Edge Separability-Based Circuit Clustering With Application to Multilevel Circuit Partitioning

Jason Cong, *Fellow, IEEE,* and Sung Kyu Lim, *Member, IEEE*

***Abstract*—In this paper, we propose a new efficient $O(n \log n)$ connectivity-based bottom-up clustering algorithm called edge separability-based clustering (ESC). Unlike existing bottom-up algorithms that are based on local connectivity information of the netlist, ESC exploits more global connectivity information using edge separability to guide the clustering process, while carefully monitoring cluster area balance. Exact computation of the edge separability $\lambda(e)$ for a given edge $e = (x, y)$ in an edge-weighted undirected graph G is equivalent to finding the maximum flow between x and y. Since the currently best known time bounds for solving the maximum flow problem is $O(mn \log(n^2/m))$, due to Goldberg and Tarjan (Goldberg and Tarjan, 1988), the computation of $\lambda(e)$ for all edges in G requires $O(m^2 n \log(n^2/m))$ time. However, we show that a simple and efficient algorithm CAPFOREST (Nagamochi and Ibaraki, 1992) can be used to provide a good approximation of edge separability (within 9.1% empirical error bound) for all edges in G without using any network flow computation in $O(n \log n)$ time. Our experimental results based on large-scale benchmark circuits demonstrate the effectiveness of using edge separability in the context of multilevel partitioning framework for cutsize minimization. We observe that exploiting edge separability yields better quality partitioning solution compared to existing clustering algorithms (Sun and Sechen, 1993), (Cong and Smith, 1993), (Huang and Kahng, 1995), (Ng *et al.*, 1987), (Wei and Cheng, 1991), (Shin and Kim, 1993), (Schuler and Ulrich, 1972), (Karypis *et al.*, 1997), that rely on local connectivity information. In addition, our ESC-based iterative improvement based multilevel partitioning algorithm LR/ESC-PM provides comparable results to state-of-the-art hMetis package (Karypis *et al.*, 1997), (Karypis and Kumar, 1999).**

***Index Terms*—Clustering, edge separability, multilevel partitioning.**

I. Introduction

DUE TO substantial advances in very large scale integrated (VLSI) technology, designers are facing a rapid increase in system complexity. One natural approach to designing highly complex systems is to decompose the large system into a set of smaller subsystems recursively and carry out the design hierarchically. In the last ten years, hierarchical algorithms have been applied with dramatic results to several important areas in VLSI computer-aided design (CAD). Concurrent with the steady advances in VLSI design, hierarchical methods for scientific computation have also emerged as the only viable class of scalable solutions for mathematical problems in the gigascale range. These so-called *multilevel methods* model problems across many levels of resolution and efficiently manage local and global communication within and between levels. Typically, they converge in the optimal time order to solutions equal or superior to those obtained by non-hierarchical means. They have had enormous impact in many fields. General examples include wavelets in signal and image processing, domain decomposition methods in computational fluid dynamics, multigrid in numerical PDE simulation, and fast multipole methods in large-scale particle simulations. The multilevel approach has been successfully applied to several areas of VLSI CAD, including circuit partitioning in `hMetis` package [10], circuit placement in `mPL` algorithm [12], and parasitic extraction in `FASTCAP` package [13].[1]

In order to design a multilevel algorithm for a particular class of problems, one must decide how to: i) improve an existing solution at a given level; ii) aggregate information at finer levels into information at coarser levels; iii) interpolate information at coarser levels into information at finer levels; and iv) solve the problem at the coarsest level of representation as accurately as possible. The circuit clustering method tries to identify closely connected components from the given netlist. The clustering result is then used to derive a smaller netlist by grouping nodes in the same cluster together. This clustering process can be recursively applied to the given netlist, and the corresponding multilevel representation of the netlist can be constructed to capture the natural hierarchy from the given circuit. Thus, an efficient circuit clustering method is indispensable to design a multilevel algorithm for several important areas in VLSI CAD.

Most clustering heuristics are bottom-up in nature; each cell initially belongs to its own cluster, and clusters are gradually grown into larger clusters from merging with others.[2] If the clustering is applied once, we establish *two-level* cluster hierarchy. In general, we can apply clustering repeatedly to obtain *multilevel* cluster hierarchy (we assume two-level unless otherwise specified).[3] Depending on the objective,

Manuscript received October 9, 2002; revised May 8, 2003. This research was supported in part by the MARCO/DARPA Gigascale Silicon Research Center (GSRC), in part by Fujitsu Laboratories of America under the California MICRO program, in part by the NSF/Georgia Tech Packaging Research Center, and in part by Georgia Yamacraw. This paper was recommended by Associate Editor T. Yoshimura.

J. Cong is with the Department of Computer Science, University of California, Los Angeles, CA 90095 USA.

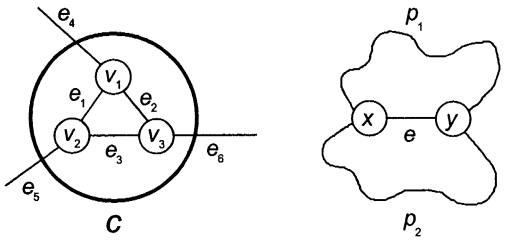
S. K. Lim is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA (e-mail: limsk@ece.gatech.edu).

Digital Object Identifier 10.1109/TCAD.2004.823353

[1]We note that the hierarchical clustering and min-cut exchange (`HCME`) method [14] is the first work that introduced multilevel partitioning into VLSI placement.

[2]We note that the well-known ratio-cut method [7] is a top-down clustering method. In [7], the given circuit is recursively partitioned into clusters, while minimizing the ratio-cut objective. This method can exploit more global information from the circuit, but usually at the cost of large computation time.

[3]Multilevel approaches are relatively new, and most of the existing clustering works are developed in two-level framework. Thus, the investigation on the possible extension of existing two-level works into multilevel framework is much needed.



$$den(C) = \frac{w(e_1)+w(e_2)+w(e_3)}{s(v_1)+s(v_2)+s(v_3)} \qquad w(e) \le \lambda(e).$$

$$rcut(C) = \frac{w(e_4)+w(e_5)+w(e_6)}{s(v_1)+s(v_2)+s(v_3)}$$

(a) (b)

Fig. 1. (a) Two local connectivity ($= w(e)$)-based clustering metrics density [4], [5] (maximize) and ratio cut [7] (minimize) of C. (b) $w(e) \le \lambda(e)$ due to additional paths p_1 and p_2 that connect u and v, where $\lambda(e)$ denotes the edge separability of e.

clustering algorithms can be classified into cutsize- and performance-driven methods. Cutsize-driven clustering methods can be further grouped into connectivity-based approaches [3]–[6], [8], [9], and signal-flow-based approaches [15], [16]. Performance-driven clustering methods can be further grouped into ones that are designed for combinational circuits [17]–[20], and for sequential circuits with retiming [21]–[23]. Some of the recent partitioning works that exploit multilevel cluster hierarchy include [4], [10], [11], and [23]–[29]. A survey of various clustering and partitioning algorithms up to 1995 can be found in [30].

In order to identify closely connected components in the given circuit, the connectivity information among cells in the given circuit plays an important role in cutsize-driven partitioning. Therefore, connectivity-based approaches and their multilevel extension have drawn a lot of attention recently.[4] In spite of the efficiency from its simple nature, however, existing connectivity-based approaches suffer from a limitation—the clustering process is guided by local connectivity-based greedy merging. In other words, the neighboring node to be clustered together is chosen solely based on the weight of the edges that connect the candidate nodes. This locality in clustering decision may lead to suboptimal decomposition of the circuit. For a given edge $e = (x, y)$ in an undirected graph $G(V, E, s, w)$, with vertex size function s and edge weight function w, Fig. 1(a) shows two well-known clustering methods that use edge weight $w(e)$—Density [4], [5] (for maximization) and ratio cut [7] (for minimization). In both methods, one can see that the edge weight $w(e)$ serves as the main criteria for the clustering process. However, the edge weight $w(e)$ does not estimate how tightly x and y are connected in G. In fact, there may exist additional paths that connect x and y other than edge e itself as illustrated in Fig. 1(b). In order to compute the connectivity between x and y after considering *all* paths, we need to compute the $x-y$ *mincut*, the minimum cutsize among all the cuts that separate x and y in G.

[4]This paper deals with the conventional cutsize objective, and our edge separability based clustering (ESC) algorithm belongs to the cutsize-driven-based multilevel approach.

If x and y are connected via an edge, i.e., $e = (x, y) \in E$, we define the *edge separability of* e to be the value of the $x - y$ mincut and denote it $\lambda(e)$ as shown in Fig. 1(b). Clearly, edge separability provides more global connectivity information between x and y compared to local edge connectivity $w(e)$. However, the best-known complexity for exact computation of $\lambda(e)$ is $O(mn\log(n^2/m))$ using the maximum flow algorithm by Goldberg and Tarjan [1]. Therefore, the computation of $\lambda(e)$ for all edges in G requires $O(m^2 n\log(n^2/m))$ time. Furthermore, if we want to exploit the $\lambda(e)$ to guide bottom-up clustering process, we not only have to compute $\lambda(e)$ of all edges in G, but also need proper update of $\lambda(e)$ after merging of x and y since G will be modified. This is extremely time consuming, even for moderate-size graphs with a few thousand vertices. However, we show that a simple and efficient $O(|V|\log|V|)$ time algorithm, named `CAPFOREST` [2], can be used to provide a good estimation of $\lambda(e)$ for all edges in G without using any flow computation.

Our experimental results based on large-scale benchmark circuits demonstrate the effectiveness of using edge separability in the context of multilevel partitioning framework for cutsize minimization. First, we observe that exploiting edge separability yields better quality partitioning solution compared to existing clustering algorithms proposed in the literature including absorption [3], density [4], [5], rent parameter [6], ratio cut [7], closeness [8], connectivity [9], and first choice [10] methods. Second, our `ESC`-based iterative-improvement-based multilevel partitioning algorithm $\texttt{LR/ESC}-\texttt{PM}$ provides comparable results to state-of-the-art `hMetis` [10] and $\texttt{hMetis}-\texttt{Kway}$ [11].

The remainder of the paper is organized as follows. Section II presents the problem formulation for K-way partitioning. Section III presents theoretical backgrounds on the edge separability. Section IV presents our `ESC` algorithm. Section V provides experimental results. Section VI concludes the paper.

II. Problem Formulation

Given a gate-level circuit (combinational or sequential) netlist $NL(C, N)$, let $C = \{c_1, c_2, \cdots, c_n\}$ denote cells that represent the basic elements in NL such as simple gates or flip-flops (FFs), and $N = \{n_1, n_2, \cdots, n_m\}$ denote nets that specify connection among cells in NL. We model NL with a hypergraph $H(V, E_H)$, where the vertex set $V = \{v_1, v_2, \cdots, v_n\}$ represents cells, and the hyperedge set $E_H = \{e_1, e_2, \cdots, e_m\}$ represents nets in NL. Each hyperedge $e \in E_H$ is a nonempty subset of V and has 1-to-1 correspondence to nets in NL. Each vertex $v \in V$ is associated with area $a(v)$, and each hyperedge $e \in E_H$ is associated with a cost $c(e)$. Under the cost-1 metric, $c(e) = 1$ if e spans more than 1 block, i.e., e contains cells that are partitioned into more than 1 block, and 0 otherwise. Under the sum of external degrees (SOED) metric, $c(e) = k$, if e spans k blocks and $k > 1$. From a different perspective, SOED is the sum of out-going nets from each block. A net with nonzero cost is called *cut*, and the sum of the cost of all nets is called *cutsize*, i.e., $\sum_{e\in E_H} c(e)$. For a given set of area constraints $A = (a_i, b_i)$ for $1 \le i \le K$, the *K-way circuit partitioning problem* seeks a partition P of V into nonempty disjoint sets $V_1, V_2, \cdots, V_K$ such that $\sum_{v\in V_i} a(v)$ is bounded by $[a_i, b_i]$, and such that the cutsize is minimized.

III. Theoretical Backgrounds

In this section, we provide theoretical backgrounds on the concept of edge separability, edge contractibility, and maximum adjacency (MA) vertex ordering for an edge-weighted undirected graph. Then, we discuss on how to compute a tight estimate of edge separability efficiently for an application in circuit clustering.

A. Edge Separability and Contractibility

The input netlist NL is modeled with a simple undirected graph called a *netlist graph* $G(V, E, s, w)$ with size function s for each $v \in V$ and weight function w for each $e \in E$. For each net $k \in NL$, we form a $|k|$-clique and assign a weight of $1/(|k|-1)$ to each edge in the $|k|$-clique.[5] Let $n = |V|$ and $m = |E|$. A nonempty subset $X \subset V$ defines a *cut*, and the *cutsize* of cut X, denoted by $c(X)$, is defined as the sum of weights of outgoing edges from vertices in X. If cut X consists of single vertex x, we use $c(x)$ instead of $c(\{x\})$ and alternatively call it *degree* of x. For a given edge $e = (x, y)$, $s(e)$ denotes $s(x) + s(y)$, and $m(e)$ denotes $\min\{c(x), c(y)\}$.

A cut X is said to separate vertices x and y, if $x \in X$ and $y \in V - X$, or $x \in V - X$ and $y \in X$. A cut U that minimizes $c(U)$ in G is called *minimum cut*, and cutsize of minimum cut U is called *minimum cutsize* and denoted by $\lambda(G)$. The minimum degree of G, denoted by $\delta(G)$, is defined as $\delta(G) = \min\{c(x) | x \in V\}$. For $x, y \in V$ with $x \neq y$, we define $x - y$ *mincut* to be the value of the minimum cutsize among the cuts separating x and y and denote it $\lambda(x, y)$. From the maximum flow minimum cut theorem by Ford and Fulkerson [31], we can obtain $\lambda(x, y)$ by computing the maximum flow between x and y. When $e = (x, y) \in E$, $\lambda(x, y)$ is called *edge separability* and denoted by $\lambda(e)$. Formally, we define this as follows.

Definition 1 (Edge Separability): For an edge $e = (x, y) \in E$ in an undirected graph $G(V, E, s, w)$ with vertex size function s and edge weight function w, edge separability of e, denoted by $\lambda(e)$, is defined as the minimum cutsize among the cuts separating x and y in G.

The following lemma provides lower and upper bounds of the edge separability.

Lemma 1: For every edge $e = (x, y) \in E$ in G, where $\lambda(e)$ denotes edge separability of e, $c(x)$ denotes the degree of vertex x, and $m(e)$ denotes $\min\{c(x), c(y)\}$, the following inequality holds:

$$w(e) \leq \lambda(e) \leq m(e).$$

The proof is straightforward. An illustration is shown in Fig. 2.

For a given edge $e = (x, y) \in E$, we define *contraction* of edge e by merging x with y, removing e from G, replacing each edge of the form (y,z) with (x,z), and updating size of x by $s(x) = s(x) + s(y)$. If this process creates parallel edges, we merge them into a single edge whose weight is equal to the sum of weights of the parallel edges as illustrated in Fig. 2. We denote the graph obtained by contracting all edges in subset $F \subseteq E$ as $G' = (G/F, s', w')$, where G/F is the graph obtained from G by this contraction, s' is the updated vertex size function, and w' is the updated edge weight function of the resulting graph. Note that the actual number of contractions performed may not equal $|F|$, since some of edges in F are merged into others during the series of edge contractions. Then, the *edge contractibility* is defined as follows.

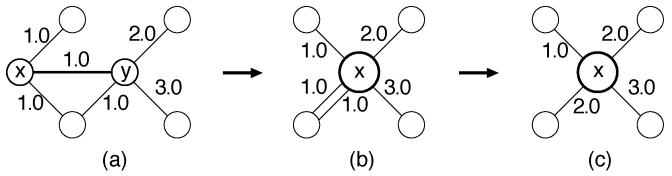


Fig. 2. (a) For $e = (x, y)$, $w(e) = 1.0$, $\lambda(e) = 2.0$, $c(x) = 3.0$, $c(y) = 7.0$, validating Lemma 1. (b) contracting $e = (x, y)$, where vertices x and y are merged into x. (c) parallel edges are merged.

Definition 2 (Edge Contractibility): Let $\bar{\lambda}$ be the cutsize of a cut $X \subset V$ in G. An edge $e = (x, y) \in E$ is called contractible with respect to $\bar{\lambda}$ if and only if $\lambda(e) \geq \bar{\lambda}$. If e is contractible with respect to $\lambda(G)$, we simply call e contractible.

Thus, if e is contractible with respect to $\bar{\lambda}$, we have $\lambda(G) = \min\{\lambda(G/e), \bar{\lambda}\}$. This indicates that computing $\lambda(G)$ is reduced to computing $\lambda(G/e)$ once such $\bar{\lambda}$ and e are found. The intuition behind edge contractibility is that the two end vertices x and y of a contractible edge $e = (x, y)$ are guaranteed to be on the same side of some minimum cut U so that $\lambda(G)$ is preserved in G/e after the contraction of e.

B. Tighter Lower Bound of Edge Separability

Computation of the edge separability for a given edge $e = (x, y)$ in an edge-weighted undirected graph G is equivalent to finding the maximum flow between x and y. Since the currently best known time bounds for solving the maximum flow problem is $O(mn \log(n^2/m))$, due to Goldberg and Tarjan [1], the computation of $\lambda(e)$ for all edges in G requires $O(m^2 n \log(n^2/m))$ time. Thus, direct computation of edge separability for all edges in G is extremely time consuming, even for moderate-size graphs with a few thousand vertices. Lemma 1 indicates that $w(e)$ serves as a lower bound of $\lambda(e)$, but we found out that (i) there exists a better approximation of $\lambda(e)$, (ii) it requires only $O(m + n \log n)$ to compute the approximation of $\lambda(e)$ for *all* edges in E.

Nagamochi and Ibaraki [2] proposed a novel algorithm named `MINCUT` that computes $\lambda(G)$, i.e., the minimum cutsize of the given $G(V, E, s, w)$ without any flow computation. `MINCUT` repeatedly calls a subroutine `CAPFOREST` that computes the set of contractible edges in G in $O(m + n \log n)$ time. `CAPFOREST` is based on traversing vertices of G according to the MA ordering of vertices in G. The intuition behind MA ordering is that it chooses a vertex that is *most tightly connected* to the vertices that are already in the order. Then, `CAPFOREST` traverses vertices of V in MA ordering, while labeling each edge with some value $q(e)$. Finally, the contractible edges are

[5]We merge all parallel edges into single edge during the transformation to make G simple.

```
CAPFOREST(G)
Input: edge weighted undirected graph G
Output: contractible edge set Z(G) ⊂ E
1.  label all v ∈ V unvisited;
2.  label all e ∈ E unscanned;
3.  r(v) = 0 for all v ∈ V;
4.  while (there exists unvisited vertex)
5.      choose an unvisited vertex x with largest r(x);
6.      for (each y where e = (x, y) is unscanned)
7.          r(y) = r(y) + w(e);
8.          q(e) = r(y);
9.          mark e scanned;
10.     mark x visited;
11. endwhile
12. Z(G) = {e | e ∈ E, q(e) ≥ λ̄(G)}
```

Fig. 3. CAPFOREST algorithm [2] for computing contractible edge set $Z(G)$ in $O(m + n \log n)$ time. The vertices are visited in MA ordering.

computed by comparing $q(e)$ to $\bar{\lambda}(G)$, where $\bar{\lambda}$ denotes the minimum cutsize discovered so far.[6]

For two nonempty subset $X, Y \subset V$, where $X \cup Y \subseteq V$, let $c(X, Y)$ denote the sum of weights of edges between X and Y. The MA ordering of vertices in G defined as follows:

Definition 3 (MA Ordering): An ordering of all vertices in V is called MA ordering in G, if it satisfies for all $1 \leq i \leq n$

$$c(\{v_1, v_2, \cdots, v_{i-1}\}, v_i) = \max\{c(\{v_1, v_2, \cdots, v_{i-1}\}, x) | x \in \{v_i, v_{i+1}, \cdots, v_n\}\}$$

If set A denotes the vertices that have already been selected and v is a candidate vertex, the degree of connection is measured by $c(A, v)$, i.e., the sum of weights of edges between A and v. The description of CAPFOREST is shown in Fig. 3. Initially, all vertices are unvisited and all edges are unscanned. CAPFOREST maintains variables $r(v)$ for each vertex v and $q(e)$ for each edge e, where $r(v)$ is $c(A, v)$, i.e., the sum of the weights of the edges between v and the set A of vertices already visited in MA ordering. $q(e)$ for $e = (x, y)$ is the value of $r(y)$ when e is scanned from x. Then, CAPFOREST always chooses the unvisited vertex v with maximum $r(v)$ and scans all its unscanned outgoing edges. The contractible edge set $Z(G)$ is computed by comparing $q(e)$ to $\bar{\lambda}(G)$, where $\bar{\lambda}$ denotes the minimum cutsize discovered so far. CAPFOREST does not require precomputation of $\lambda(G)$ in order to calculate contractible edge set $Z(G)$. Instead, the minimum degree $\delta(G)$ is used as a starting point to perform gradual update on $\lambda(G)$. CAPFOREST updates $\bar{\lambda} = \min\{\bar{\lambda}, c'(x)\}$ at the contraction of $e = (x, y)$, where $c'(x)$ denotes updated degree of vertex x.

CAPFOREST manages unscanned vertices with Fibonacci heap for which UPDATE_HEAP (line 7) takes $O(\log n)$ time while FIND_MAX (line 5) takes $O(1)$ time. Since the while loop repeats $O(V)$ times, the algorithm spends $O(n \log n)$ to manage vertex heap. In addition, CAPFOREST scans all edges once (line 8). Thus, the overall time complexity is $O(m + n \log n)$.

An illustration of CAPFOREST algorithm is shown in Fig. 4. The given edge weighted graph G with $c(U) = \lambda(G) = 4$ and $\delta(G) = 6$ is shown in (1), where U denotes the minimum cut of G. The vertex at the upper left corner is randomly chosen as the first vertex in MA ordering since $r(x) = 0$ initially for all $x \in V$. Then, CAPFOREST visits each vertex in MA ordering by choosing x with maximum $r(x)$ and labels each edge e with $q(e)$. (12) shows that $c(U) = \lambda(G) = 4$ is still preserved even after contraction of edges in $Z(G)$.

In the original paper by Nagamochi and Ibaraki [2], the following theorem is provided.

Theorem 1 ([2]): For every edge $e = (x, y) \in E$ in $G = (V, E, s, w)$, where $q(e)$ denotes the edge label computed by CAPFOREST algorithm and $\lambda(e)$ denotes the edge separability of e, the following inequality holds:

$$q(e) \leq \lambda(e).$$

Proof: The proof is based on the concept of *sparse k-connected spanning subgraph of a k-connected graph* [33], and mathematically demanding. Since the introduction, many attempts [34]–[36] have been made in the graph connectivity community to simplify the proof. □

We can further show the relationship between $w(e)$ and $q(e)$.

Lemma 2: For every edge $e = (x, y) \in E$ in $G = (V, E, s, w)$, where $w(e)$ denotes the weight of e and $q(e)$ denotes the edge label computed by CAPFOREST algorithm, the following inequality holds:

$$w(e) \leq q(e).$$

Proof: Assume that $x \in A$ and $y \notin A$ for $e = (x, y) \in E$ and that $r(y) = \max\{r(v) | v \notin A\}$, where A denotes the set of vertices visited by CAPFOREST based on MA ordering. Let $E(y) = \{(v, y) \in E | v \in A\}$. If $E(y) = \{e\}$, $r(y) = w(e) = q(e)$, since $r(y) = c(A, y)$. Otherwise, there exists a vertex $z \neq x$, such that $e' = (z, y) \in E(y)$. Then, $r(y) = w(e) + w(e') = q(e)$, showing that $w(e) < q(e)$. □

Finally, from Lemma 1, Theorem 1, and Lemma 2, we establish the following theorem.

Theorem 2: For every edge $e = (x, y) \in E$ in $G = (V, E, s, w)$, where $w(e)$ denotes the weight of e, $q(e)$ denotes the edge label computed by CAPFOREST algorithm, $\lambda(e)$ denotes the edge separability of e, and $m(e)$ denotes $\min\{c(x), c(y)\}$, the following inequality holds:

$$w(e) \leq q(e) \leq \lambda(e) \leq m(e).$$

In VLSI circuits, $|V| \simeq |E|$, since the size of nets is bounded by a relatively small constant ($\simeq$100). Under such an assumption, the complexity of computing $w(e)$, $q(e)$, $\lambda(e)$, and $m(e)$ are $O(1)$, $O(n \log n)$, $O(n^2 \log n)$, and $O(1)$, respectively. Section V provides detailed statistics of these values collected from benchmark circuits as well as the effectiveness of using them in the context of multilevel partitioning.

IV. Edge-Separability-Based Clustering Algorithm

In this section, we discuss our graph-connectivity-based multilevel clustering algorithm ESC that runs in $O(n \log n)$ time. We provide an overview of the ESC algorithm, followed by a discussion on the edge contraction algorithm, under a given cluster-size constraint. Lastly, an application of ESC algorithm in the context of multilevel partitioning is presented.

[6]We note that the first work that used the MA ordering for partitioning application is by Alpert and Kahng [32]. In their approach, MA ordering is computed and split into partitions using dynamic programming. Our ESC clustering algorithm is also based on MA ordering. However, ESC does not use the vertex ordering itself, but it uses edge label $q(e)$ computed by CAPFOREST that visits vertices in MA order.

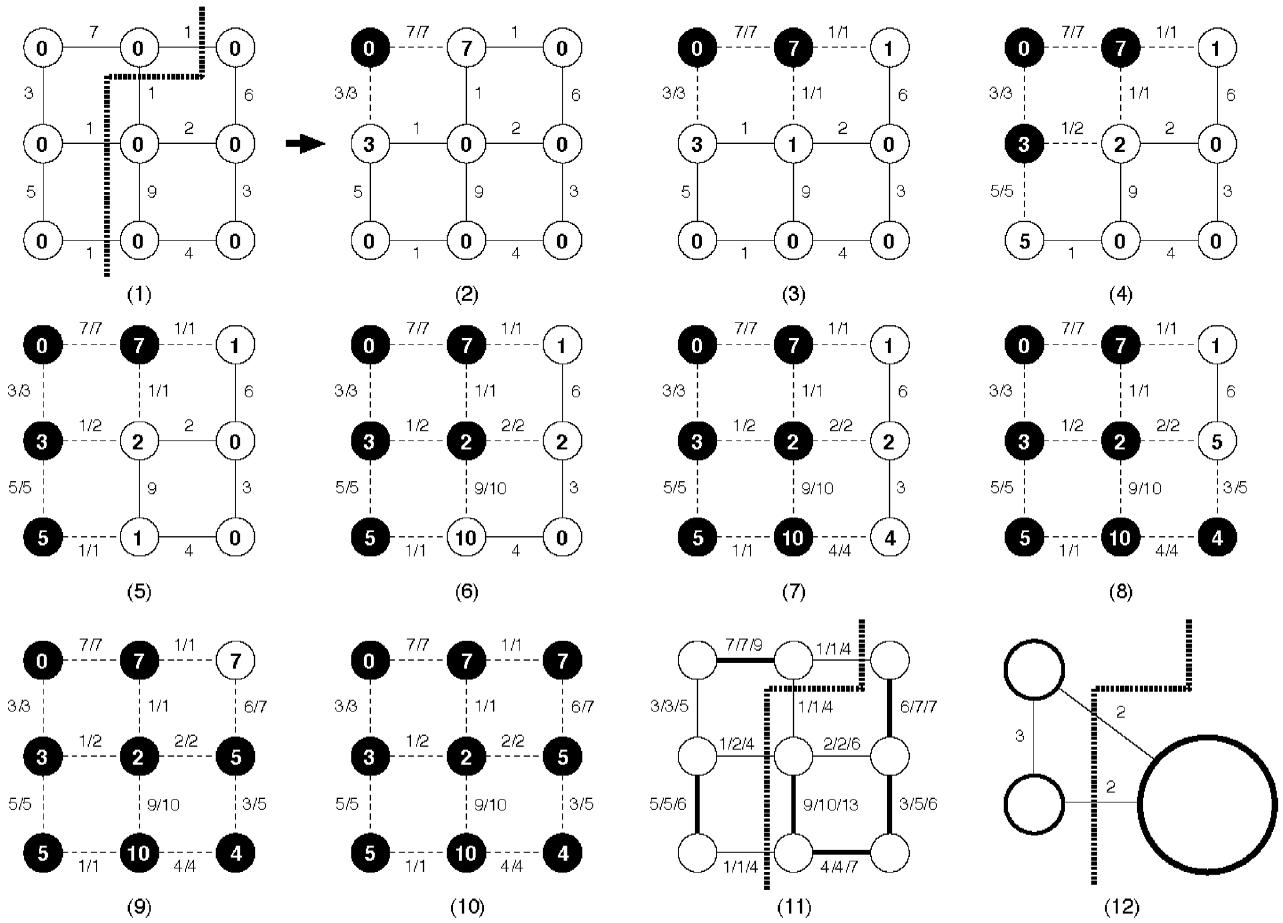


Fig. 4. Illustration of CAPFOREST algorithm. Values inside vertex x denotes $r(x)$, and edge label denotes $w(e)/q(e)/\lambda(e)$. Visited vertices are shown in dark circles, and scanned edges are shown in dotted lines.

A. Overview of ESC Algorithm

The ESC algorithm is a bottom-up clustering algorithm, where clusters grow from the contraction of contractible edges. This is the equivalence to merging two clusters that have direct connection via a contractible edge. Each vertex belongs to its own cluster initially, and the clusters grow from greedy merging based on edge separability. The clustering process is guided by $q(e)$, a better estimation of edge separability $\lambda(e)$ than edge weight $w(e)$ as shown in Theorem 2. The ESC clustering algorithm can be applied repeatedly to build multilevel cluster hierarchy. Assuming that $\bar{\lambda}$ denotes the minimum cutsize discovered so far, the following provides overall flow of the ESC algorithm that constructs h-level cluster hierarchy.

1) Let $\bar{\lambda} = \delta(G)$.
2) Run CAPFOREST(G) and compute $Z(G) = \{e | e \in E, q(e) \geq \bar{\lambda}\}$.
3) Edges in $Z(G)$ are sorted into heap Q based on their *rank* (to be defined in Section IV-B).
4) Remove edge e from the top of Q and see if contraction of e violates the cluster-size constraint.[7] If it satisfies the constraint, contract e, and update G, $\bar{\lambda}$ and Q accordingly (to be discussed in Section IV-B). Repeat until Q is not empty.
5) Repeat Steps 2 to 4 h times to obtain h-level cluster hierarchy.

The contractible edge set $Z(G)$ is computed by comparing $q(e)$ to the minimum cutsize $\lambda(G)$. However, ESC does not require precomputation of $\lambda(G)$ in order to calculate contractible edge set $Z(G)$. Instead, the minimum degree $\delta(G)$ is used as a starting point to perform gradual update on $\lambda(G)$. More specifically, upon the contraction of $e = (x, y)$, we update $\bar{\lambda} = \min\{\bar{\lambda}, c'(x)\}$, where $c'(x)$ denotes updated degree of vertex x. Step 4 involves our CONTRACT algorithm explained in the following Section IV-B.

B. Size-Constrained Edge Contraction

After the computation of contractible edge set $Z(G)$, we heapify edges in $Z(G)$ into edge heap Q. The *rank* of edge $e = (x, y)$ in Q, denoted by $t(e)$, is defined to be $q(e)/m(e)$, where $q(e)$ denotes the edge label CAPFOREST computes and $m(e) = \min\{c(x), c(y)\}$. All edges in Q are ordered in descending order of their rank. The rank $t(e)$ is computed in such a way that it gives higher priority to edges with larger $q(e)$ values and edges whose contraction results in smaller increase of degree. We use $\min\{c(x), c(y)\}$ instead of $c(x) + c(y)$

[7]Note that we allow the merging of the vertex with both unmerged and merged vertices. As a result, each cluster may contain an arbitrarily large number of vertices up to the given size limit.

```
CONTRACT(Z(G), A)
Input: contractible edge set Z(G) ⊆ E of G, size limit A
Output: reduced graph G' = (V', E', s', w')
1.  Q = BUILD_HEAP(Z(G));
2.  while (there exists edge in Q)
3.      e = (x, y) = FIND_MAX(Q);
4.      if (s(x) + s(y) ≤ A)
5.          remove e and y from G;
6.          s(x) = s(x) + s(y);
7.          for (each vertex z adjacent to y by edge e'(y, z))
8.              if (ē = (x, z) ∉ E)
9.                  disconnect e'(y, z) and connect e''(x, z);
10.             else
11.                 delete e' from G;
12.                 w(ē) = w(ē) + w(e');
13.                 q(ē) = max{q(ē), q(e')};
14.             c(x) = c(x) + w(e');
15.         c(x) = c(x) − w(e);
16.         if (c(x) < λ̄)
17.             λ̄ = c(x);
18.         for (each edge e' incident to x)
19.             if (e' ∉ Q and q(e') ≥ λ̄)
20.                 INSERT_HEAP(Q, e');
21.             else
22.                 UPDATE_HEAP(Q, e');
23. endwhile
24. return G;
```

Fig. 5. CONTRACT algorithm to grow clusters by contraction of edges in the given contractible edge set $Z(G)$ under size constraint A.

to encourage absorbing of dangling vertices (e.g., vertices with connection to only one other cluster). Ties are randomly broken. The following operations are used to manage edge heap Q-based on edge rank $t(e)$:

- BUILD_HEAP$(Z(G))$: build Q from edges in $Z(G)$ in $O(k \log k)$, where $k = |Z(G)|$;
- FIND_MAX(Q): return the edge with maximum rank from the top of Q in $O(1)$;
- INSERT_HEAP(Q, e): insert e into Q in $O(\log k)$;
- UPDATE_HEAP(Q, e): update ordering of Q from the position of e in $O(\log k)$.

Fig. 5 shows CONTRACT algorithm that performs size-constrained edge contraction for given contractible edge set $Z(G)$ and cluster size limit A. Two major operations CONTRACT performs areL i) building and managing edge heap Q-based on edge rank $t(e)$ and ii) updating the given graph G upon each edge contraction to obtain correct $G/Z(G)$ upon its termination. After building Q from $Z(G)$ (line 1), CONTRACT removes edge $e = (x, y)$ from the top of Q (line 3) and see if contraction of e violates the cluster size constraint (line 4). If it satisfies the constraint, e and y are removed from G (line 5), and $s(x)$ is updated (line 6). Then, CONTRACT replaces each edge of the form (y, z) (lines 7 and 8) with (x, z) (line 9). If this process may create parallel edges due to the existing connection between x and z (line 10), they are merged into a single edge (line 11) whose weight is equal to the sum of weights of the parallel edges (line 12). Then, the maximum q value among q values of the parallel edges is assigned to $q(x, z)$ to maintain a lower bound on $\lambda(x, z)$ (line 13). The degree of x is updated accordingly (lines 14 and 15) and used to update the current minimum cutsize $\bar{\lambda}$ (lines 16 and 17). Then, for each edge e' incident to x (line 18), CONTRACT either inserts e' into Q, if it was not in Q, but becomes contractible due to the update of $\bar{\lambda}$ (line 19 and 20) or updates position of e' in Q due to the update of $q(e')$ and $c(x)$ and, thus, $t(e')$ (line 22).

```
ESC(NL, h, A)
Input: netlist NL, hierarchy level h, and size limit A
Output: h-level cluster hierarchy H
1.  H_0 = TRANSFORM_HYPERGRAPH(NL);
2.  G = TRANSFORM_GRAPH(NL);
3.  λ̄ = δ(G) = min{c(x) | x ∈ V};
4.  for (i = 1 to h)
5.      CAPFOREST(G);
6.      Z(G) = {e | e ∈ E, q(e) ≥ λ̄};
7.      G' = CONTRACT(Z(G), A_i);
8.      H_i = BUILD_LEVEL(H_{i−1}, G');
9.  return H;
```

Fig. 6. Description of ESC algorithm that builds a multilevel clustering hierarchy.

Finally, the most updated $G' = (V', E', s', w') = G/z(G)$ is returned (line 24).

If t_1 denotes $|Z(G)|$ and t_2 denotes the maximum degree among vertices in G, the complexity of CONTRACT is $O(t_1 t_2 \log t_1)$, due to the $O(\log t_1)$ time update of Q for $O(t_2)$ number of edges, which repeats for t_1 times in total. In most cases, large nets (>100) such as clock nets are ignored during clustering and partitioning in VLSI circuit design. Then, we may assume that the maximum degree is bounded by a constant k, which also implies that $|E| \leq |V| \cdot k/2$. Under such an assumption, the time complexity of CONTRACT becomes $O(t_1 t_2 \log t_1) = O(n \cdot k/2 \cdot k \log(n \cdot k/2)) = O(n \log n)$.

C. Extension to Multilevel Clustering

In the ESC algorithm, the combination of CAPFOREST and CONTRACT is repeated h times to build h-level cluster hierarchy. During each call of CONTRACT, we can build a multiple-level cluster hierarchy by establishing parent–child relation among vertices. More specifically, upon contraction of $e = (x, y)$, where x remains in G, x becomes the parent of y in a tree-like cluster hierarchy. In order to guide the clustering process to generate a balanced hierarchy, where clusters at each level are of similar size, we impose different size constraint at different level. More specifically, we impose a size limit of A_i at clustering at level i, where $A_1 < A_2 < \cdots < A_h$. Fig. 6 shows the ESC algorithm grows clusters by repeatedly applying CAPFOREST and CONTRACT under the given size constraint $A = \{A_1, A_2, \cdots, A_h\}$. BUILD_LEVEL constructs next coarser hypergraph based on the parent-child relation obtained during edge contraction in $O(n)$ time. CAPFOREST takes $O(m + n \log n)$ and CONTRACT takes $O(n \log n)$ time. Therefore, the overall complexity of the ESC algorithm is $O(h \cdot (m + n \log n + n \log n + n)) = O(n \log n)$. An illustration of the ESC algorithm with $h = 2$ is shown in Fig. 7.

D. Application to Multilevel Partitioning

In general, the impact of clustering is more visible when combined with the subsequent partitioning. In such a case, clustering is applied as a preprocess to reduce the problem size so that the subsequent partitioning can optimize their objective functions such as cutsize and wirelength more effectively.

ESC clustering can be used in the *two-level partitioning* framework. First, a clustering C of a netlist H is generated, then this clustering is used to induce the coarser netlist H'

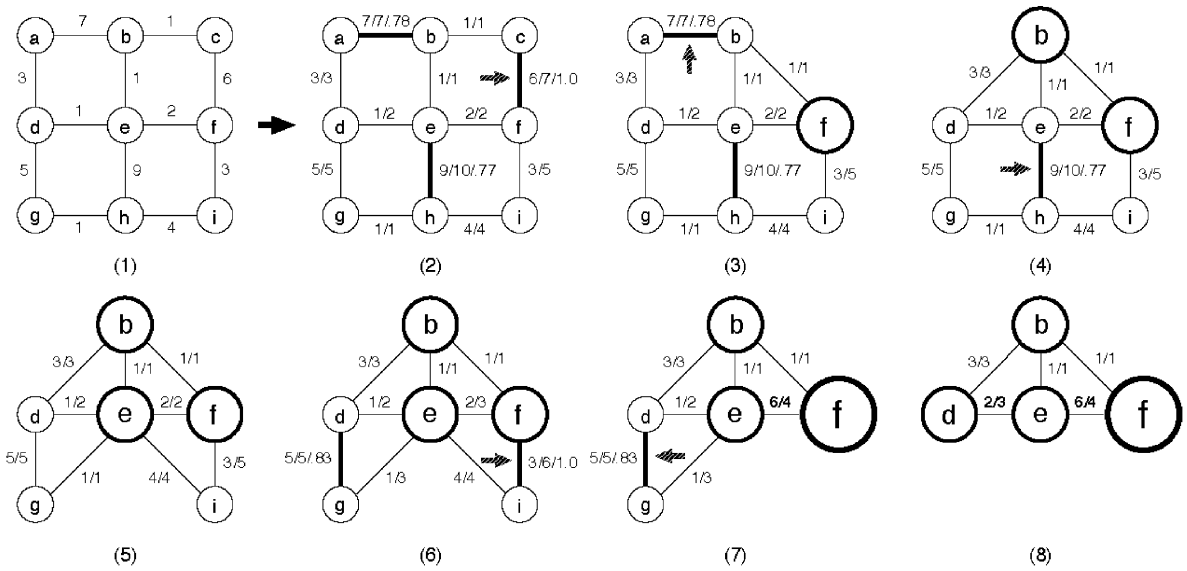


Fig. 7. Illustration of ESC algorithm with $h = 2$. Contractible edges are shown in thick lines, and arrows point to the edges contracted. The edge label means $w(e)/q(e)/t(e)$, and bold edge label denotes the updated $w(e)/q(e)$ from contraction. (1–5) shows clustering under $A_1 = 2$, and (6–8) shows clustering under $A_2 = 3$. (1) G, (2) $Z(G)$ based on $\bar{\lambda} = \delta(G) = c(g) = 6$, (4) $\bar{\lambda} = c(b) = 5$, (6) $Z(G)$ based on $\bar{\lambda} = 5$.

from H. Partitioning is then run once on H' to yield a solution P', and P' is projected to a new partitioning P on top of H. Finally, partitioning is run a second time on H using P as its initial solution. These first and second partitioning runs are called a *refinement* step, which refers to the improvement of an initial solution via local moves. The two-level partitioning framework requires a two-level clustering hierarchy (original and clustered level), which is obtained by applying clustering once on top of the original circuit. Note that the two-level partitioning framework can be easily generalized to the *multilevel partitioning* framework. The multilevel partitioning makes use of a multilevel clustering hierarchy (original and several intermediate levels), which is obtained by applying clustering successively on top of the prior clustered netlist.

Let h be the height of cluster hierarchy desired and H_i denote netlist at level i, where H_0 corresponds to the original (= bottom level) netlist. In a multilevel framework, a clustering of H_0 is used to induce the coarser netlist H_1, then a clustering of H_1 induces H_2, and so on until the most coarsened netlist H_h is constructed. A bipartitioning solution $P_h = \{X_h, Y_h\}$ is found for H_h, and this solution is then projected to $P_{h-1} = \{X_{h-1}, Y_{h-1}\}$. P_{h-1} is then refined by partitioning again. This collaboration of partitioning solution projection and refinement is called *uncoarsening* process, and it continues until a refined partitioning of H_0 is obtained.

Definition 4 (Multilevel Bipartitioning): Let h denote the height of cluster hierarchy desired. Let H_i denote netlist at level i, where H_0 corresponds to the original netlist and H_{i+1} is induced from H_i by the clustering $C = \{x, y, \cdots, C_k\}$ for $0 \leq i \leq h-1$. The projection of bipartitioning solution $P_{i+1} = \{X_{i+1}, Y_{i+1}\}$ of H_{i+1} onto H_i is the solution $P_i = \{X_i, Y_i\}$, where $X_i = \{v \in V_i | C_h \in C$ and $C_h \in X_{i+1}$ and $v \in C_h\}$ and $Y_i = \{v \in V_i | C_h \in C$ and $C_h \in Y_{i+1}$ and $v \in C_h\}$.

Note that it is straightforward to extend this definition to that of multilevel K-*way partitioning*, where clusters are partitioned into K-blocks instead of two, i.e., $P_h = \{B_h^1, B_h^2, \cdots, B_h^K\}$, where B_j^i denotes the i-th partition at level j.

Multilevel partitioning offers several advantages over two-level partitioning. First, the single coarsening step of two-level approaches can make H_1 too coarse representation of H_0. Multilevel approaches enables coarsening to proceed more slowly, giving the iterative engine more opportunities for refinement. Second, multilevel approaches can be extremely efficient if a fast clustering and refinement strategy is used. Refinement for each netlist H_i typically requires only a few iterations of partitioning to converge since it begins with a high-quality initial solution. Finally, refinement proceeds with progressively larger netlists, implying that the number of local moves performed during a single iteration of partitioning becomes progressively larger. This permits the refinement algorithm to avoid bad local minima via big steps at high levels, while still being able to find a good final solution via detailed refinement at low levels. Section V provides various experimental results on multilevel partitioning to demonstrate the impact of our clustering algorithm ESC.

V. Experimental Results

We implemented our algorithms in C++/STL, compiled with gcc v2.4, and tested on an ULTRA SPARC60 at 360 MHz. The benchmark circuits are from the Microelectronics Center of North Carolina (MCNC) and ISPD'98 Benchmark Suite [37] for ESC algorithm evaluation. We obtained the latest binary executable of the state-of-the-art cutsize-driven multilevel partitioning algorithm `hMetis` [10] (v1.5.3) for the evaluation. We report cutsize and runtime from 2-, 8-, 16-, and 32-way partitioning results. The area of the cells is assumed to be uniform, and all pads are included in the partitioning. 2-, 8-, 16-, and 32-way algorithms use area skew of $[0.45, 0.55]$, $[0.45^3 = 0.091,\ 0.55^3 = 0.166]$, $[0.45^4 = 0.041,\ 0.55^4 = 0.092]$, and

TABLE I
STATISTICS OF $w(e)$ (= EDGE WEIGHT), $q(e)$ (= ESTIMATE OF EDGE SEPARABILITY), $\lambda(e)$ (= EDGE SEPARABILITY), AND $m(e)(= \min\{c(x), c(y)\})$ FROM MCNC AND ISPD [37] BENCHMARK CIRCUITS. THE AVERAGE VALUES FOR ALL EDGES ARE SHOWN. % DIFF SHOWS THE PERCENTAGE DIFFERENCE BETWEEN $q(e)$ AND $\lambda(e)$. TIME SHOWS THE NORMALIZED CLUSTERING TIME TO $\lambda(e)$-BASED SCHEME

Circuits		Average values				
name	$\|V\|$	$w(e)$	$q(e)$	$\lambda(e)$	$m(e)$	%diff
prim1	833	0.31	0.93	1.07	3.37	13.1
struct	1952	0.32	0.76	0.98	2.73	22.4
prim2	3014	0.25	1.05	1.12	3.39	6.2
biomed	6514	0.20	0.67	0.77	2.61	13.0
s13207	8772	0.45	0.79	0.82	2.46	3.6
s15850	10470	0.49	0.77	0.83	2.44	7.2
ind2	12637	0.22	1.05	1.12	3.75	6.3
ind3	15406	0.39	1.29	1.34	4.08	3.7
s35932	18148	0.70	0.99	1.05	2.40	5.7
s38584	20995	0.41	0.79	0.84	2.54	6.0
avq.s	21918	0.53	0.94	1.02	2.53	7.8
s38417	23949	0.59	0.91	0.99	2.48	8.1
avq.l	25178	0.52	0.91	1.00	2.56	9.0
ibm01	12752	0.33	1.09	1.21	3.82	10.0
ibm02	19601	0.32	1.07	1.23	3.76	13.0
ibm03	23136	0.42	1.22	1.33	3.70	8.3
ibm04	27507	0.47	1.20	1.32	3.31	9.1
ibm05	29347	0.38	1.11	1.25	3.79	11.2
TIME		0.0008	0.0019	1.0000	0.0008	-

TABLE II
PARTITIONING RESULTS FROM MCNC AND ISPD [37] BENCHMARK CIRCUITS. THE MINIMUM CUTSIZE OF 20 RUNS OF LR BASED [16] MULTILEVEL BIPARTITIONING IS SHOWN. RATIO SHOWS THE NORMALIZED TOTAL TO $\lambda(e)$ RESULTS. TIME SHOWS THE NORMALIZED PARTITIONING TIME TO $\lambda(e)$-BASED SCHEME

Circuits			Minimum cutsize			
name	$\|V\|$	$\|E\|$	$w(e)$	$q(e)$	$\lambda(e)$	$m(e)$
prim1	833	4708	48	48	47	51
struct	1952	8318	33	33	33	33
prim2	3014	21836	153	148	147	187
biomed	6514	45676	84	83	84	86
s13207	8772	23983	71	53	53	77
s15850	10470	27919	55	46	46	67
ind2	12637	109493	182	169	169	199
ind3	15406	77561	264	241	241	287
s35932	18148	34331	68	54	48	67
s38584	20995	69552	74	59	52	77
avq.s	21918	57434	154	129	127	178
s38417	23949	56401	65	50	52	71
avq.l	25178	61405	167	130	137	187
ibm01	12752	99962	185	180	181	190
ibm02	19601	170200	312	297	299	332
ibm03	23136	198273	1134	972	968	1347
ibm04	27507	187059	601	526	526	623
ibm05	29347	349676	1867	1709	1712	1934
TOTAL			5517	4927	4922	5993
RATIO			1.12	1.00	1.00	1.22
TIME			1.01	1.02	1.00	1.02

$[0.45^5 = 0.018, 0.55^5 = 0.050]$, respectively. We use two cutsize metrics: Cost 1 and sum of external degree (SOED). Cost 1 counts the number of nets spanning more than one partition, and SOED is the sum of outgoing nets from each partition. All cutsizes are based on a minimum of 20 runs. Runtimes are measured in seconds and represent total elapsed CPU time. The cluster hierarchy level h is set to ten and the size limit is $A_i = 10 \cdot 2^{i-1}$ for $1 \leq i \leq h$.

A. Statistical Analysis

Table I shows the statistics of $w(e)$ (= edge weight), $q(e)$ (= estimate of edge separability computed by CAPFOREST), $\lambda(e)$ (= edge separability), and $m(e)(= \min\{c(x), c(y)\})$ for each edge $e = (x, y) \in E$. We implemented Dinic's blocking flow-based max-flow algorithm [38] for efficient computation of the edge separability $\lambda(e)$. The edge capacity of the flow network is set to $w(e)$, and we compute maximum flow from x to y and, thus, $x - y$ mincut for each edge in $G = (V, E, s, w)$. Nets that connect more than 30 cells are ignored during the clique-based transformation of the netlist into G. We applied CAPFOREST on each circuit for the computation of $q(e)$. The complexity of computing $w(e)$, $q(e)$, $\lambda(e)$, and $m(e)$ for all edges is $O(1)$, $O(n \log n)$, $O(n^3 \log n)$, and $O(1)$, respectively, for all edges.

First, we observe that the average value of $m(e)$ is about three to ten times larger than that of $w(e)$. This means there are three to ten neighboring nodes for each node in the circuit on the average. This increases the necessity of making the right decision in choosing which one to cluster together. Second, we observe that the average value of $\lambda(e)$ is about three times larger than that of $w(e)$. This shows that there are an average of two additional paths that connect x and y, rather than e itself. Thus, it is worthwhile looking at these additional paths in determining connectivity between x and y. Third, the average value of $q(e)$ is about three times larger than that of $w(e)$ and very close to $\lambda(e)$. The average difference between $q(e)$, and $\lambda(e)$ is 9.1%. This shows that $q(e)$ is indeed a tight lower bound of $\lambda(e)$, as proved in Section III-B.

Second, we use $w(e)$, $q(e)$, $\lambda(e)$, and $m(e)$ to guide clustering to validate the impact of using them in the context of cutsize minimization-based partitioning. We use the multilevel partitioning framework explained in Section IV-D for the evaluation of each clustering scheme. The edge contraction cost functions we use are $w(e)/m(e)$, $q(e)/m(e)$, $\lambda(e)/m(e)$, and $m(e)/[c(x)+c(y)]$. We use the LR [16] partitioning scheme for refinement during the uncoarsening phase.[8] We show in Table II the bipartitioning cutsize results as well as the total partitioning time normalized to that of $\lambda(e)$. We observe that the best cutsize results are obtained when $q(e)$ and $\lambda(e)$ are used during clustering. This means that both cost functions that are based on the edge separability serve as an effective guidance for cutsize minimization. However, we note that the computation of $\lambda(e)$ involves a prohibitive amount of runtime—as much as three days. Thus, using $q(e)$ proves to be the best option in terms of quality and runtime tradeoff.

B. Comparison to Other Clustering Algorithms

For the given edge $e = (x, y) \in E$ from a netlist graph $G = (V, E)$, whose contraction generates new larger cluster C, the following well known *clustering cost functions* are proposed in the literature.

1) *Absorption* [3] (maximize): The absorption of C is defined as $abs(C) = \sum_{e \in C} w(e)$. It measures the sum of weights of edges "absorbed" into C.

[8]We note that the performance of LR is more stable than that of FM [39]—FM results vary significantly depending upon the random initial solution, whereas LR generates more stable (and better) results regardless of the initial solutions.

TABLE III
COMPARISON OF VARIOUS CUTSIZE DRIVEN BOTTOM-UP CLUSTERING ALGORITHMS IN TERMS OF MULTILEVEL LR BIPARTITIONING RESULTS. ALGORITHMS IN COMPARISON INCLUDE ABSORPTION (ABS), DENSITY (DEN), RENT PARAMETER (REP), RATIO CUT (RTC), CLOSENESS (CLO), CONNECTIVITY (CON), FIRST CHOICE (FC), AND OUR EDGE SEPARABILITY BASED METHOD (ESC). RATIO SHOWS THE NORMALIZED TOTAL TO ESC RESULTS. TIME SHOWS THE TOTAL PARTITIONING TIME

Circuit		Multi-level LR Bipartitioning Cutsize							
name	size	ABS	DEN	REP	RTC	CLO	CON	FC	ESC
ibm01	12752	192	191	261	186	184	302	185	181
ibm02	19601	299	298	271	299	291	272	297	298
ibm03	23136	1037	1023	1175	1071	1081	1182	1109	973
ibm04	27507	596	570	619	599	570	621	587	573
ibm05	29347	1821	1761	1734	1751	1741	1752	1822	1718
ibm06	32498	960	952	967	1035	991	1012	1035	938
ibm07	45926	945	912	964	971	1124	1023	957	855
ibm08	51309	1193	1183	1491	1256	1473	1423	1199	1158
ibm09	53395	1023	979	1361	741	765	932	815	624
ibm10	69429	1651	1351	1573	1479	1427	1810	1401	1280
ibm11	70558	1331	1231	1321	1159	1410	1951	1179	994
ibm12	71076	2421	2234	2523	2021	2200	2582	2105	1912
ibm13	84199	1011	1036	1225	1144	1231	1936	1001	864
ibm14	147605	2093	2091	3150	1981	2379	3073	2121	1894
ibm15	161570	3410	3328	3893	3510	4221	4081	3114	2736
ibm16	183484	2321	2243	2851	2575	2290	2261	1986	1762
ibm17	185495	2841	2931	4031	2741	2754	2927	2723	2219
ibm18	210613	2459	2526	3491	2726	2964	2931	2134	1768
TOTAL	-	27604	26840	32901	27245	29096	32071	25770	22747
RATIO	-	1.21	1.18	1.44	1.20	1.28	1.41	1.13	1.00
TIME	-	9242	9971	13431	9872	9912	9841	9134	9423

2) *Density* [4], [5] (maximize): The density of C is defined as $abs(C)/s(C)$. It measures the "density" of C by taking the ratio of the edges "absorbed" in C to the size of C.
3) *Rent Parameter* [6] (minimize): The Rent parameter of C is computed as follows:

$$\frac{\ln(c(C)) - \ln(d(C))}{\ln s(C)} \qquad d(C) = \frac{1}{|C|}\sum_{x \in C} c(x)$$

where $d(C)$ is the average degree of vertices in C, and $|C|$ is the number of vertices in C. A "better" placement can be obtained from the smaller Rent parameter associated with each cluster, according to the Rent's rule.
4) *Ratio Cut* [7] (minimize): The ratio cut of C is computed as $c(C)/s(C)$ It is a ratio of the sum of weights of outgoing edges to the size of C. It tries to identify "natural" clusters by finding cuts that minimize $c(C)/s(C)$.
5) *Closeness* [8] (maximize): The closeness between x and y is defined as follows:

$$\frac{w(e)}{\min\{c(x), c(y)\}} - \gamma\frac{s(x)+s(y)}{\text{ave\_size}}$$

where ave_size denotes the average cluster size, and γ is a user-specified parameter to determine the magnitude of penalty term for generating large clusters. It measures the "attraction" between two clusters based on local connectivity information.
6) *Connectivity* [9] (maximize): The connectivity between x and y is defined as follows:

$$\frac{w(e)}{s(x)\cdot[c(x) - w(e)]} + \frac{w(e)}{s(y)\cdot[c(y) - w(e)]}.$$

It is another local connectivity based method that focuses on: i) minimizing number of edges cut after contraction and ii) preventing early formation of large clusters.

TABLE IV
COMPARISON BETWEEN FC (FIRST CHOICE) [10] AND ESC IN TERMS OF: 1) THE NUMBER OF NETS AT THE TOP LEVEL CLUSTER HIERARCHY AND 2) FINAL CUTSIZE AT THE BOTTOM LEVEL. THE TOTAL LEVEL IS SET TO $ln(\#cell)$. RATIO SHOWS THE NORMALIZED TOTAL TO ESC RESULTS

Circuit			Top level nets		Final cut nets	
name	#cell	lev	FC	ESC	FC	ESC
ibm01	12752	9	4027	3341	185	181
ibm02	19601	9	7129	7791	297	298
ibm03	23136	10	9240	8529	1109	973
ibm04	27507	10	11654	11467	587	573
ibm05	29347	10	8802	8689	1822	1718
ibm06	32498	10	9877	9048	1035	938
ibm07	45926	10	14362	14131	957	855
ibm08	51309	10	14959	14138	1199	1158
ibm09	53395	10	21339	16952	815	624
ibm10	69429	11	25719	24504	1401	1280
ibm11	70558	11	29036	24489	1179	994
ibm12	71076	11	28243	26831	2105	1912
ibm13	84199	11	32345	27937	1001	864
ibm14	147605	11	46267	46053	2121	1894
ibm15	161570	11	67716	60285	3114	2736
ibm16	183484	12	59977	51813	1986	1762
ibm17	185495	12	63770	59135	2723	2219
ibm18	210613	12	52096	45565	2134	1768
TOTAL	-	-	506558	460338	25770	22747
RATIO	-	-	1.10	1.00	1.13	1.00

7) *First Choice* [10] (maximize): The connectivity between x and y is defined as follows:

$$\sum_{h \in hp(e)} \frac{1}{|h| - 1}$$

where $hp(e)$ denotes all hyperedges that contain both x and y. Notice that this connectivity value is identical to $w(e)$ from the given netlist graph $G = (V, E)$. In First Choice, however, the vertices are visited in a random order instead of a fixed one. Thus, we do not need to build the netlist graph explicitly since we can compute $w(e)$ on-the-fly.

TABLE V
COMPARISON AMONG hMetis [10], [40], hMetis − Kway [11], AND OUR MULTILEVEL AND MULTIWAY PARTITIONING ALGORITHM LR/ESC − PM. 2-WAY PARTITIONING RESULTS ARE BASED ON COST 1 METRIC, WHILE OTHERS ARE BASED ON SOED METRIC. RATIO SHOWS THE NORMALIZED TOTAL TO hMetis RESULTS. TIME SHOWS THE TOTAL RUNTIME

	hMetis & hMetis-Kway				LR/ESC-PM			
ckt	2-way	8-way	16-way	32-way	2-way	8-way	16-way	32-way
ibm01	180	1750	2883	4149	181	1777	2829	4147
ibm02	262	3850	7556	11821	298	4490	8145	11787
ibm03	956	5820	8205	11077	973	5737	8297	11002
ibm04	542	6214	8992	12495	573	6170	9123	12341
ibm05	1715	10749	15206	20020	1718	10410	14372	18898
ibm06	888	5784	8661	12779	938	6468	9173	13088
ibm07	853	7586	11040	15559	855	7566	11109	15224
ibm08	1142	7979	10976	15327	1158	8214	11281	15646
ibm09	624	5822	8634	12460	624	5918	8481	12339
ibm10	1256	9144	13130	19941	1280	8795	13908	19922
ibm11	960	7874	11706	17118	994	8129	11723	16941
ibm12	1918	12910	17848	25228	1912	13035	17976	24161
ibm13	840	6079	11819	17350	864	6866	12547	17902
ibm14	1837	11258	18232	29699	1894	10992	17745	27760
ibm15	2625	14586	20826	31874	2736	15565	20453	31162
ibm16	1755	14616	22924	34879	1762	14877	23520	34775
ibm17	2238	19930	33344	45961	2219	20468	33124	44403
ibm18	1541	12177	19598	30558	1768	12885	19311	29188
TOTAL	22132	164128	251580	370295	22747	168362	253117	360686
RATIO	1.00	1.00	1.00	1.00	1.03	1.03	1.01	0.97
TIME	7234	-	-	-	9423	27035	33449	38406

In order to measure and compare the impact of these clustering algorithms, we adopt the multilevel partitioning framework explained in Section IV-D. In this case, the clustering cost functions mentioned above are used to rank the edges to be contracted except for the First Choice. All of the above algorithms do not impose restrictions on grouping between already merged vertices. Note that the first four cost functions are defined in terms of a single cluster rather than a single edge that connects two clusters, i.e., $e = (x, y)$. However, it is straightforward to extend these cost functions so that we pick the edge that maximizes these costs during edge contraction. We observe that local connectivity information $w(e)$ plays a major role in determining the sequence of edge contraction for the seven schemes shown above. ESC uses $t(e)$, which is defined in Section IV-B as the contraction cost function.

Table III shows the comparison among various bottom-up clustering algorithms introduced in this section in the multilevel partitioning framework. We use LR partitioning again for cutsize refinement. Algorithms in comparison include absorption [3], density [4], [5], rent parameter [6], ratio cut [7], closeness [8], connectivity [9], first choice [10], and our edge separability-based method. We observe that ESC outperforms all other algorithms that rely on local connectivity information in terms of bipartitioning results. TIME includes total clustering and partitioning, which are comparable in all cases. In particular, we compare our ESC clustering algorithm to the First Choice (FC) scheme used in hMetis [10]. We observe that LR/ESC outperforms LR/FC by 11.7% on ISPD benchmark circuits in terms of bipartitioning results. This LR/ESC result is also very comparable to that of hMetis as shown in the next set of experimental results (Table V). The success of hMetis not only comes from FC clustering but also complex V-cycle and v-cycle refinement schemes used in hMetis. However, our connectivity based ESC clustering does not require any additional refinement schemes to produce comparable results to hMetis.

Table IV shows the comparison between FC [10] and ESC in terms of: i) the number of nets at the top level cluster hierarchy and ii) final cutsize at the bottom level. The former is the cutize among clusters at the top level and gives good indication on how good is the clustering in terms of cutsize minimization. The number of level is fixed at $ln(\#cell)$. We observe from Table IV that ESC generates consistently a smaller number of nets at the top-level netlist, with an average improvement of 10% over FC. Thus, the probability that ESC gives better initial partitioning result is higher than that of FC. In addition, this advantage at the top level is propagated all the way down to the bottom level and results in an average final cutsize improvement of 13% over FC, as seen in the last two columns.

C. Comparison to Other Partitioning Algorithms

In order to validate the effectiveness of using $q(e)$, we developed a multilevel and multiway partitioning algorithm named LR/ESC − PM—it performs LR bipartitioning algorithm [16] on top of multilevel ESC cluster hierarchy. This is then used as the bipartitioning engine for pairwise movement-based multiway partitioning framework PM [41]. In PM, matching of K-blocks are computed, and bipartitioning is applied simultaneously on these block pairs. At the end of the current pass, a new block paring configuration is derived, depending on the cutsize gain of the previous pass, which continues until no more gain is observed. We observe from the results summarized in Table V that LR/ESC − PM obtains comparable results to state-of-the-art hMetis [10] and hMetis − Kway [11] in terms of 2-, 8-, 16-, and 32-way partitioning results. The runtime of LR/ESC − PM is also comparable to that of hMetis algorithms.[9]

[9]A direct runtime comparison between our algorithms and hMetis algorithms is not possible due to the use of different machines. It is reported in [40] that hMetis 2-way results are from Pentium Pro 4-way SMP at 200 MHz. Other hMetis − Kway results [11] are from Pentium II at 300 MHz.

VI. Conclusion

We presented a new efficient multilevel bottom-up clustering algorithm called ESC, which exploits global connectivity information, called edge separability, to guide the clustering process. Our experimental results demonstrate the effectiveness of using edge separability in the context of multilevel partitioning framework for cutsize minimization. Our recent studies on performance driven partitioning [23], [27], [29] also demonstrate that ESC clustering is well suited for constructing a circuit hierarchy for effective performance optimization. In addition, we observe from our recent studies on multilevel placement [12] that ESC is effective in constructing a circuit hierarchy for effective wirelength optimization.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments. The authors would like to thank C.-C. Chang and T. Kong of the Computer Science Department, UCLA.

References

[1] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum flow problem," *J. ACM*, pp. 921–940, 1988.

[2] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs," *SIAM J. Discrete Math.*, pp. 54–66, 1992.

[3] W. Sun and C. Sechen, "Efficient effective placements for very large circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1993, pp. 170–177.

[4] J. Cong and M. Smith, "A parellel bottom-up clustering algorithm with applications to circuit partitioning VLSI design," in *Proc. ACM Design Automation Conf.*, 1993, pp. 755–760.

[5] D. J. Huang and A. B. Kahng, "When clusters meet partitions: New density-based methods for circuit decomposition," in *Proc. Eur. Design Test Conf.*, 1995, pp. 60–64.

[6] T. K. Ng, J. Oldfield, and V. Pitchumani, "Improvements of a mincut partition algorithm," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1987, pp. 470–473.

[7] Y. C. Wei and C. K. Cheng, "Ratio cut partitioning for hierarchical designs," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 911–921, July 1991.

[8] H. Shin and C. Kim, "A simple yet effective technique for partitioning," *IEEE Trans. VLSI Syst.*, vol. 12, pp. 380–386, Sept. 1993.

[9] D. M. Schuler and E. G. Ulrich, "Clustering and linear placement," in *Proc. ACM Design Automation Conf.*, 1972, pp. 50–56.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. ACM Design Automation Conf.*, 1997, pp. 526–529.

[11] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. ACM Design Automation Conf.*, 1999.

[12] T. Chan, J. Cong, T. Kong, and J. Shinnel, "Multilevel optimization for large-scale circuit placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 171–176.

[13] K. Nabors and J. White, "Fastcap: A multipole accelerated 3-D capacitance extraction program," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1447–1459, Nov. 1991.

[14] M. Edahiro and T. Yoshimura, "New placement and global routing algorithms for standard cell layouts," in *Proc. ACM Design Automation Conf.*, 1990, pp. 642–645.

[15] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," in *Proc. ACM Design Automation Conf.*, 1993, pp. 213–218.

[16] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu, "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1997, pp. 441–446.

[17] E. L. Lawler, K. N. Levitt, and J. Turner, "Module clustering to minimize delay in digital networks," *IEEE Trans. Computer-Aided Design*, vol. C-18, pp. 47–57, Jan. 1966.

[18] R. Murgai, R. K. Brayton, and A. S. Vincentelli, "On clustering for minimum delay/area," in *Proc. IEEE Int. Conf. Computer-Aided Design*, vol. 10, Jan. 1991, pp. 6–9.

[19] R. Rajaraman and D. F. Wong, "Optimal clustering for delay minimization," in *Proc. ACM Design Automation Conf.*, 1993, pp. 309–314.

[20] H. Yang and D. F. Wong, "Circuit clustering for delay minimization under area and pin constraints," in *Proc. Eur. Design Test Conf.*, 1995, pp. 65–70.

[21] P. Pan, A. K. Karandikar, and C. L. Liu, "Optimal clock period clustering for sequential circuits with retiming," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 489–498, June 1998.

[22] J. Cong, H. Li, and C. Wu, "Simultaneous circuit partitioning/clustering with retiming for performance optimization," in *Proc. ACM Design Automation Conf.*, 1999, pp. 460–465.

[23] J. Cong, S. K. Lim, and C. Wu, "Performance driven multi-level and multiway partitioning with retiming," in *Proc. ACM Design Automation Conf.*, 2000, pp. 274–279.

[24] S. Hauck and G. Borriello, "An evaluation of bipartitioning techniques," in *Proc. Conf. Adv. Res. VLSI*, 1995, pp. 383–402.

[25] C. J. Alpert, L. W. Hagen, and A. B. Kahng, "A hybrid multilevel/genetic approach for circuit partitioning," in *Proc. ACM/SIGDA Phys. Design Workshop*, 1996, pp. 100–105.

[26] C. J. Alpert, D. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM Design Automation Conf.*, 1997, pp. 530–533.

[27] J. Cong and S. K. Lim, "Performance driven multiway partitioning," in *Proc. Asia South Pacific Design Automation Conf.*, 2000, pp. 441–446.

[28] ——, "Edge separability-based circuit clustering with application to circuit partitioning," in *Proc. Asia South Pacific Design Automation Conf.*, 2000, pp. 429–434.

[29] ——, "Physical planning with retiming," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 2–7.

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

[31] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.

[32] C. J. Alpert and A. B. Kahng, "A general framework for vertex ordering with applications to netlist clustering," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1994, pp. 652–657.

[33] H. Nagamochi and T. Ibaraki, "Linear time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph," *Algorithmica*, pp. 583–596, 1992.

[34] A. Frank, "On the Edge Connectivity Algorithm of Nagamochi and Ibaraki," Labaratoire Artemis, IMAG, Université J. Fourier, Grenoble, France, Tech. Rep. 1994.

[35] S. Fujishige, "Another simple proof of the validity of Nagamochi and Ibaraki's minimum cut algorithm and Queyrannes extension to symmetric submodular function minimization," Forschungsinstitut für Diskrete Math., Univ. Bonn, Bonn, Germany, Tech. Rep. 1994.

[36] M. Stoer and F. Wagner, "A simple min cut algorithm," in *Proc. Eur. Symp. Algorithms, Lecture Notes Comput. Sci.*, vol. 855, 1994, pp. 141–147.

[37] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proc. Int. Symp. Phys. Design*, 1998, pp. 80–85.

[38] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Math. Dokl.*, pp. 1277–1280, 1970.

[39] C. Fiduccia and R. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM Design Automation Conf.*, 1982, pp. 175–181.

[40] C. Alpert. (1999). [Online]. Available: http://vlsicad.cs.ucla.edu/~cheese/errata.html

[41] J. Cong and S. K. Lim, "Multiway partitioning with pairwise movement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1998, pp. 512–516.

Jason Cong (S'88–M'90–SM'96–F'00) received the B.S. degree from Peking University, Beijing, China, in 1985, and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign, in 1987 and 1990, respectively, all in computer science.

Currently, he is a Professor and Co-Director of the VLSI CAD Laboratory in the Computer Science Department, University of California, Los Angeles. He has been appointed as a Guest Professor at Peking University since 2000. He has published over 170 research papers and led over 20 research projects supported by the Defense Advanced Research Projects Agency, the National Science Foundation, the Semiconductor Research Corporation (SRC), and a number of industrial sponsors in these areas. His research interests include layout synthesis and logic synthesis for high-performance low-power VLSI circuits, design and optimization of high-speed VLSI interconnects, field-programmable gate array (FPGA) synthesis, and reconfigurable computing.

Prof. Cong has served as the General Chair of the 1993 ACM/SIGDA Physical Design Workshop, the Program Chair and General Chair of the 1997 and 1998 International Symposia on FPGAs, respectively, and on program committees of many VLSI CAD conferences, including the Design Automation Conference, International Conference on Computer-Aided Design, and International Symposium on Circuits and Systems. He is an Associate Editor of *ACM Transactions on Design Automation of Electronic Systems* and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. He received the Best Graduate Award from Peking University, in 1985, and the Ross J. Martin Award for Excellence in Research from the University of Illinois, in 1989. He received the Research Initiation and Young Investigator Awards from the National Science Foundation, in 1991 and 1993, respectively. He received the Northrop Outstanding Junior Faculty Research Award from the University of California, in 1993, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award in 1995. He received the ACM Recognition of Service Award in 1997, the ACM Special Interest Group on Design Automation Meritorious Service Award in 1998, and the Inventor Recognition and Technical Excellence Awards from the SRC, in 2000 and 2001, respectively.

Sung Kyu Lim (S'94–M'00) received the B.S., M.S., and Ph.D. degrees in computer science from the University of California, Los Angeles (UCLA), in 1994, 1997, and 2000, respectively.

From 2000 to 2001, he was a Post-Doctoral Scholar at UCLA, and a Senior Engineer at Aplus Design Technologies, Inc. He joined the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, as an Assistant Professor in August 2001 and the College of Computing as an Adjunct Assistant Professor in September 2002. He is currently the Director of the Georgia Tech Computer-Aided Design Laboratory. His affiliation with the research centers at Georgia Institute of Technology includes the Center for Research in Embedded Systems and Technology, the Center for Experimental Research in Computer Systems, and the Packaging Research Center. His primary research focus is on the development of physical design automation algorithms and tools for high-performance, low-power, and reliable computing systems targeting system-on-chip, system-on-package, quantum-cell automata, and polymorphic computing architecture technologies.

Dr. Lim has been on the advisory board of ACM/SIGDA since 2003. He is currently serving on the Technical Program Committees of the IEEE International Symposium on Circuits and Systems, the ACM Great Lakes Symposium on VLSI, and the IEEE International Conference on Computer Design. He has been awarded the ACM/SIGDA Design Automation Conference Graduate Scholarship in June 2003 for his research on three-dimensional mixed-signal system-on-package layout.