length of the path from the root to the leaves. The level of a vertex u in the tree is the height of the subtree rooted at u (see Fig. 3). At each level, a vertex has a maximum number M of branches. The interconnection cost between vertices at each level l has a weighting factor $w(l)$. All the leaves are at the same level and represent the basic components. They all have a same size upper bound B that represents the capacity of a basic component. Note that we use “node” and “vertex” to refer to hypergraph nodes and tree vertices respectively in this paper.

A hierarchical tree partition P of a hypergraph $H=(V, E)$ assigns the nodes in V to the leaves of a tree T. Let $N(u)$ denote the nodes that are assigned to leaf u in P. A partition P is feasible, if for any leaf u in T, the total size of nodes in $N(u)$ does not exceed the size upper bound, i.e. $\sum_{v \in N(u)} s(v) \leq B$.

In hierarchical tree partitioning, assigning a node to a vertex u in the tree also assigns the node to the parent vertex of u. Therefore, the tree partitioning forms a multi-way partition with respect to each level of the tree. For example, in Fig. 3, the hierarchical tree partitioning forms a 8-way partition at level 0, where the hypergraph is divided into 8 parts. Similarly, there is a 4-way partition at level 1 and a two-way partition at level 2.

For each net e in E, we define $span(e, l)$ to be 0 if e connects exactly one block at level l, and f if the net connects exactly f blocks where $f \geq 2$. Here, $span(e, l)$ represents the number of blocks at level l to which net e contributes I/O pin cost. ($span(e, L)$ is always 0.) The interconnection cost of each net e is defined as the total weighted cost on the blocks to which it connects at all levels, i.e. $cost(e) = \sum_{0 \leq l \leq L-1} span(e, l) \times w(l) \times c(e)$.

2.2. Problem Statement

The Hierarchical Tree Partitioning Problem: Given a hypergraph $H=(V, E)$, the size upper bound B of a leaf, the maximum number M of branches of a vertex, and function w ($w(l)$ defines the cost weighting factor at level l), find a feasible tree hierarchy T and a partition P for T such that $\sum_{e \in E} cost(e)$ is minimized.

In this paper, we set the weighting factor of the interconnection cost to have an exponential growth from the bottom level to the top level, i.e. $w(l) = 2^l$. The weighting factor is set to derive a good metric on the interconnection cost of a partition in VLSI applications. With the factor being expo-

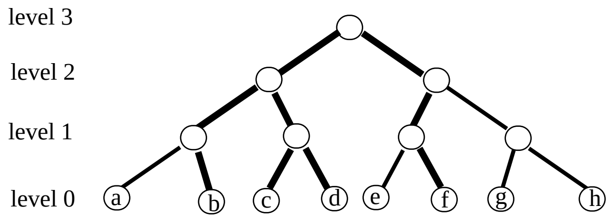

Fig. 3. A rooted tree hierarchy with height 3.

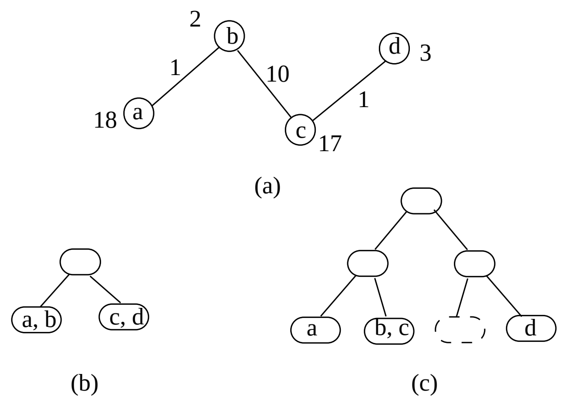

Fig. 4. An example for determining the height of a tree hierarchy. (a) A hypergraph with 4 nodes and 3 nets. (b) A partition with a cost of 20 in a tree hierarchy with height 1. (c) A partition with a cost of 8 in a tree hierarchy with height 2.

nential in the level number, the metric discourages interconnections between nodes separated by many levels of hierarchy. For different applications, the weighting factor can be adapted to optimize the entire system.

2.3. Complexity of the Hierarchical Tree Partitioning Problem

To find an optimal tree hierarchy, we may start by determining the height of the tree. Due to the cost weighting factor at each level, intuitively, partitioning a circuit into a tree hierarchy with a lower height could achieve a smaller cost. Fig. 4 shows a counterexample to this statement. The circuit in Fig. 4(a) has four nodes; each node is labeled with its size and each net is labeled with its weight. Suppose the size upper bound of a leaf in the tree structure is 20. Fig. 4(b) shows the optimum partition for mapping the circuit into a tree with height 1. This partition has a cost of 20 since the net connecting **b** and **c** contributes a cost of 10 to each part. On the other hand, Fig. 4(c) shows another partition where the height of the tree is 2. This partition has a cost of only 8. Thus, a tree hierarchy with a lower height does not necessarily generate a partition with a smaller interconnection cost.

The problem of determining the optimal tree hierarchy with a height not greater than a specific number K is NP-hard. We can reduce the BIPARTITION problem [2, page 255] to this problem by setting K=1 and the size bound of each basic component to be the size bound of the BIPARTITION problem. Therefore, the hierarchical tree partitioning problem, which has no constraint on the height of the tree, is also NP-hard. Even if the tree structure is specified, the problem is still NP-hard since multiway partitioning is a special case of this problem.

3. A HEURISTIC HIERARCHICAL TREE PARTITIONING ALGORITHM

Since the problem of finding the optimal tree hierarchy is NP-hard, we focus on the hierarchical tree partitioning problem with a given tree structure. We propose an iterative improvement heuristic for the problem to improve an initial partition.

3.1. Outline of the Algorithm

We show the outline of the heuristic algorithm for a specified tree structure in Fig. 5. Starting with given initial partition, the algorithm adopts the local search method of Fiduccia-Mattheyses algorithm [1] for iterative improvement by moving a node from one leaf to another leaf of the tree. The initial partition can be any feasible partition of the hypergraph for the hierarchical tree.

In Fig. 5, Step 1 to Step 5 show one pass of the heuristic in which each node is considered once for possible movement. In Step 1, the gain (reduction in cost) of moving each node is computed and stored. A node is subsequently considered for movement in Step 2. Then, the following steps update the gain (Step 3) and check the condition for moving the next unlocked node (Step 4) or start another pass (Step 5). Because the computation of the cost function is complicated, we devise an efficient method described next for computing the gain of moving a node. Also, we use a tree structure for retrieving the next node to be moved.

```
Given a hypergraph H=(V, E), a tree T, and an initial partition P.
Step 1. Compute the gains of all the nodes in V with respect
        to the current partition P and unlock all the nodes;
Step 2. Select an unlocked node v to move;
Step 3. Move v and log the move with the cost;
        Lock node v and update the gains after moving v;
Step 4. If there are unlocked nodes, goto Step 2;
Step 5. If the smallest cost logged is better than the cost of P,
            Update P to the best partition logged and goto Step 1;
        Otherwise, stop.
```

Fig. 5. Outline of the partitioning algorithm.

3.2. Computing the Gain and the Destination for a Node

When partitioning a huge circuit system, the number of levels and number of branches of a vertex in the tree structure can be large. This means there is an enormous number of leaves in the tree to which a node may be moved. Therefore, it is impractical to maintain all the possible moves and gains in *bucket arrays* as it is done in the FM-based multiway partitioning algorithm [4]. In our heuristic, we use an efficient and flexible method to maintain the best possible moves. For each node v, we only maintain a destination vertex (denoted by *dest*(v)) and its gain (denoted by *gain*(v)) in

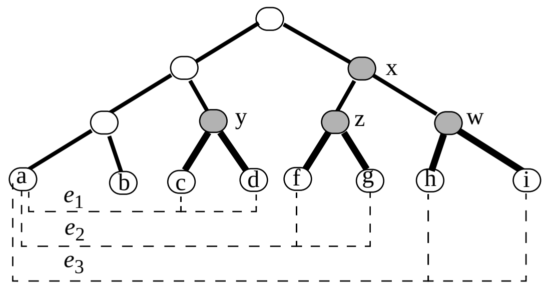

Fig. 6. Destination to which a node is moved.

the tree. The destination vertex may be a leaf or an internal vertex which represents a set of leaves in a subtree. For example, if moving a node v to any of leaves **f**, **g**, **h**, or **i** in Fig. 6 results in maximum gain, we set *dest*(v) to be vertex **x**, which is the root of the smallest subtree containing **f**, **g**, **h**, and **i**. The particular destination leaf of the subtree rooted at **x** to which v is moved is determined when v is selected as the node to move, and this decision depends on the current capacities of the leaves. This method is efficient since we only compute and store a destination vertex and a gain for each node. It is also flexible since we keep the best moves that have the same gain and the tie is resolved dynamically only when a decision is demanded.

The destination vertex and the gain of each node v is computed from the current bindings of all the nets containing v. Given a partition, let *root*(e) be the root of the smallest subtree containing net e, i.e. all the nodes connected to e are assigned to a leaf in the subtree. Then, *root*(e-$\{v\}$) denotes the root of the smallest subtree that contains all the leaves to which other nodes in e excluding node v are assigned. Moving node v to vertex *root*(e-$\{v\}$) will reduce the interconnection cost of net e, if v is not currently assigned to this vertex, (i.e. assigned to a leaf of the subtree rooted at *root*(e-$\{v\}$.) On the other hand, if v is assigned to *root*(e-$\{v\}$), moving v out of *root*(e-$\{v\}$) will create new interconnection cost for net e. Considering all the nets e containing v, the best destination *dest*(v) for node v is the root of the subtree in which most of the vertices *root*(e-$\{v\}$) are located. We use a search operation on all the vertices *root*(e-$\{v\}$) to compute *dest*(v). Once the destination vertex is found, the gain of moving v to *dest*(v) is computed.

For example, suppose a node v currently assigned to leaf **a** in Fig. 6 is connected to three 3-pin nets e_1, e_2 and e_3. Each dashed line in Fig. 6 represents each of the three nets and depicts the leafs to which the nodes of the net are currently assigned. If we consider net e_1, moving v to *root*(e_1-$\{v\}$) = **y** gives a positive gain. Similarly, for net e_2 and net e_3, moving v to *root*(e_2-$\{v\}$)=**z** or *root*(e_3-$\{v\}$)=**w** also reduces the interconnection cost. Since there are two vertices, **z** and **w,** in the right subtree of the root, vertex **x** is chosen as the destination vertex *dest*(v). Neither **z** nor **w** is further searched as the destination because there is a tie in the gain of moving node v to **z** or **w**.

3.3. *Selecting a Node to be Moved*

In the Fiduccia-Mattheyses algorithm [1], bucket arrays are used to store the gains and retrieve the next node to be moved. As mentioned in Section 3.2, one problem in using bucket arrays in hierarchical tree partitioning is that there are too many blocks (leaves) and each one requires a bucket array. Another problem is that the size of the array [-p..p] is large where p is the maximum possible change in the gain of a node. ($p = d_{max} \times c_{max} \times \sum_{0 \le l \le L-1} w(l)$, where d_{max} is the maximum number of nets to which a node connects, c_{max} is the maximum weight of a net, and L is the height of the tree.) Note that p is proportional to the sum of the weighting factors $w(l)$, which can be exponential in the height of the tree in our problem.

Instead of using bucket arrays, we use a sorting tree for selecting the next node to be moved. The sorting tree follows the structure of the tree hierarchy T in partitioning. For each leaf u of T, the node in $N(u)$ (nodes assigned to u) with the biggest gain is stored at the corresponding leaf. These gains are sorted from the bottom level to the top level of the tree. Therefore, the node with the biggest gain is available at the root of the tree. Initially, sorting the gains requires $O(b \log b)$ time where b is the number of leaves in T. For subsequent steps, it takes $O(\log b)$ for updating the sorting tree each time the gain stored in a leaf of T is changed.

Using the same strategy, we also maintain the gains of all the nodes in $N(u)$ for each leaf u in T by a sorting tree T_u. Thus, a similar sorting algorithm can be used to sort and maintain the gains of nodes stored in T_u. The time complexity for each update when the gain of a node in T_u is changed will be $O(\log n/b)$ if the partitioning is balanced so that each T_u will contain approximately the same number of nodes.

3.4. *Selecting a Node to be Replaced*

When selecting a node v to be moved, it is possible that its new location has its capacity being full. One solution is to discard the selection of node v and choose the next best node to move. However, this process of searching a feasible move may take several iterations. In our heuristic, we use a replacement strategy instead of discarding the best (but illegal) move. When moving node v, the capacity of its destination *dest*(v) is checked. If the capacity is not full, node v is moved to a leaf of the subtree rooted at *dest*(v). If the capacity is full, we choose a node v' currently assigned to a leaf of the subtree rooted at *dest*(v) to move next and place v in the location of v'. Selecting node v' is straightforward by utilizing the sorting tree that maintains the gains. We select the node with the biggest gain stored in *dest*(v) in the sorting tree, which is the best node assigned to *dest*(v) to move. Therefore, the sorting tree provides an efficient mechanism for both selecting the next node to be moved and selecting the node to be replaced. Note that the best node and the biggest gain stored in *dest*(v) may have already changed after v is moved. For efficiency, we update the gains approximately before the next node to move in *dest*(v) is retrieved.

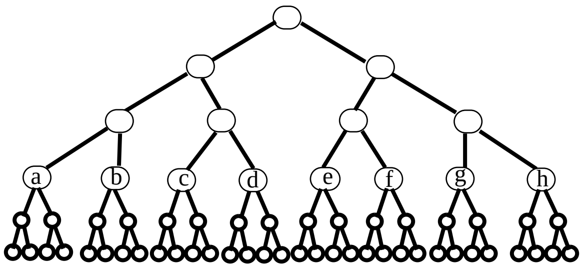

Fig. 7. Extended tree hierarchy *T**.

4. A Multi-Phase Approach for Hierarchical Tree Partitioning

As described in Section 3, we use the given tree structure *T* as a sorting tree to maintain the best gains at the vertices of *T*. The gains of all the nodes assigned to the same leaf of *T* are also maintained by a similar tree structure. In fact, we can use an extended tree hierarchy T^* as shown in Fig. 7 to combine these two tree structures. The leaves in T^* represent nodes in the hypergraph.

In the heuristic outlined in Fig. 5, the local change of the partition only moves or replaces one leaf (i.e. one node in *V*) at the bottom level of T^*. To further explore the solution space by moving a group of nodes at a time, we apply a multi-phase hierarchical tree partitioning approach. A "phase" refers to an execution of the iterative improvement algorithm in Fig. 5, using an extended tree hierarchy T^* to partition a hypergraph *H*. After executing the algorithm, nodes assigned to the same parent vertex are considered as a cluster and are merged to a supernode. Thus, T^* is reduced by one level and the hypergraph *H* is condensed. Given the new tree hierarchy, we then apply the iterative improvement algorithm again on the condensed hypergraph. This process of iterative improvement and merging of the leaves of the tree hierarchy is repeated for several phases until the tree has a small height and the hypergraph has a small number of nodes. After the last phase, we flat the hypergraph and map the partition of the condensed hypergraph to a corresponding partition of the flattened hypergraph.

The effectiveness of the multi-phase approach depends on how nodes are chosen to be merged into a supernode. Those nodes have to form a good cluster, i.e. a highly connected group. Recall that the weighting factor of interconnection cost at each level of the tree grows exponentially from the bottom to the top. With the objective of minimizing the total cost, there is a tendency that highly connected nodes will be assigned to the same subtree at the bottom level. Thus, we merge these nodes with the same parent vertex to supernodes in each phase.

5. Experimental Results

We implemented the multi-phase hierarchical partitioning algorithm (HIPAR) described in this paper and tested it on MCNC benchmarks to show the effectiveness of the algorithm. The multi-phase algorithm was iteratively applied until no further improvement was made. Two experiments using different initial partitions for iterative improvement were conducted. The first experiment used initial partitions generated from multiway partitions produced by the *Gradient decent based Fiduccia-Mattheyses* (*GFM*) algorithm in [3]. The second one used initial partitions generated by a recursive FM (RFM) algorithm. In both experiments, the target tree hierarchy was a full binary tree with height 4 for all test cases. Number of nodes in the test cases ranges from 924 to 2856 and number of nets ranges from 860 to 2824. The size upper bound *B* was calculated as (total node size) $\times$ 110% / 16 since there are 16 leaves in the tree. The program was run on a Sun Sparc20 and the runtime reported in the tables are measured in seconds.

Table 1 shows the results of the HIPAR algorithm using initial partitions produced by GFM. We applied the GFM algorithm to generate a 16-way partition with the objective of minimizing the total number of I/O pins, without considering the cost weighting factors. We then mapped the 16-way partition to an initial partition for the target tree hierarchy by taking the cost weighting factors into account. The initial partition was iteratively improved by the HIPAR algorithm to minimize the total interconnection cost. From Table 1, we can see that the HIPAR algorithm reduced the cost by 31.4% to 42.6%.

Table 1. Results using initials produced by GFM

Circuit	Initial Cost	Final Cost	Improv.	Runtime (s)
c2670	2740	1572	42.6%	529
c3540	5208	3345	35.8%	843
c5315	5412	3198	40.9%	3519
c6288	5326	3437	35.5%	4517
c7552	4268	2926	31.4%	9561

To investigate how the HIPAR algorithm improved the partitioning, we calculate the interconnection cost at each level and compare the cost distribution of the initial partition with that of the final partition. Fig. 8(a) shows the cost distributions of initial partitions produced by the GFM algorithm. Since the GFM algorithm minimized the number of I/O pins at level 0, the initial partitions (mapped from the multiway partitions) of all test cases had most of the cost at higher levels. After applying our HIPAR algorithm, the costs at higher levels were greatly reduced as shown in Fig. 8(b).

Table 2 shows the results of the HIPAR algorithm using initial partitions produced by RFM. We applied the RFM algorithm, which recursively called the two-way FM algorithm, to generate an initial partition for the target tree hierarchy. The objective of each two-way partitioning was to minimize the number of crossing nets. Therefore, it did not take the cost weighting factors and the global cost into

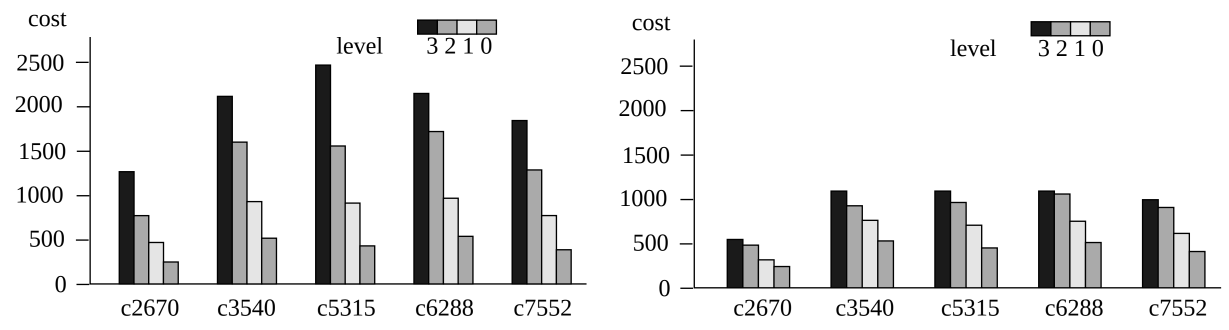

Fig. 8. Interconnection cost distributions of partitions. (a) Initial partitions produced by GFM. (b) Final partitions improved by HIPAR.

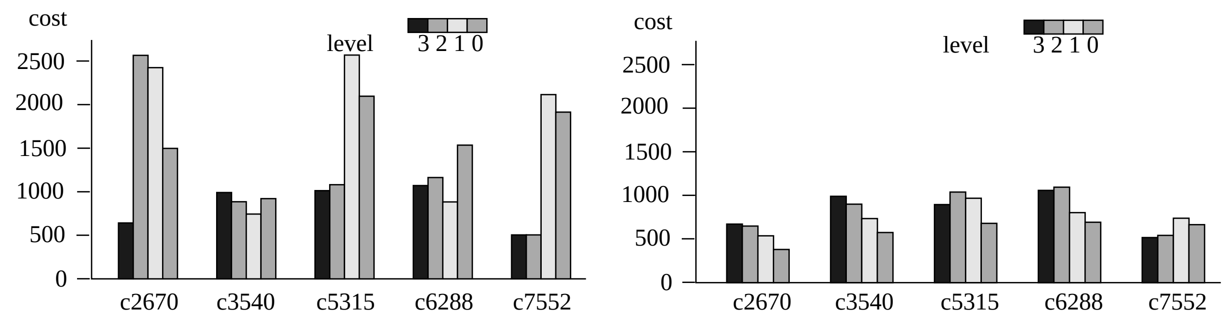

Fig. 9. Interconnection cost distributions of partitions. (a) Initial partitions produced by RFM. (b) Final partitions improved by HIPAR.

TABLE 2. RESULTS USING INITIALS PRODUCED BY RFM

Circuit	Initial Cost	Final Cost	Improv.	Runtime (s)
c2670	7214	2277	68.4%	1394
c3540	3596	3189	11.3%	977
c5315	6827	3555	47.9%	5851
c6288	4626	3676	20.5%	3720
c7552	5084	2461	51.6%	8494

account. Starting with the initial partition, the HIPAR algorithm was applied for iterative improvement to minimize the total interconnection cost. From Table 2, we can see that the HIPAR algorithm reduced the cost by 11.3% to 68.4%.

As in the first experiment, we also compare the cost distributions of the initial partition and the final partition. In Fig. 9(a), the initial partitions produced by the RFM algorithm had lower costs at level 3 since the algorithm partitioned the netlist starting from the top level. However, the partitioning qualities of the subsequent levels 2, 1, and 0 were sacrificed. After applying the HIPAR algorithm, the costs at lower levels were reduced significantly to improve the partition (Fig. 9(b)).

6. CONCLUSIONS

In this paper, we have investigated the partitioning problem of mapping a circuit into a tree hierarchy. Since both recursive top-down partitioning and bottom-up clustering methods optimize the partition one level at a time, they are not able to handle the hierarchical tree partitioning problem which has an objective of minimizing a global cost. Therefore, we proposed a heuristic algorithm for partitioning a circuit into a given tree hierarchy. Experimental results showed that the heuristic algorithm can effectively improve the partitions. In the future, we will extend our work to generic hierarchical tree partitioning where the tree hierarchy is dynamic.

REFERENCES

[1] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improved Network Partitions," *Proceedings of the 19th Design Automation Conference*, 1982, pp. 241-247.

[2] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, New York, Wiley, 1990.

[3] L.-T. Liu, M.-T. Kuo, S.-C. Huang, and C.-K. Cheng, "A Gradient Method on the Initial Partition of Fiduccia-Mettheyses Algorithm," *Proceedings of the International Conference on Computer-Aided Design*, November 1995, pp. 229-234.

[4] L. A. Sanchis, "Multiple-Way Network Partitioning," *IEEE Transactions on Computers*, Vol. 38 No. 1, January 1989, pp. 62-81.

[5] G. Vijayan, "Generalization of Min-Cut Partitioning to Tree Structures and Its Applications," *IEEE Transactions on Computers*, Vol 40, No. 3, March 1991, pp. 307-314.