Design Hierarchy Guided Multilevel Circuit Partitioning[*]

Yongseok Cheon and D. F. Wong
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712

ABSTRACT

In this paper, we present a new multilevel circuit partitioning algorithm (dhml) which is guided by design hierarchy. In addition to flat netlist hypergraph, we use user design hierarchy as a hint for partitioning because it already has some implications on connectivity information between logical blocks in the design. Using design hierarchy in partitioning is nontrivial since hierarchical elements in design hierarchy does not necessarily have strong internal connectivity, hence we need to determine whether it is preferable to break up or preserve the hierarchical elements. In order to identify and select the hierarchical elements with strong connectivity, Rent exponent is used. Then, the selected hierarchical elements are used as effective clustering scopes during multilevel coarsening phase. The scopes are dynamically updated (enlarged) while building up a clustering tree so that the clustering tree resembles the densely connected portions of the design hierarchy.

We tested our algorithm on a set of large industrial designs in which the largest one has 1.8 million cells, 2.8 million nets, and 11 levels of hierarchy. By exploiting design hierarchy, our algorithm produces higher quality partitioning results than the state-of-the-art multilevel partitioner hMetis[7]. Furthermore, experimental results show that dhml yields significantly more stable solutions, which is helpful in practice to reduce the number of runs to obtain the best result.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids; J.6 [**Computer Applications**]: Computer-Aided Engineering—*computer-aided design (CAD)*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Circuit partitioning, design hierarchy, clustering, Rent's rule

*This work was partially supported by the National Science Foundation under grant CCR-9912390.

1. INTRODUCTION

Circuit partitioning is a critical optimization problem in many areas of VLSI design automation because the partitioning solutions have great impacts on automatic placement and routing procedures. The attempts to solve this NP-complete problem have concentrated on finding heuristic algorithms which yield near-optimal solution in polynomial time. Some of the best known approaches include the iterative improvement methods such as Kernighan-Lin (KL), Fiduccia-Mattheyses (FM) algorithms and their variations[5, 9, 1]. Recently, a new multilevel partitioning scheme has been introduced in order to improve partitioning results of iterative improvement approaches especially for bigger designs[2, 7, 8]. The multilevel partitioning is believed to be the most effective approach to produce the best partitioning quality in smaller run time.

Generally a multilevel partitioning consists of 1) multilevel clustering (coarsening), 2) initial partitioning at the coarsest level, and 3) multilevel FM refinement with unclustering (uncoarsening). During the coarsening phase, the problem size is gradually reduced over the levels while capturing strong connectivity in the circuit netlist. Then, the initial partition at the coarsest level is propagated to lower levels, at which FM partitioning is performed to improve the current initial partition which has been inherited from the upper level. At each level, only a small number of passes are needed for FM refinement since the initial partition from upper level already has quite good quality.

In multilevel partitioning, the levels are determined in the coarsening phase while identifying and grouping the strongly connected vertices. Through the successive level-by-level clustering, a multilevel clustering tree C is constructed. The clustering tree C and design hierarchy tree D is similar in that both are the representations of multilevel hierarchical groupings. The proposed work is motivated by that the well-grouped hierarchical elements in D can be used to guide the clustering tree construction. Since the design hierarchy already has some implications on connectivity information between the logical blocks in the design in most cases, it can be beneficial to build up the clustering tree as similar to the design hierarchy as possible. However, we do not blindly follow every grouping in D, rather we identify and select some good hierarchical elements (i.e., the hierarchical elements with higher internal connectivity) to use them as clustering scopes. Rent exponent is used as a quality indicator to find the good hierarchical elements, which are called positive scopes. After completion of each one-level clustering, a clustering scope is updated to a larger scope if clustering process in the scope turns out to be *saturated* so that the vertices in the current scope now have chances to be merged with others in the larger scope at the next level clustering.

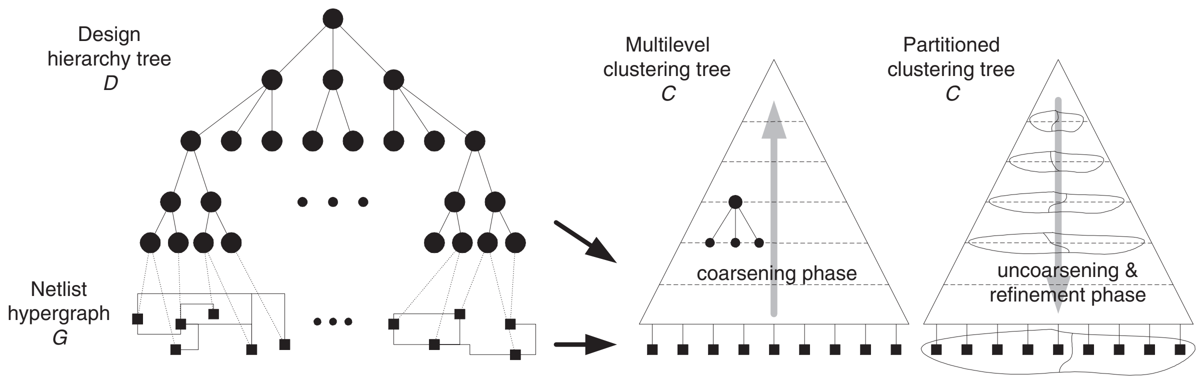

Figure 1: Design guided multilevel circuit partitioning problem.

By this scope restriction, we expect entire clustering phase to produce a clustering tree which is biased to preserve the well-grouped logical blocks in the design hierarchy.

For FPGA applications, a few partitioning methods utilizing design hierarchy have been reported recently[10, 3, 4]. They mainly focus on problem size reduction using design-based clustering. The hierarchical elements are selectively preserved if they are feasible — both size and pin count are smaller than the limit of each FPGA device. For non-feasible hierarchical elements, some operations are applied to intelligently break up the elements. Under the size and pin count constraints, usually their goal is to find a set of the good feasible blocks which maximizes device utilization, i.e., minimizes the number of FPGA's used. However, their frameworks are not directly applicable to general partitioning problems, and they may preserve the hierarchical elements with loosely connected internal cells unless the external pin count exceeds the upper bound. Moreover, they are not easy to be transformed into the multilevel scheme.

This paper proposes a new multilevel circuit partitioning, `dhml`, that benefits from design hierarchy. It takes a user design hierarchy as well as a netlist hypergraph, and builds up a clustering tree that resembles the design hierarchy (See Figure 1). With this guidance in multilevel clustering phase, experimental results show that `dhml` yields higher quality solutions than the conventional multilevel partitioner. Speed-up has been also achieved in a sense that near-optimal solutions are more frequently obtained in multiple runs since `dhml`'s partitioning solutions are more stable. Aggressively reduced number of levels in clustering phase also contributes the speedup.

2. PROBLEM FORMULATION

Definition 1. A circuit is modelled by a network of leaf cells represented by *hypergraph* $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of *leaf cells* and $\mathcal{E}$ is a set of *hyperedges (nets)*. A set of leaf cells which is a subset of $\mathcal{V}$ is defined as a *cluster*.

Definition 2. A design hierarchy provided by designer is represented by a rooted tree. A *design hierarchy tree* D is a collection of nodes and arcs such that a node is either a leaf cell or a hierarchical element which contains other nodes as children.

Notation 1. For a given netlist hypergraph $G = (\mathcal{V}, \mathcal{E})$ and design hierarchy D,
1) $S(v) \equiv$ the size of a leaf cell v.
2) $S(H) = \sum_{v \in H} S(v) \equiv$ the sum of sizes of leaf cells contained in a cluster H. $S(D) = \sum_{v \in \mathcal{V}} S(v) \equiv S_{total}$.
3) $|H| \equiv$ the number of leaf cells in a cluster H.
4) $E(v) \equiv$ the pin count of a leaf cell v.
5) $E(H) \equiv$ the external pin count of a cluster H.

Problem 1. Given a design hierarchy D and $G = (\mathcal{V}, \mathcal{E})$, the partitioning problem is to partition $\mathcal{V}$ into k disjoint subsets $V_1, \cdots, V_k$, with the objective of minimizing $\sum_{i=1}^{k} E(V_i)$. If $k = 2$, we call the problem bipartitioning.

3. DESIGN HIERARCHY

Figure 1 shows an example of design hierarchy. We use Rent exponent as a quality indicator to determine which hierarchical element has a strong internal connectivity.

3.1 Rent's Rule and Rent Exponent

Rent's rule is an empirical formula which describes the general relationship between the number of cells and the number of external nets in a subcircuit (or cluster).

$$E = \bar{P} \cdot B^r \tag{1}$$

where r is the Rent exponent or Rent parameter with $r \leq 1$, E is the number of external nets of a cluster, $\bar{P}$ is the average number of pins per cell, and B is the number of cells in the cluster.

Rent's rule has been widely used for interconnection complexity estimation. Hagen et al.[6] defined *intrinsic* Rent exponent to characterize the quality of a partitioning algorithm. The intrinsic Rent exponent of a given partitioning tree is the representative value of the quality measure for the corresponding partitioning algorithm. Also, there have been a few clustering algorithms to identify strongly connected cells by using Rent exponent as a projected quality measure[12, 11]. In the Rent's rule based clustering algorithm, the locality of Rent exponent is more emphasized to select the best merging combinations from candidate neighbors.

Our approach is inspired by the combination of global and local connectivity information. Since a design hierarchy

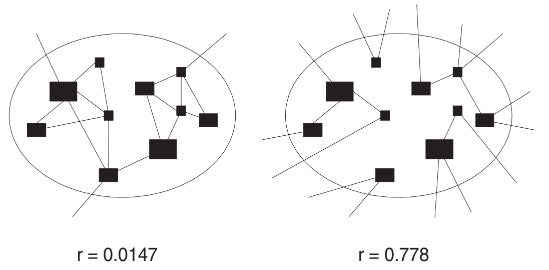

Figure 2: Implication of local Rent exponent.

tree is given, we can estimate the global quality of the tree by computing a representative value of Rent exponent, $\bar{r}$. Contrary to the partitioning trees, design hierarchy trees usually do not have regular patterns in sizes and number of nodes. Hence, Rent exponent extraction method in [6] may not be feasible because data points gathering for linear regression is not totally controllable. Hence we have used the average value of all Rent exponent of the hierarchical clusters in D weighted by size. The representative value obtained from the above is not so useful unless it is combined with local measure. As used in [12, 11], local Rent exponent is beneficial to exploit local connectivity information.

The equation (1) can be rewritten for a hierarchical element H as follows.

$$E(H) = \bar{P}_H \cdot |H|^r \tag{2}$$

Let $P(H)$ be the total number of pins of cells in H, i.e., $P(H) = \sum_{v \in H} E(v)$. Also, let $I(H)$ be the total number of internal pins in H, i.e., $I(H) = P(H) - E(H)$. Since $\bar{P}_H = P(H)/|H| = (I(H) + E(H))/|H|$, from equation (2),

$$\begin{aligned} r(H) &= \frac{\ln E(H) - \ln((I(H)+E(H))/|H|)}{\ln |H|} && (3) \\ &= \frac{\ln(E(H)/(I(H)+E(H)))}{\ln |H|} + 1 && (4) \end{aligned}$$

From the equation (4), we note that Rent exponent also captures some information on the internal connectivity. It is obvious that if all pins are contributed to external nets, i.e., $E(H) = P(H)$, $r(H) = 1$ which is the maximum value. From the viewpoint of internal connectivity, small Rent exponent implies relatively high connectivity inside and large Rent exponent implies low connectivity. In [10], a similar measure — S/T quality (ratio of size to external pin count) — was used to estimate the connectivity quality of hierarchical elements. However, it does not capture the internal connectivity, which is more helpful to identify the hierarchical elements that have more strongly connected cells inside.

As shown in Figure 2, a hierarchical element H with small r implies that it contains relatively more strongly connected cells inside. Thus it is preferable to preserve the internal connectivity. On the other hand, a hierarchical element H with large r implies that it has relatively more connections with outside cells, which means it is preferable to remove the grouping by H so that the cells in H can have chances to be chosen as strongly connected neighbors from outside of H.

For a hierarchical element H, the weighted average of Rent exponents, $\bar{r}$ is used as a threshold value to determine whether the corresponding Rent exponent $r(H)$ is small or

```
Procedure construct_cluster_tree
Input: bottommost netlist hypergraph $G = (\mathcal{V}, \mathcal{E})$
       $h$-level design hierarchy tree $D$
Output: $k$-level clustering tree $C$
1. Extract scope tree $D'$ from $D$
2. for each leaf cell $v \in \mathcal{V}$ do
3.     Determine a clustering scope $H(v)$ in $D'$
4. $G_0(\mathcal{V}_0, \mathcal{E}_0) = G(\mathcal{V}, \mathcal{E})$,  $k = 0$
5. do
6.     $G_{k+1}$ = cluster_one_level($G_k$)
7.     $k = k + 1$
8. while $|\mathcal{V}_k| > \alpha$ and $|\mathcal{V}_k|/|\mathcal{V}_{k-1}| < \beta$
```

Figure 3: Clustering tree construction procedure.

large. A hierarchical element H is said to be a *positive* scope if $r(H) < \bar{r}$, a *negative* scope otherwise. With the guidance of preliminary knowledge of connectivity information from design hierarchy, multilevel clustering is performed while restricting the clustering scopes to good hierarchical groupings — positive scopes.

4. MULTILEVEL PARTITIONING

In this section, we provide a multilevel clustering algorithm which is guided by the Rent exponents which imply the local connectivity quality of the design hierarchy. Then, the entire partitioning algorithm, `dhml`, is presented.

4.1 Design Hierarchy Guided Clustering

As shown in Figure 1, coarsening phase of the multilevel partitioning consists of successive bottom-up clustering procedures from a set of leaf cells. During the coarsening phase, large nets are contracted to smaller nets and a sequence of successively smaller hypergraphs are constructed. Thus, several vertices at the current level merge to form a bigger vertex at upper level, eventually forming a k-level tree C.

The main purpose of multilevel clustering is to create a small hypergraph such that a good bisection of the small hypergraph is not significantly worse than the bisection directly obtained from the original hypergraph[7]. This quality preservation ensures that the top-down refinement later does not need much effort to improve the initial partition inherited from the upper levels. Although best initial partition at the coarsest level does not always guarantee the best partition at the finest level with leaf cells, there are more chances to reach higher quality final solution if we build up a higher quality clustering tree. Fortunately we have an additional grouping information in user design hierarchy tree D, which is originally based on functional decomposition. Design hierarchy D and Rent exponents of hierarchical elements in D give a guidance for the multilevel clustering procedure.

Clustering is defined as a merging process on the existing vertices to form smaller number of bigger vertices. In multilevel partitioning schemes published, pairwise merging has been widely used[7, 8, 2]. We have performed extensive experiments with various clustering methods, and FC(First Choice) coarsening proposed in [8] turned out to be the most effective. Thus, we are using a connectivity cost and merging policy similar to FC in `hMetis`. Even though we are using the same idea as the one in `hMetis` to form bigger ver-

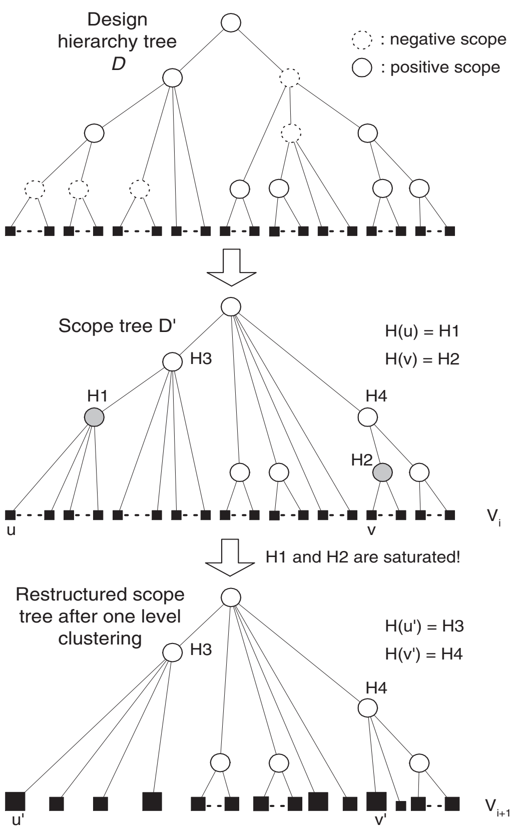

Figure 4: Design hierarchy restructuring.

tices, entire clustering scheduling is quite different because of the existence of guidance from the design hierarchy D.

Figure 3 summarizes our multilevel clustering tree construction procedure. As an initialization process, the original design hierarchy tree D is restructured such that all the negative scopes are removed from D (See Figure 4). The resulting tree only with the hierarchical elements which are positive scopes are called *scope tree*. In general, the vertices in a negative scope have more freedom to be merged with others outside of the scope. Meanwhile, the vertices in the positive scope are restricted to be merged with those within the scope. Next, for each leaf cell v, we determine a *clustering scope* in D' (Step 2, 3). Initially the immediate parent of v in D' is chosen as a clustering scope of v, and denoted by $H(v)$. If there is no positive scope for v in D, the root of D is assigned to $H(v)$, which means that scope restriction is not used for such v.

Then, starting from the bottommost netlist hypergraph, we successively construct the upper level hypergraph until the number of vertices is reduced enough or clustering process is saturated (Step 4–8). In step 8, α and β represents a desirable size at the coarsest level and a threshold value of slow reduction rate respectively. In our experiment, $\alpha = 100$ and $\beta = 0.9$ have been used.

The main clustering procedure is presented in Figure 5. First, we arrange the vertices in $\mathcal{V}_i$ so that cluster v cannot be matched before u if $H(u) \subset H(v)$. If the vertices have clustering scopes with level(height) k in D, then the vertices

```
Procedure cluster_one_level
Input: hypergraph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$
Output: contracted hypergraph $G_{i+1}$
1.  Compute $\{\mathcal{W}_k \mid \mathcal{W}_k = \{v \mid v \in \mathcal{V}_i, H(v)$'s level is $k\}\}$
2.  for $k = 1$ to $h$ do
3.      while there exist unmatched vertices in $\mathcal{W}_k$ do
4.          Randomly select an unmatched $v$ in $\mathcal{W}_k$
5.          Find $w \in \mathcal{V}_i$ that maximizes clus_cost($v$, $w$)
                such that $H(w) \subseteq H(v)$
6.          if there is no such $w$
7.              Create a singleton cluster $v'$ in $\mathcal{V}_{i+1}$
8.          else if $w$ is unmatched
9.              Create a cluster $v'$ in $\mathcal{V}_{i+1}$ with $(v, w)$
10.         else if $w$ is matched in existing $v' \in \mathcal{V}_{i+1}$
11.             Append $v$ to $v'$
12.         $H(v') = H(v)$
13. Remove saturated hierarchical elements in $D'$
        and update $H(v')$ accordingly
```

Figure 5: Main clustering procedure.

are grouped together in $\mathcal{W}_k$ (Step 1).

Next, matchings are performed from the lowest level $\mathcal{W}_1$ at which the scopes are the smallest. For each level of the scope, we randomly select a cluster v which has not yet been matched to contribute one of the upper level vertices. For the cluster v, we consider the neighbors which are connected to v by some nets in $\mathcal{V}_i$ and are located inside of $H(v)$. We have an upper bound of size μ so that the resulting upper level vertex should be smaller than μ. In step 5, with the restricted scope and the size upper bound, the best neighbor w is chosen that maximizes a `clus_cost`(v, w) defined as $\sum_{e \in \{e \mid v \in e, w \in e\}} 1/(|e| - 1)$ which is adopted from [7]. Using the restricted scope, only potentially good neighbors in $H(v)$ are searched to be merged with v. If the best neighbor w is not yet contributed to form any upper level vertices, v is merged with w and a new vertex $v' \in \mathcal{V}_{i+1}$ is created (Step 8–9). If the best neighbor w has been already matched with others and contributed to form an upper level vertex v', v is appended to v' which w belongs to (Step 10–11). In case that there is no such best neighbor for v, a new singleton vertex v' only with v is created (Step 6–7).

Like FC coarsening in [7], only-unmatched-neighbor condition is relaxed, and even the matched neighbors are also considered unless the size of the resulting upper cluster violates the maximum allowable size, μ. In our experiments, μ has been set to $b \times S_{total}$, where b is the balance ratio parameter for a partitioning problem. If both matched and unmatched vertices are considered as neighbors, the number of vertices in successive coarse hypergraphs may decrease by a large factor, potentially limiting the effect of multilevel refinement. Hence, usually the reduction rate is controlled by a certain constant, e.g. 1.7 to ensure sufficiently many levels in a clustering tree for refinement procedure to effectively improve the quality. However, we have removed the reduction rate control to take full advantage of the guidance of design hierarchy, and matching procedure is performed until every lower level vertices is contributed to an upper level cluster. As a result, the number of levels is usually significantly less than that of the conventional multilevel partitioning. Nevertheless, our experimental results show that it

```
Algorithm dhml
Input: netlist hypergraph $G = (\mathcal{V}, \mathcal{E})$
       design hierarchy tree $D$
Output: bipartition $P_0 = \{V_1, V_2\}$
1. Perform Rent exponent computation on $D$
2. construct_cluster_tree($G$, $D$)
3. for $i = 1$ to $max$ do
4.     Generate a random initial bipartition $R$ on $G_k$
5.     $Q_i$ = FMbipartition($G_k$, $R$)
6. $P_k$ = best bipartition among $Q_i$'s
7. for $i = k - 1$ down to 0 do
8.     $P_i$ = FMbipartition($G_{i+1}$, $P_{i+1}$)
```

Figure 6: The `dhml` multilevel partitioning.

does not degrade the quality since our clustering procedure builds up a clustering tree with better connectivity quality in spite of having fewer levels due to the correct guidance from design hierarchy.

The clustering scope of v', $H(v')$ must be updated for next round clustering on $\mathcal{V}_{i+1}$. It is obvious that $H(v')$ is set to $H(v)$ for a new vertex v' created in step 7 and 9 since $H(w) \subseteq H(v)$. Due to the initial ordering by scope level in step 1, $H(v') \subseteq H(v)$ is also guaranteed, so we choose $H(v)$ for the new clustering scope of v' (Step 12).

After the completion of one-level clustering, the existing clustering scopes are examined and updated for next level in step 13. As the global saturation condition is examined in `construct_cluster_tree`, for each clustering scope, local saturation condition is checked and the scope is removed to enlarge the scope if necessary as shown in Figure 4. Let $c_k(X)$ be the set of vertices which belong to X at level k. A clustering scope X is said to be *saturated* at level k and removed if either $|c_k(X)| \leq \alpha(X)$ or $|c_k(X)|/|c_{k-1}(X)| \geq \beta(X)$ (in our experiments, $\alpha(X) = \lceil S(X)/\mu \rceil$ and $\beta(X) = 0.7$). Consequently for every cluster v in $c_k(X)$, $H(v)$ is updated with the parent of X in the scope tree D', which is larger but still is a good grouping in D. Whenever one level clustering is done, we check and remove the local saturation in this manner until global saturation condition holds.

4.2 Design Hierarchy Guided Multilevel Partitioning

Figure 6 describes `dhml`, our multilevel bipartitioning algorithm using design hierarchy guided clustering. As a first step, the algorithm computes Rent exponent for each hierarchical element. Then, as shown in the previous section, design hierarchy tree D is used to guide multilevel clustering. After completion of clustering, we have a k-level clustering tree. At the coarsest level k, FM bipartition is applied for max times with random initial bipartition, and the best cut quality partition is chosen to be propagated down to successive lower levels as the initial partition (In our experiments, $max = 20$). The initial partitioning phase (step 3–5) is very fast since there are only a few vertices to be partitioned at the coarsest level. In step 7 and 8, uncoarsening and FM refinement is performed with the current best partition from upper levels as the initial partition. The number of passes to improve the current partition is only one or two in most cases because the initial partition from the coarser levels has already good quality.

We also can extend the algorithm `dhml` to multi-way partitioning by applying this bipartitioning recursively. For each bipartitioning, a partial design hierarchy tree is extracted from the original entire design hierarchy tree D, in which the leaf cells are the cells that belong to the current partition. This partial tree is used to guide the sub-partitioning in the same way described previously.

5. EXPERIMENTAL RESULTS

We implemented our algorithm in C++/STL and evaluated the performance on six large scale industrial circuits, which are real circuits used in industries.[1] The characteristics of design hierarchies and netlist hypergraphs for these circuits are shown in Table 1, where the largest circuit `ind6` has about 1.8 million cells, 2.8 million nets, and 11 levels of hierarchy. The fourth column shows the height of each design hierarchy tree and the number of hierarchical elements. We first describe the stability of our algorithm, and then the cut quality is shown as the number of partitions varies.

Circuit	No. cells	No. nets	h/No. hier. nodes
ind1	15186	19152	6/302
ind2	136340	183340	9/10427
ind3	224908	187595	5/57590
ind4	414633	414013	13/94796
ind5	1213105	1317889	13/33277
ind6	1841147	2788461	11/35449

Table 1: The characteristics of the circuits.

As pointed out in [1], a common weakness of partitioning methods based on iterative improvement is that the solution quality is *not stable*. Conventional multilevel partitioners cannot be ruled out of this unstableness since they also use move-based approaches which depend on the initial solutions. However our algorithm which is guided by design hierarchy shows more stable solution ranges while having better minimum solution as shown in Figure 7. That implies `dhml` will have more chances to achieve near-optimal solutions in smaller number of runs. Also, the average solution is very close to the minimum solution, which is useful in real CAD tools because hundreds of runs cannot be performed in the practical CAD tools.

Table 2 summarizes the cut set size comparison of `dhml` and `hMetis`.[2] For fair comparison, `hMetis` results are based on FC coarsening and FM refinement options. As shown in the table, `dhml` produces up to 16% improved results in terms of the minimum cut set sizes in half runs of `hMetis`. The CPU time per run is about the same as that of `hMetis` even though we have some additional steps to use design hierarchy information since clustering scope restriction and level reduction help to reduce the runtime. In the case of `ind1`, the minimum cut set size of 64 was obtained in every run of `dhml`. Moreover, due to the design hierarchy guided

[1]Note that we cannot use the standard partitioning benchmark circuits from MCNC and ISPD-98 since the design hierarchy information are not given. Moreover, the industrial circuits we use are significantly larger than the standard benchmark circuits.

[2]Note that 256-way partitioning is meaningful since systems with more than 200 FPGA's are commercially available these days.

Circuit	2-way		16-way		256-way	
	`dhml`(5 runs)	`hMetis`(10 runs)	`dhml`(5 runs)	`hMetis`(10 runs)	`dhml` (5 runs)	`hMetis`(10 runs)
ind1	64	69	437	483	–	–
ind2	133	134	1203	1294	14633	16137
ind3	292	305	1454	1551	7450	7508
ind4	202	208	3394	3498	12013	13999
ind5	1376	1352	7410	7950	22474	24454
ind6	55	56	8275	8265	33472	35075

Table 2: Minimum cut set size comparison of `dhml` *vs.* `hMetis` with 5% balance ratio at each bipartitioning.

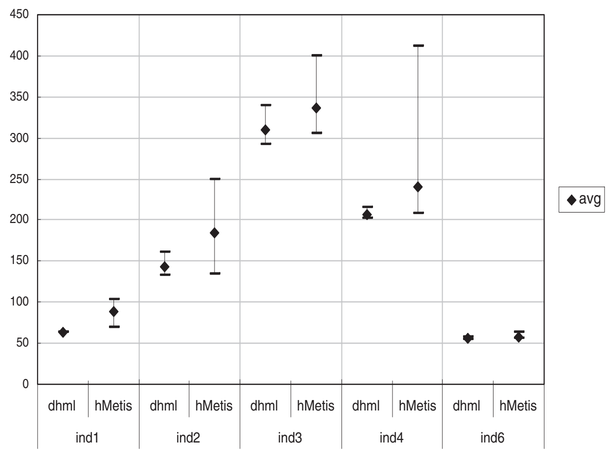

Figure 7: The ranges of 2-way partitioning solutions(cut set size).

clustering, all the minimum cuts have been achieved at the coarsest level (i.e., no further improvement was needed in refinement phase), where problem size has been reduced to 1/150. In most cases, the initial partitions at the coarsest level show 20%–50% better qualities than `hMetis`, which justifies superior quality of the multilevel clustering tree guided by design hierarchy. Also, the number of levels in `dhml` is reduced to 55%–75% of that in `hMetis` while the number of passes for FM refinement at each level is not increased.

6. CONCLUSIONS

A new multilevel partitioning framework that takes advantage of user design hierarchy has been presented. As a guidance of design hierarchy, clustering scope restriction is used to construct a multilevel clustering tree. The clustering scopes are selectively determined by Rent exponent computation and updated dynamically while the clustering tree being built up. Due to the benefit from the guidance by the design hierarchy which has implications on connectivity between functional blocks, the proposed algorithm generates better multilevel clustering tree while the number of levels is aggressively reduced. Our experiments on large scale real circuits show that `dhml` yields more stable and higher quality partitioning solutions in smaller runs than `hMetis` does.

7. ACKNOWLEDGMENTS

We would like to thank Axis systems Inc. for their valuable support. We also thank Chwen-Cher Chang, Youlin Liao, Richard Sun, and Ren-Song Tsay for helpful discussions.

8. REFERENCES

[1] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: a survey. *Integration, the VLSI Journal*, pages 1–81, 1995.

[2] C. Alpert, J. Huang, and A. B. Kahng. Multilevel circuit partitioning. *IEEE Trans. on Computer-Aided Design*, vol. 17, no. 8, pages 655–667, 1998.

[3] D. Behrens, K. Harbich, and E. Barke. Hierarchical partitioning. In *Proc. IEEE Int'l Conf. Computer-Aided Design*, pages 470–477, 1996.

[4] W. Fang and A. C.-H. Wu. Multi-way FPGA partitioning by fully exploiting design hierarchy. *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, no. 1, pages 34–50, 2000.

[5] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proc. ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[6] L. Hagen, A. B. Kahng, F. J. Kurdahi, and C. Ramachandran. On the intrinsic Rent parameter and spectra-based partitioning methodologies. *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 1, pages 27–37, 1994.

[7] G. Karypis, R. Aggarwal, V. Kumar, and S. Sheckhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. ACM/IEEE Design Automation Conference*, pages 526–529, 1997.

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

[9] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning of electrical circuits. *Bell System Technical Journal*, vol. 49, no. 2, pages 291–307, 1970.

[10] H. Krupnova, A. Abbara, and G. Saucier. A hierarchy-driven FPGA partitioning method. In *Proc. ACM/IEEE Design Automation Conference*, pages 522–525, 1997.

[11] C. Liang and C. Ho. A new optimization driven clustering algorithm for large circuits. In *Proc. European Design Automation Conf.*, pages 28–32, 1993.

[12] T. K. Ng, J. Oldfield, and V. Pitchumani. Improvements of a mincut partition algorithm. In *Proc. Int'l. Conf. on Computer-Aided Design*, pages 470–473, 1987.