A Fast Hierarchical Quadratic Placement Algorithm

Gi-Joon Nam, *Member, IEEE*, Sherief Reda, *Student Member, IEEE*, Charles J. Alpert, *Senior Member, IEEE*, Paul G. Villarrubia, and Andrew B. Kahng, *Member, IEEE*

***Abstract*—Placement is a critical component of today's physical-synthesis flow with tremendous impact on the final performance of very large scale integration (VLSI) designs. Unfortunately, it accounts for a significant portion of the overall physical-synthesis runtime. With the complexity and the netlist size of today's VLSI design growing rapidly, clustering for placement can provide an attractive solution to manage affordable placement runtimes. However, such clustering has to be carefully devised to avoid any adverse impact on the final placement solution quality. This paper presents how to apply clustering and unclustering strategies to an analytic top-down placer to achieve large speedups without sacrificing (and sometimes even enhancing) the solution quality. The authors' new bottom-up clustering technique, called the best choice (BC), operates directly on a circuit hypergraph and repeatedly clusters the globally best pair of objects. Clustering score manipulation using a priority-queue (PQ) data structure enables identification of the best pair of objects whenever clustering is performed. To improve the runtime of PQ-based BC clustering, the authors proposed a lazy-update technique for faster updates of the clustering score with almost no loss of the solution quality. A number of effective methods for clustering score calculation, balancing cluster sizes, handling of fixed blocks, and area-based unclustering strategy are discussed. The effectiveness of the resulting hierarchical analytic placement algorithm is tested on several large-scale industrial benchmarks with mixed-size fixed blocks. Experimental results are promising. Compared to the flat analytic placement runs, the hierarchical mode is 2.1 times faster, on the average, with a 1.4% wire-length improvement.**

***Index Terms*—Clustering, physical design, placement, very large scale integration (VLSI).**

I. Introduction

THE TASK of very large scale integration (VLSI) placement is to assign exact locations to various circuit components within the chip area. It typically involves optimizing a number of objectives such as wire length, timing, and power. The solution of placement has a significant impact on the final performance of the design, thus being considered as one of most critical tasks in the physical layout synthesis. However, the placement itself is an extremely computation-intensive process, which accounts for a significant portion of the overall physical-synthesis runtime. With today's multimillion component designs, one iteration of a physical-synthesis flow can easily take a few days. This length of turnaround time represents a serious obstacle to the productive development cycle in today's competitive market. The objective of this work is to speed up today's placement process via an effective clustering technique, particularly on large-scale designs, with no loss of the solution quality.

Clustering has been applied in several applications in VLSI computer-aided design such as verification, extraction, global routing, partitioning, and placement. Clustering is useful for reducing the problem size so that the optimization can be run more efficiently. Recent years have seen a continued explosion in the size of placement problems that need to be optimized. For example, the largest benchmark in the 1998 International Symposium on Physical Desig (ISPD) benchmark suite [1] has 210 000 gates. On the 2005 ISPD benchmark suite [19], some designs have 2 500 000 gates. Despite the availability of ever improving computing resources, placing these large designs is increasingly challenging. Modern placers must be able to produce high-quality solutions and scale to large design sizes. It is not unreasonable to expect a placer to be able to succeed on designs with five million gates, and this ceiling will continue to rise as the design complexity increases.

Despite this increasing complexity, most placers still run flat. Capo [6] and FengShui [25] are multilevel partitioning placement algorithms, but the entire design is still dissolved at each cut. Hu and Marek-Sadowska [14] applied clustering to the Capo placer and showed that runtime speedups can be achieved with only a small degradation of the wire length. However, they were limited to just one level of clustering hierarchy, i.e., fine granularity clustering, to minimize potential quality degradation. A notable exception is the approach of Chan *et al.* [7], [8]. They apply significant amount of clustering in their multilevel mPL placement flow. The clustering makes their nonlinear optimization process practical by reducing the size of problem. Without clustering, the runtime would be prohibitive. Clustering has also been applied in the simulated-annealing-based placement [21]. The primary reason for the absence of clustering in placement is that a poor clustering and unclustering strategy could yield severe degradation in the quality of results. Handling boundary conditions from dividing into ever smaller bins and handling large fixed blocks can make clustering lead to a potentially bad solution that one cannot recover from.

This paper shows how hierarchical clustering and unclustering techniques can be integrated into CPLACE [2], which has been used in the production of A large number of real designs. CPLACE contains a flat analytic placement algorithm called analytic top–down placement (ATP). The integration of hierarchical clustering and unclustering techniques within ATP results in a new enhanced multilevel global placement algorithm

Manuscript received June 24, 2005; revised September 19, 2005. This paper was recommended by Associate Editor P. H. Madden.

G.-J. Nam, C. J. Alpert, and P. G. Villarrubia are with the Austin Research Laboratory, International Business Machines Corporation, Austin, TX 78758 USA.

S. Reda and A. B. Kahng are with the University of California, San Diego, CA 92093 USA.

Digital Object Identifier 10.1109/TCAD.2006.870079

called the hierarchical ATP (hATP). We show that: 1) the hATP leads to a large speedup for the flat global placement and 2) the solution quality is not only maintained, but often improved. We also show how the degree of clustering and unclustering trades off the solution quality for the runtime. Preliminary experimental results are promising. Compared to the flat ATP algorithm, the hATP is 2.1 times faster, on the average, with a 1.4% wire-length improvement.

The rest of the paper is organized as follows. In Section II, we provide a brief overview of the flat ATP algorithm implemented within CPLACE. Section III reviews related works on clustering in placement and the motivation for this work. Particularly, Section III-B presents the new best choice (BC) clustering algorithm and the lazy-update speedup technique. Section IV discusses the overall flow of the hierarchical version of the ATP algorithm with clustering/unclustering techniques. The effectiveness of the proposed method is demonstrated through extensive experiments in Section V, finally, conclusions are presented in Section VI.

II. ATP Overview

The proposed hATP algorithm integrates clustering and unclustering within a flat quadratic placer that we call the ATP. Hence, we begin with a review of the ATP. The ATP global placement algorithm is similar to the method reported by Vygen [23]. It is a top-down analytic placement algorithm with geometric partitioning.

The objective of an analytic quadratic placement is to minimize quadratic wire lengths that can be formulated into

$$\text{minimize } \phi(\vec{x},\vec{y}) = \sum_{i>j} w_{ij}\left[(x_i-x_j)^2+(y_i-y_j)^2\right] \quad (1)$$

where $\vec{x}=[x_1,x_2,\ldots,x_n]$ and $\vec{y}=[y_1,y_2,\ldots,y_n]$ are the coordinates of the movable cells $\vec{v}=[v_1,v_2,\ldots,v_n]$ and w_{ij} is the weight of the net connecting v_i and v_j. The optimal solution is found by solving one linear system for $\vec{x}$ and one for $\vec{y}$ using techniques such as the successive over-relaxation (SOR) or the conjugate gradient (CG). Quadratic placement only works on a placement with fixed objects (anchor points). Otherwise, it will simply produce a degenerate solution where all cells are on top of each other on a single point.

Although the solution of (1) provides a placement solution with an optimal squared wire length, the solution will have lots of overlapping cells. To remove overlaps, either partitioning [18], [22], [23] or density-driven additional forces [13] can be applied. We adopt a geometric four-way partitioning [23]. A four-way partitioning, or quadrisection, is a function : $f: V \rightarrow i \in \{0,1,2,3\}$ where i represents an index for one of the subregions or bins B_0, B_1, B_2, B_3. The assignment of cells to bins needs to satisfy the capacity constraint for each bin. With the given cell locations determined by quadratic optimization, the four-way geometric partitioning tries to minimize the weighted total cell movements defined as

$$\sum_{v\in V} \text{size}(v)\cdot d\left((x_v,y_v),B_{f(v)}\right) \quad (2)$$

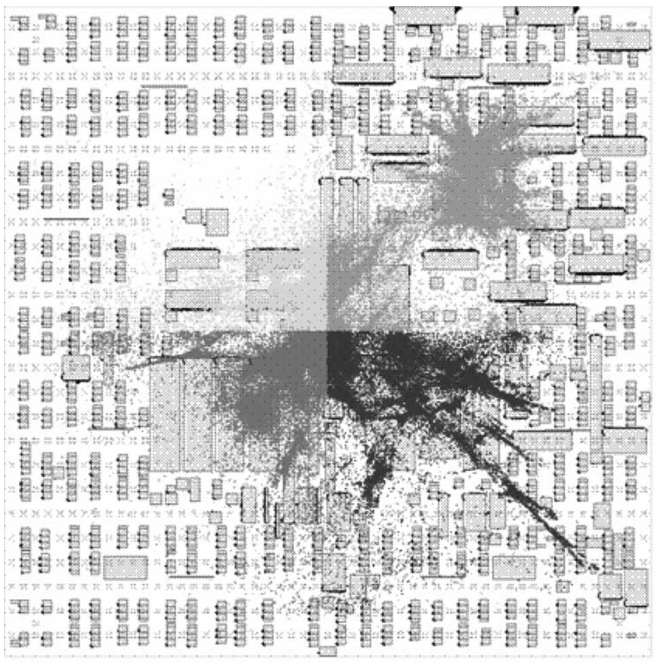
Fig. 1. Geometric partitioning result in flat ATP.

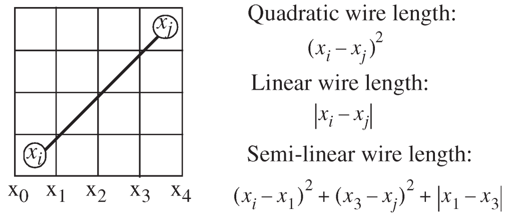

Fig. 2. Net splitting and its wire-length modeling.

where v is a cell, (x_v,y_v) is the location of cell v from a quadratic solution, and $B_{f(v)}$ refers to one of four bins that cell v is assigned to. The distance term $d((x,y),B_i)$ with $i\in\{0,1,2,3\}$ is the Manhattan distance from coordinate (x,y) to the nearest point to the bin B_i. The distance is weighted by the size of cell, size(v). This minimum total distance objective function is quite different from the traditional min-cut objective of partitioning because, in this geometric partitioning formulation, the netlist is not considered at all. The intuition of the new objective function is to obtain the partitioning solution with the minimum perturbation to the previous quadratic-optimization solution. This geometric partitioning is a nondeterministic polynomial-time (NP)-hard problem. To find a near-optimal solution efficiently, Vygen [23] proposes an efficient linear time algorithm to solve a relaxed fractional minimum movement partitioning problem. Fig. 1 shows the example of a geometric partitioning solution.

A quadrisection allows for the definition of a placement level. At level k, there are 4^k placement subregions or bins. For each bin, the process of quadratic optimization and the subsequent geometric partition are repeated until each subplacement region contains a trivial number of objects.

The ATP applies the net-splitting technique [23] during bin refinement. It can be best explained with an example. In Fig. 2, cells x_i and x_j are connected by a single net and located at the

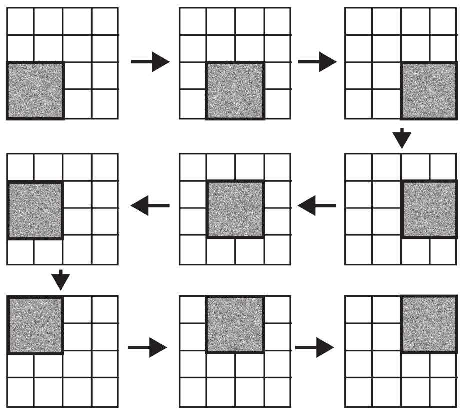
Fig. 3. One pass of overlapping repartitioning.

lower-left and the upper-right bins, respectively. Considering horizontal optimization, the net is split into three segments, (x_i, x_1), (x_1, x_3), and (x_3, x_j), where x_1 and x_3 are bin-boundary coordinates. Then, during the subregion quadratic optimization, only $(x_i - x_1)^2$ and $(x_3 - x_j)^2$ terms are optimized. Note that the term $|x_1 - x_3|$ is a constant. The net-splitting technique has several advantages: 1) A net is modeled by a semiquadratic wire length, as opposed to a pure quadratic wire length; 2) cells are guaranteed to be located within the bin boundary after quadratic optimization, allowing for simultaneous quadratic optimizations for subregions; and 3) the quadratic optimization becomes faster as the placement level increases due to more boundary constraints. The semilinear wire-length model, which falls between quadratic and linear wire lengths, is beneficial because a quadratic wire-length model tends to penalize long wires exceedingly as pointed out in [18].

Often, cells in a given bin may overlap so much that obtaining a geometric partitioning is difficult. In this case, even the geometric partitioning algorithm needs to make arbitrary decisions to partition cells, usually leading to inferior placement solutions. To prevent this situation, a center-of-mass constraint is added into an original quadratic formulation [18], [23]. The idea is still to minimize quadratic wire lengths but with the center of mass of populated cells being enforced toward the center of the subregion. The new center-of-mass point can be literally the center of the subregion [18] or somewhere between the original center-of-mass point and the center of the subregion [23]. In either case, the new quadratic-optimization solution distributes cells better and enables geometric partitioning to find a higher quality partitioning solution.

At each placement level, the ATP applies the local refinement technique called repartitioning (also known as reflow). Repartitioning consists of applying an arbitrary placement algorithm (quadratic, partitioning-driven, or even simulated annealing) on each 2×2 subproblem instance in a sequential manner. Huang and Kahng [15] introduced an overlapping window technique (Fig. 3) that shifts a placement window one column/row at a time, yielding an overlapping pattern. With $n \times n$ subregions, one pass of an overlapping technique consists of $(n-1)^2$ repartitioning operations. In general, an overlapping technique requires significant CPU time but produces much better placement solutions. The fundamental reason why repartitioning can improve wire lengths of placement is that new cell locations and partitioning results of outside subregions (i.e., prior repartitioning results) are immediately reflected for the current 2×2 window under consideration through terminal propagation techniques.

III. Clustering for VLSI Placement

Clustering is one of the most important algorithmic contributions to the hATP algorithm proposed in this paper. This section first reviews the previous clustering techniques and a new bottom-up clustering technique called the BC method is presented.

Circuit clustering is an attractive solution to manage the runtime and the quality of placement results on large-scale VLSI designs. Naturally, it has a long history of research activity [3]–[5], [7], [9], [11], [12], [14], [16], [17], [20], and [21].

In terms of the interactions between clustering and placement, the prior study can be classified into two categories. The first category of clustering in VLSI placement uses transient clustering as part of the core placement algorithm [6], [24], [25]. In these approaches, the act of clustering and unclustering is generally part of the internal placement-algorithm iterations. For example, in multilevel partitioning (MLP)-based placers [16], a cluster hierarchy is first generated, followed by a sequence of partitioning and unclustering. A partitioning result of a prior clustered netlist is projected to the next level through unclustering, which becomes a seed for the subsequent partitioning. Typically, several partitioning attempts are executed, thereby providing for multiple clustering and unclustering operations as part of the inner loop of the placement algorithm. For further optimization, concepts such as V-cycling [16] have been introduced where multiple clustering operations occur at each level of the hierarchy.

The second category of clustering in VLSI placement involves persistent clusters. In this case, the cluster hierarchy is generated at the beginning of a placement in order to reduce the size of a problem. The coarsened netlist is then presented to the placer [14]. Usually, the clustered objects will be dissolved at or near the end of a placement process, with a “clean-up” operation applied to the uncoarsened netlist to improve the results. In some cases, these approaches take the opportunity to uncluster and/or recreate clusters at strategic points in a placement flow [9]. However, in these methods, it can be argued that the clustering algorithm itself is not part of the core placement algorithm but rather a preprocessing step that produces a smaller/simpler netlist structure for the placement algorithm. For instance, in relatively time-consuming simulated-annealing placement [21], a significant runtime speedup can be achieved with persistent clustering.

We employ a semipersistent clustering strategy for our hierarchical placement method. Semipersistent clustering falls into the second category. Persistent clustering offers significant runtime improvements at the expense of the quality of the final placement solution. This problem is particularly magnified as more clustering operations are performed. Another problem associated with persistent clustering is the control of physical cluster sizes. During the placement flow, the size of clustered objects may be too large relative to the decision granularity, which results in the degradation of the final placement solution quality. Therefore, the goal of our semipersistent clustering is to address these two deficiencies. First, we seek to generate high-quality clustering solution so that any potential loss of the final placement solution quality is minimized (or prevented). Second, we take advantage of the hierarchical nature of clustering so that the clustered objects are dissolved slowly during the placement flow. At the early stage of the placement algorithm, a global optimization process is performed on the highly coarsened netlist while local optimization/refinement can be executed on the almost flattened netlist at a later stage.

A. Previous Studies in Clustering

We now review some of the relevant literature on clustering. In edge coarsening (EC) [3], [16], objects are visited in a random order, and only a set of unmatched adjacent objects (i.e., objects that have never been visited or clustered before) is considered for each object u. Among these objects, the one with the largest weight is clustered to u. In EC, a hyperedge of k pins is assigned a weight of $1/(k-1)$. Karypis and Kumar [17] modified the EC scheme and proposed the first-choice (FC) clustering approach. In the FC approach, similar to EC, objects are visited in a random order. But for each object u, all objects that are adjacent to u are considered, regardless of their matching status. Again, the object with the largest weight is matched to u. Thus, a clustered object with multiple layers of clustering hierarchy can be formed. To limit the cluster size, the FC approach stops clustering when the coarsened netlist reaches a certain threshold.

In another approach, Cong and Lim [12] transform a given hypergraph into a graph by decomposing every k-pin hyperedge into a clique, with an edge weight $1/(k-1)$. Then, they: 1) rank edges according to a connectivity-based metric using a priority-queue (PQ) data structure; 2) cluster two objects with the highest ranking edge if their sizes do not exceed a certain size limit; and 3) update the circuit netlist and the PQ structure accordingly. We note that decomposing a hyperedge into a clique can cause a discrepancy in edge weights once any two objects of a k-pin hyperedge are clustered. This discrepancy leads to incorrect edge weights as demonstrated by the following example.

Example 1: Assume that two objects $v_1 \in e$ and $v_2 \in e$ are clustered, where e is a k-pin hyperedge. In Karypis and Kumar's scheme [17], the clustering score of any other objects in e becomes $1/(k-1-1)$; while in Cong and Lim's scheme [12], the clustering score stays the same with $1/(k-1)$. This edge-weight discrepancy occurs because the transformation of a hyperedge to a clique of edges is performed only once before clustering starts.[1]

Chan *et al.* [7] uses a connectivity-based approach similar to that of Cong and Lim [12]. The difference is that the area of a clustered object is included in the objective function to produce a more balanced clustering. The inclusion of the cluster size

```
Input: Flat Netlist
Output: Clustered Netlist
1. Until target object number is reached:
    2. Find closest pair of objects
    3. Cluster them
    4. Update netlist
```

Fig. 4. Bottom–up clustering.

```
Input: Flat Netlist
Output: Clustered Netlist
Phase I. Priority-queue PQ Initialization:
  1. For each object u:
  2. Find closest object v, and its associated clustering score d
  3. Insert tuple (u, v, d) into PQ with d as key
Phase II. Clustering:
  1. While target object number is not reached and top tuple's
  score d > 0:
    2. Pick top tuple (u, v, d) of PQ
    3. Cluster u and v into new object u'
    4. Update netlist
    5. Find closest object v' to u' with its clustering score d'
    6. Insert tuple (u', v', d') into PQ with d' as key
    7. Update clustering scores of all neighbors of u'
```

Fig. 5. BC clustering algorithm.

in an objective function is originally proposed in [20]. Another recent approach [14] proposes fine-grain clustering particularly targeted for improving the runtime in mincut-based placement. The approach decomposes hyperedges into cliques and uses a connectivity-based net-weighting scheme similar to [20]. A PQ is used to rank all the edges according to the calculated net weights. Clustering proceeds in an iterative fashion. At each iteration, clustering is allowed only if both target objects have never been visited before during the same iteration. A cluster is typically limited to a few (two to three) objects; thus, the name fine-grain clustering.

We think that the general drawbacks of previous approaches are as follows.

1) The hypergraph-to-clique transformation [7], [12], [14], [21] leads to a discrepancy in edge weights and increases the size of required PQ.
2) Pass-based clustering methods (i.e., clustering iterations) [3], [14], [16] that disallow an object to be revisited during the same iteration lead to suboptimal choices because an object might be prevented from getting clustered to its best neighbor.
3) Non-PQ-based implementations [17] lead to suboptimal clustering choices due to the lack of a global picture of clustering sequences.

Given this brief overview of the related literature, we next describe our clustering method, which avoids the general drawbacks of previous methods.

B. BC Bottom-Up Clustering

The general concept of the bottom-up clustering and the BC clustering are given in Figs. 4 and 5. The key idea of the BC clustering is to identify the globally best pair of objects to cluster by managing a PQ data structure with the clustering score as a comparison key. PQ management naturally

[1]However, note that the clustered object $\{v_1, v_2\}$ will have a score of $2/(k-1)$ compared to other objects on the same hyperedge e [12].

provides an ideal clustering sequence, and it is always guaranteed that the two objects with the best clustering score will be clustered.

The degree of clustering is controlled by a target clustering ratio α. The target number of objects is determined by the original number of objects divided by the target clustering ratio α. The clustering operation is simply repeated until the overall number of objects becomes the target number of objects. For example, a target clustering ratio of $\alpha = 10$ indicates that the clustered netlist will have one tenth of the number of objects in the original netlist.

The challenges associated with BC clustering are as follows:

1) using an efficient and effective clustering score function, which leads to higher quality placement solutions;
2) accurately handling hyperedges;
3) an efficient netlist and PQ data structure updating after each clustering is performed;
4) controlling the physical size of a clustered object for a more balanced clustering result;
5) handling fixed blocks and attached movable objects around these fixed blocks.

These challenges will be addressed as follows.

C. BC Clustering Score Function

The weight w_e of a hyperedge e is defined as $1/|e|$, which is inversely proportional to the number of objects that are incident to the hyperedge. Given two objects u and v, the clustering score $d(u,v)$ between u and v is defined as

$$d(u,v) = \sum_{e \in E | u,v \in e} \frac{w_e}{(a(u) + a(v))} \tag{3}$$

where e is a hyperedge connecting objects u and v, w_e is a corresponding edge weight, and $a(u)$ and $a(v)$ are the areas of u and v, respectively. The clustering score of two objects is directly proportional to the total sum of edge weights connecting them and is inversely proportional to the sum of their areas. Suppose N_u is the set of neighboring objects to a given object u. We define the closest object to u, denoted $c(u)$, as the neighbor object with the highest clustering score to u, i.e., $c(u) = v$ such that $d(u,v) = \max\{d(u,z) | z \in N_u\}$.

In order to identify the globally closest pair of objects with the best clustering score, a PQ-based implementation is proposed as shown in Fig. 5. The BC algorithm is composed of two phases. In phase 1, for each object u in the netlist, the closest object v and its associated clustering score d are calculated. Then, the tuple (u, v, d) is inserted to the PQ with d as a comparison key. For each object u, only one tuple with the closest object v is inserted. This vertex-oriented PQ allows for more efficient data-structure management than edge-based methods. Phase 1 is a simple PQ initialization step.

In the second phase, the top tuple (u, v, d) in the PQ is picked up (step 2), and the pair of objects (u, v) are clustered creating a new object u' (step 3). The netlist is updated (step 4), the closest object v' to the new object u' and its associated clustering score d' are calculated, and a new tuple (u', v', d') is inserted to the PQ (steps 5–6). Since clustering changes the

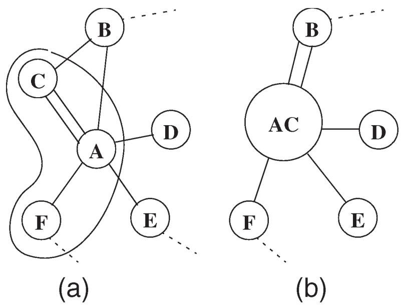

Fig. 6. Clustering a pair of objects A and C.

netlist connectivity, some of previously calculated clustering scores might become invalid. Thus, the clustering scores of the neighbors of the new object u' (equivalently all neighbors of u and v) need to be recalculated (step 7), and the PQ is adjusted accordingly. The following example illustrates clustering score calculation and updating.

Example 2: Assume the input netlist with six objects $\{A, B, C, D, E, F\}$ and eight hyperedges $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{A, E\}$, $\{A, F\}$, $\{A, C\}$, $\{B, C\}$, and $\{A, C, F\}$ as in Fig. 6(a). The size of each object is 1. By calculating the clustering score of A to its neighbors, we find that $d(A, B) = 1/4$; $d(A, C) = 2/3$; $d(A, D) = 1/4$; $d(A, E) = 1/4$; and $d(A, F) = 5/12$. $d(A, C)$ has the highest score, and C is declared as the closest object to A. Since $d(A, C)$ is the highest score in the PQ, A will be clustered with C and the circuit netlist will be updated as shown in Fig. 6(b). With a new object AC introduced, the corresponding cluster scores will be $d(AC, F) = 1/3$, $d(AC, E) = 1/6$, $d(AC, D) = 1/6$, and $d(AC, B) = 1/3$.

We can summarize the main advantages of our BC clustering methodology as follows.

1) Clustering will always be performed on the globally best pair of objects.
2) Direct hyperedge handling, without converting hyperedges into clique or star models [12], [14], is performed.
3) Object-based PQ manipulation. The size of the PQ is equal to the number of objects (rather than the number of clique edges [12]) in the netlist by recording only the closest neighbor per object, leading to more efficient PQ management.

As will be demonstrated in Section V, the BC scheme can produce high-quality clustering results for analytic placement. However, the overall clustering runtime is not yet competitive to other faster clustering algorithms such as the EC [16] or the FC [17] approaches. To improve the runtime of the BC, we propose to update the PQ in a lazy fashion.

D. Lazy-Update Speedup Technique for BC Clustering

Analyzing the runtime characteristic of the BC clustering algorithm of Fig. 5 (step 7), turned out to be the most time-consuming task. To update the score PQ after each clustering, each neighbor object of a newly created object needs to be visited to find its new closest object and its clustering score. The closest object of a given target object u can only be found by visiting all the neighbor objects of u. Therefore, updating the

```
Input: Flat Netlist
Output: Clustered Netlist

Phase II. Clustering:
  1. While target object number is not reached and top tuple's
  score $d > 0$:
      2. Pick top tuple ($u$, $v$, $d$) of PQ
      3. If $u$ is marked as invalid, re-calculate closest object $v'$
      and score $d'$ and insert tuple ($u$, $v'$, $d'$) to PQ
      4. else
          5. Cluster $u$ and $v$ into new object $u'$
          6. Update netlist
          7. Find closest object $v'$ to $u'$ with its clustering score $d'$
          8. Insert tuple ($u'$, $v'$, $d'$) into PQ with $d'$ as key
          9. Mark all neighbors of $u'$ as invalid
```

Fig. 7. Lazy-update speedup technique for BC clustering.

clustering scores after a clustering operation (step 7) typically involves two levels of netlist exploration.

However, a statistical analysis of clustering score PQ management reveals the following facts.

1) An object in the PQ might be updated a number of times before climbing up to the top (if ever). Effectively, all the updates but the last one are useless since only the final update determines the final location within the PQ.
2) In 96% of clustering score updates, a new score decreases, i.e., most of time, objects are moving downward the PQ rather than popping up.

Motivated by these observations, we propose lazy-update technique which delays updates of clustering scores as late as possible, thus reducing the actual number of score-update operations on the PQ. More specifically, a lazy update waits until an object reaches the top of the PQ and updates the object's score only if necessary. The modification to the clustering phase (phase 2) is shown in Fig. 7. In step 9 of the modified algorithm, we mark neighbor objects without recalculating their scores to indicate that their scores are invalid. If an object reaches the top of PQ, we check whether it is marked or not. If it is marked (invalid), its new closest object and its score are recalculated and reinserted into the PQ (step 3); otherwise (valid), it is clustered with its precalculated closest object. In the experimental section, we demonstrate that the lazy-update technique can dramatically reduce the clustering runtime with almost no adverse impact on the clustering quality.

E. Cluster Size Growth Control

The presence of the area function in the denominator of (3) provides an indirect way to automatically control the physical sizes of clustered objects, potentially leading to more balanced clustering results. Without such an area control, gigantic clustered objects might be formed by continually absorbing small objects around it. In general, these gigantic clustered objects have a detrimental influence on the quality of the placement solution. Two classes of cluster size-control methods are discussed here: indirect control versus direct control.

1) Indirect Size Control: The cluster size is controlled automatically via a clustering score function as in (3), which is inversely proportional to the size of the cluster object. A more generic form of this approach will be as follows. Given a target object u and its neighbor v, a clustering score between u and v is defined as

$$d(u,v) = \sum_e \frac{w_e}{(a(u)+a(v))^k} \tag{4}$$

where $k \geq 1$, and k can be either fixed number or it can be dynamically adjusted by setting it to $k = \lceil (a(u)+a(v))/\mu \rceil$, where μ is the average cell size multiplied by the clustering ratio α. μ represents the expected average size of clustered objects. Another possibility is to use the total number of pins instead of the object area because, in general, the number of pins in a cluster is well correlated with its cluster size.

2) Direct Size Control: The clustering algorithm can take a more direct approach by imposing a bound constraint on the size of clusters. Given two objects u and v, two methods are proposed as follows.

1) Hard Bound: If the total area $a(u)+a(v) \leq k \cdot \mu$, then accept clustering, else reject it.
2) Soft Bound: If the total area $a(u)+a(v) \leq k \cdot \mu$, then accept clustering, else accept it with a probability equal to $2^{(\mu/(a(u)+a(v)))^k} - 1$, where $k \geq 1$.

With the hard bound, an upper bound on the cluster size is strictly enforced; while with the soft bound, the upper bound is slightly relaxed, where the probability of clustering two objects declines as the sum of areas increases. The parameter k controls the amount of relaxation. The plot of Fig. 8 demonstrates the probability of clustering two objects for various values of k. The x-axis is $(a(u)+a(v))/\mu$, and the y-axis shows the corresponding probability of a clustering occurrence. The hard cluster bound can be incorporated in two ways during the calculation of the closest objects.

1) Method A: Pick the closest object among all neighbors and check if the chosen object satisfies the area constraints.
2) Method B: Pick the closest object only from the set of neighbor objects that satisfy the area constraints.

Basically, method A ensures to choose the object with the highest clustering score despite that it might get rejected due to the area constraint violation later, while method B ensures that the chosen object meets the area constraints, despite that its clustering score might not be the highest among neighbor objects. In method A, if the chosen object does not satisfy the area constraint, the clustering is aborted and these objects are reinserted to a PQ with newly found closest objects. A new top object from a PQ is taken for the next clustering. Therefore, methods A and B produce different clustering sequences. A soft-bound method only makes sense with method A. Empirically, we have found that using method A produces better results than method B. The empirical comparison of different methods of cluster size control will be presented in Section V.

F. Handling Fixed Blocks During Clustering

The presence of fixed blocks in a netlist might alter how clustering is performed. We observe that sometimes, particularly

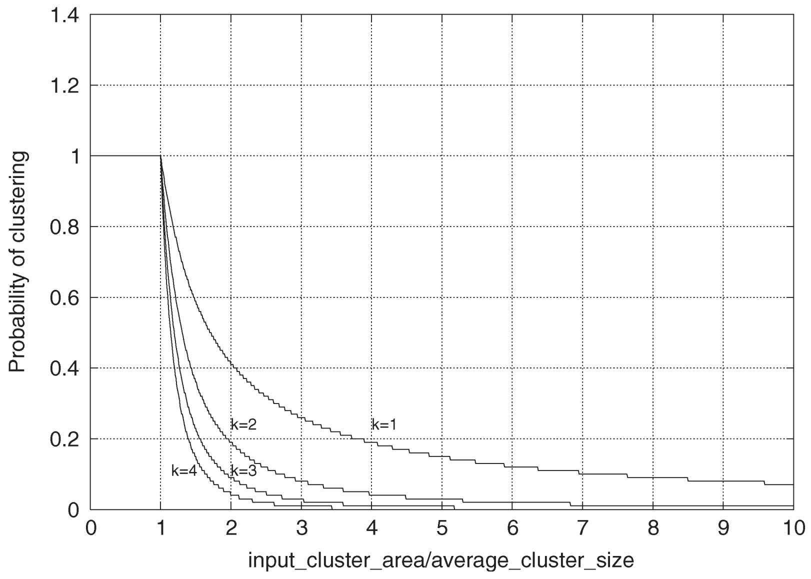

Fig. 8. Probabilistic cluster size-control curves.

when a significant degree of clustering is performed, movable cells directly connected to fixed blocks are being clustered to objects located far away from them. This might cause an adverse impact on timing results as well as on wire lengths. Ideally, movable objects around fixed blocks need to be placed around those directly connected fixed blocks after placement, regardless of the degree of clustering performed. To our knowledge, there has been no prior clustering work that explicitly considers fixed blocks and their neighbor movable objects. We have attempted the following techniques to address this issue:

1) by ignoring all nets connected to fixed objects, since such nets cannot be eliminated by clustering;
2) by ignoring all pins connected to fixed objects, thus altering the weight of nets connecting movable objects and fixed blocks;
3) by including fixed blocks during clustering, only to remove them from clusters after the clustering is over and before the placement process starts;
4) by chaining all movable objects attached to a fixed block by a set of additional artificial nets in order to control their affinity to fixed blocks during placement.

However, none of the aforementioned techniques made distinguishable improvements to final placement results so far, and none of the above techniques is used for the experimental sections. We leave this topic as future study to further explore an effective method of handling fixed blocks during clustering.

IV. hATP Algorithm

In this section, we present a new global placement algorithm based on ATP (with geometric partitioning) with hierarchical clustering and unclustering techniques. The method is called hATP, and the overall flow is summarized in Fig. 9. With a given initial netlist, a coarsened netlist is generated via BC clustering, which is used as a seed for the subsequent ATP global

```
Input: netlist structure
Output: legal placement solution

hATP()
 1. best-choice clustering()
 2. for placement_level =1 to n
 3.    for each bin in current placement_level
 4.          quadratic optimization()
 5.          geometric partitioning()
 6.          area-based selective unclustering()
 7.    end_for
 8.    repartitioning()
 9. end_for
10. uncluster_all()
11. legalization & detailed_placement()
```

Fig. 9. hATP algorithm.

placement. Steps 2–5 are the basic ATP algorithms described in Section II. After quadratic optimization and quadrisection are performed for each bin, an area-based selective unclustering is performed to dissolve large clustered objects (step 6). At the end of each placement level, a repartitioning refinement is executed for local improvement (step 8). Steps 2–9 consist of the main global placement algorithm. If there exist clustered objects after the global placement, those are dissolved unconditionally (step 10) before the final legalization and detailed placement are executed (step 11). As can be seen, the hATP algorithm relies on three strategic components: BC clustering, ATP global placement, and area-based selective unclustering. Each component is briefly described next.

A. Semipersistent BC Clustering

This preprocessing style of clustering helps reduce the overall global placement runtime by producing a smaller/simpler netlist structure. The generated clustering hierarchy is preserved during the most global placement phase only, being dissolved selectively based on the size of objects. Thus, it is called semipersistent clustering. Clustering offers the advantage of significant runtime improvements. However, in order to

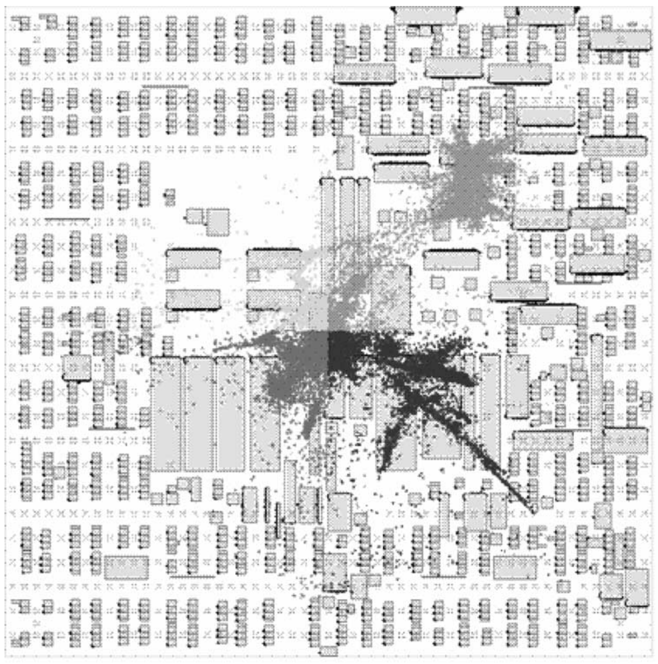
Fig. 10. Geometric partitioning result in hATP.

minimize any potential loss of (or possibly to improve) the quality of final placement solutions, a high-quality clustering technique is absolutely required. To provide effective runtime and quality tradeoffs, the BC clustering algorithm, introduced in Section III, was employed in a semipersistent context. The degree of clustering is controlled by a user-specified parameter, clustering ratio α. Clustering is performed until the final number of movable cells reaches the target number of objects defined as the initial number of movable objects divided by a clustering ratio α. Thus, a higher value of α implies more extensive clustering.

B. Analytic Top-Down Global Placement (ATP)

With the generated clustering hierarchy, the basic ATP global placement algorithm is applied as described in Section II. Fig. 10 shows an example of a quadrisection result of hATP on the same circuit with Fig. 1. For this example, the clustering ratio α is set to 5. Note that the overall cell distribution and the partitioning solution are very similar to that of flat ATP (Fig. 1). This can be attributed to the high-quality BC clustering technique.

C. Area-Based Selective Unclustering

In our semipersistent clustering scenario, the clustering hierarchy is preserved during the most global placement. However, if the size of a clustered object is relatively large to the decision granularity, the geometric partitioning result on this object can affect not only the quality of global placement solution but also the subsequent legalization due to the limited amount of available free space. To address this issue, we employ an adaptive area-based unclustering strategy. For each bin, the size of each clustered object is compared to the available free space. If the size is bigger than the predetermined percentage of the available free space, the clustered object is dissolved. Our empirical analysis shows that with the appropriate threshold value (5%), most clusters can be preserved during the global placement flow with an insignificant loss of wire length. The area-based selective unclustering is another knob to provide a tradeoff between the runtime and the quality of a placement solution. More aggressive unclustering (lower threshold value) produces better wire lengths at the cost of higher CPU time. The detailed tradeoff results will be discussed in the experimental section.

TABLE I
BENCHMARK CHARACTERISTICS

Bench	Cells	Blks	IOs	Nets	Density	Util
AL	270163	4235	14100	292425	44.74%	22.54%
BL	276194	14461	17380	327102	69.37%	20.37%
CL	351056	26713	6360	395918	82.32%	37.59%
DL	425610	14665	17960	465927	56.25%	34.91%
EL	457516	3460	7094	478842	72.12%	51.90%
FL	880410	53481	23255	1010392	74.77%	48.19%
AD	389226	0	35944	401463	87.65%	83.55%
BD	285085	0	13286	309050	87.76%	85.87%
CD	56436	2	1968	57595	57.43%	57.32%

V. EXPERIMENTAL RESULTS

The hATP placement algorithm is implemented within the industrial placement tool CPLACE [2]. To demonstrate the effectiveness of hATP, two classes of experiments are performed:

1) Clustering technique evaluation—BC versus FC within hATP.
2) Hierarchical placement evaluation—flat ATP versus hATP.

For all experiments, several large-scale industrial benchmarks are used, ranging from 56 000 to 1 000 000 objects. The detailed characteristics are presented in Table I with the following information.

1) Cells: the number of placeable (movable) objects.
2) Blocks: the number of fixed blocks, i.e., logic macros, blocked areas, reserved areas for outside modules, etc.
3) IOs: the number of I–O ports.
4) Nets: the number of nets.
5) Density%: the overall design density, e.g., the sum of cell area divided by the total placement area.
6) Util%: the design utilization (density) defined as the total area of placeable cells (not including fixed blocks) divided by the available free space [2]. Note that the design has an abundant free space available for movable cells.

Two sets of benchmarks are used. The benchmarks with names ending with an "L" are low-utilization designs, and those ending with "D" have high design utilization. For all experiments, a workstation with four Intel Xeon 2.40-GHz CPUs, a 512-KB cache, and 6 GB of memory is used.

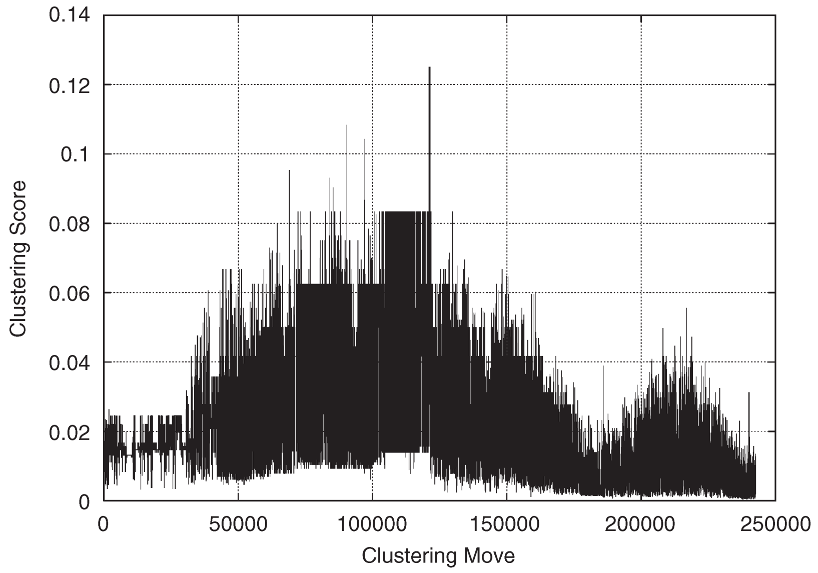

Fig. 11. EC clustering score plot. Total clustering score $= 5301.05$. Clustering runtime $= 9.23$ s.

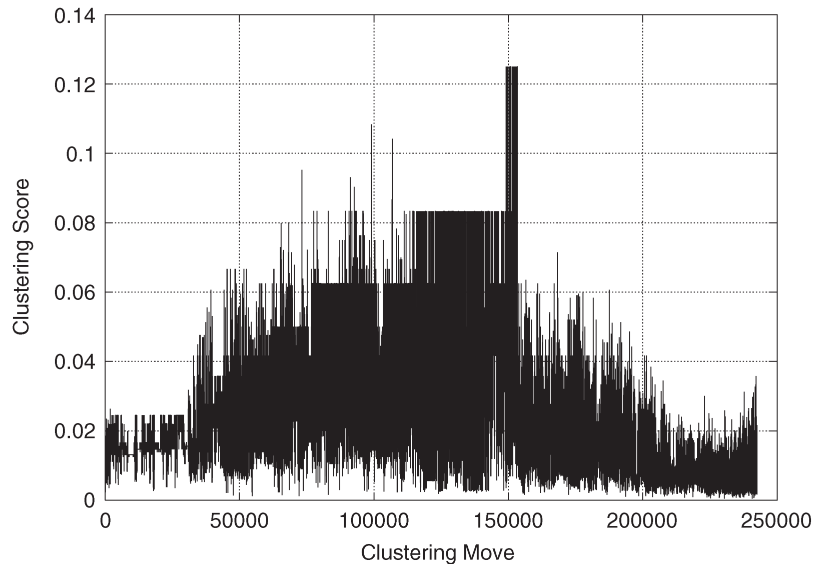

Fig. 12. FC clustering score plot. Total clustering score $= 5612.83$. Clustering runtime $= 9.03$ s.

A. BC Clustering Experiment

In this experiment, we demonstrate that the ATP algorithm produces better placement solutions with the BC approach than with other clustering approaches. To speed up the placement process significantly, a rather high clustering ratio $\alpha = 10$ is used to reduce the number of objects in benchmark by an order of magnitude. For comparison purposes, we implement two approaches: EC [16] and FC [17].[2]

We first construct clustering score plots (3) for a typical benchmark AD, as shown in Figs. 11–14. The x axis represents the sequence of clustering operations, and the y axis is the corresponding clustering score. For example, 10 on the x axis represents the tenth clustering operation. We also compute the total clustering score value of each clustering method. From the figures, the total clustering score of the EC, the FC, and the BC are 5301, 5612, and 6671, respectively. The BC achieves the highest total clustering score among these three methods. With lazy-update speedup technique, we observe that the BC clustering runtime is reduced by 50% with almost no loss in the total clustering score—a negligible drop from 6671 to 6658—as shown in Figs. 13 and 14.

We now investigate how different clustering algorithms can affect the final placement results. Table II presents the final placement wire length of the EC, the FC, and the BC clustering algorithms on the circuits of Table I. All results are normalized with respect to the EC. In the table, "CPU" shows clustering

[2]We also tried the fine-granularity clustering proposed in [14]. However, this method is not devised for an extensive degree of clustering such as $\alpha = 10$. Even with $\alpha = 2$, we observed that the BC produces better placement solutions than fine-granularity clustering. Thus, those results are not included here.

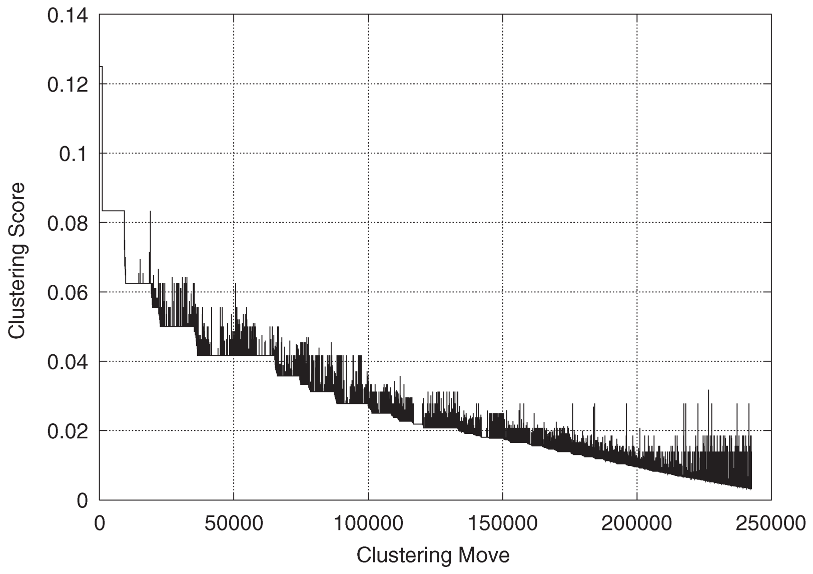

Fig. 13. BC clustering score plot. Total clustering score $= 6671.53$. Clustering runtime $= 97.35$ s.

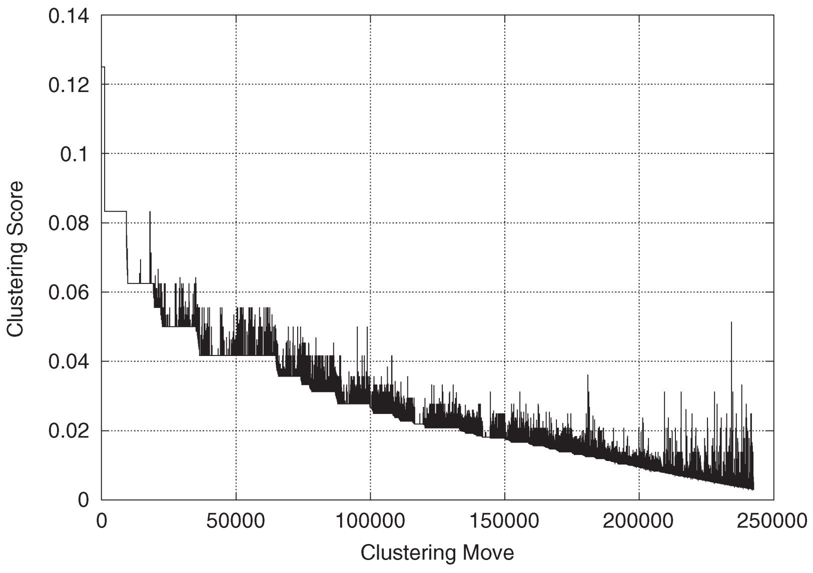

Fig. 14. BC with lazy-update clustering score plot. Total clustering score $= 6658.23$. Clustering runtime $= 49.84$ s.

CPU times with respect to that of the EC and "WL(%)" presents the percentage improvement in half-perimeter wire length (HPWL) over the EC's HPWL. We make the following observations.

1) BC clustering dominates over other standard clustering methods on all benchmarks with an average improvement of 4.3% over the EC and 3.2% over the FC.
2) A lazy update significantly improves BC clustering runtime for all benchmarks with an average runtime reduction of 57% and with almost no impact to the quality of results, only a 0.11% change in the final HPWL.

Fig. 15 shows the runtime breakdown of BC clustering on the largest benchmark FL. From the plot, we immediately notice that the runtime reduction with the lazy update increases as clustering progresses, i.e., as more clusterings are performed, the lazy update becomes more effective.

In the next experiment for clustering evaluation, we examine how the cluster size control affects the quality of placement. We have found that the cluster size control is particularly critical to dense designs (AD, BD, and CD). For sparse benchmarks (the ones ending with "L"), no distinguishable impact has been found with different size-control methods. Three additional area-control methods are implemented and compared to the standard area-control method with (3). In Table III, "Automatic" refers to the method using (4) with $k = 2$. "Hard" and "Soft" refer to the bounding methods of Section III-E2. Both hard and soft bounds are executed with $k = 3$. All size-control methods are implemented within our BC clustering framework. The quality of the solution is measured by the final placement wire length. We also report the maximum and the

TABLE II
CLUSTERING RESULTS. ALL RESULTS ARE NORMALIZED WITH RESPECT TO EC

Bench	EC		FC		BC		BC+LazyUpdate	
	WL(%)	CPU	WL(%)	CPU	WL(%)	CPU	WL(%)	CPU
AL	0.00	1.00	-0.43%	x0.98	3.19%	x9.77	2.10%	x4.02
BL	0.00	1.00	3.30%	x0.94	6.99%	x8.69	6.28%	x3.61
CL	0.00	1.00	2.14%	x0.96	3.43%	x4.99	4.23%	x3.29
DL	0.00	1.00	0.44%	x1.03	2.22%	x9.76	2.07%	x3.84
EL	0.00	1.00	2.37%	x0.98	5.72%	x4.92	6.27%	x3.37
FL	0.00	1.00	-1.14%	x1.04	4.33%	x14.48	4.25%	x4.50
Avg.	**0.00**	**1.00**	**1.11%**	**x0.98**	**4.31%**	**x8.76**	**4.20%**	**x3.77**

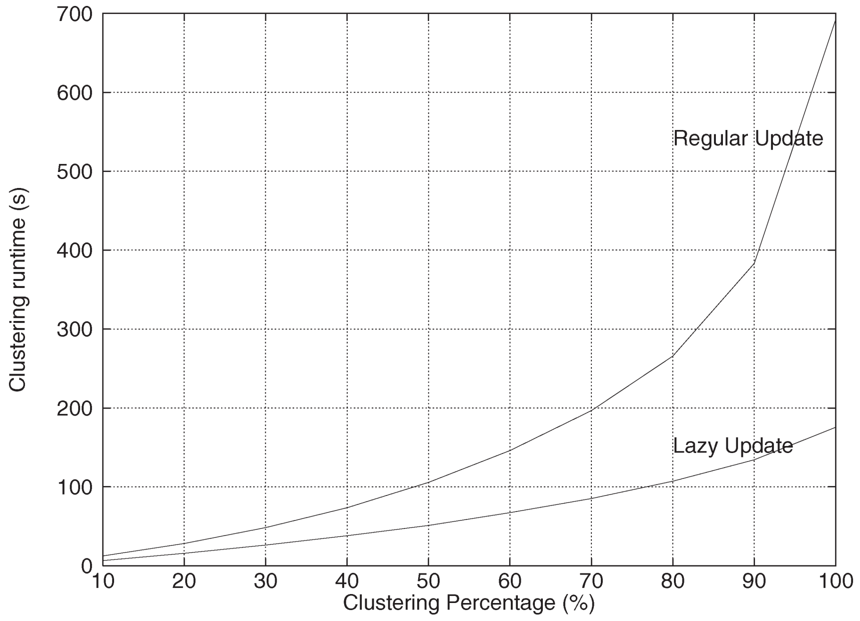

Fig. 15. Runtime breakdown of BC clustering with/without lazy update on benchmark FL.

average cluster sizes after clustering is done. From the table, we observe that a careful control of the cluster size can improve the placement wire length by up to 13%. The results also indicate that a probabilistic control of cluster size, "Soft," produces the best results. We believe that a soft probabilistic control occasionally provide a way to form high-quality but slightly larger clusters, which lead to better results.

B. Hierarchical Placement Experiment

This section presents comparative results between flat (i.e., no clustering) and hATP runs. Figs. 16 and 17 show the normalized wire lengths and the CPU times of the hATP relative to the flat ATP. The values are the average of six sparse circuits ending with "L." The degree of clustering, clustering ratio α, varies from 1 (no clustering) to 20. Wire lengths and CPU times are measured after global placement, not the final placement, because: 1) The focus of this paper is the global placement algorithm and 2) the final wire length could be influenced by the degree of detailed placement performed. However, a half-perimeter wire length was measured after the legalization is performed for a fair comparison. Thus, there exists no overlap of cells.[3] From these data, we observe the following.

1) With $\alpha \leq 7$, the hierarchical clustering actually improves the performance ranging from 0% to 7% on circuit by circuit and 1% on the average. The runtime speedup ranges from $2x$ to $4x$.
2) With $7 < \alpha \leq 10$, virtually the same wire-length placement solutions with those of the flat ATP are achievable in four–five times less CPU time.
3) With $\alpha \geq 10$, the wire length starts to degrade compared to the flat ATP, but the maximum degradation is only limited up to 5%. The runtime speedup seems to saturate to 5–$6x$, even with more clustering. This is because the legalization and the detailed placement CPU times become a new bottleneck to the overall placement time.

Overall, the experimental results demonstrate that the appropriate amount of clustering can help to reduce the wire length with significant CPU-time savings in the ATP algorithm. This is

[3]Note that there is a strong correlation between global placement and final wire lengths. Typically, a better global-placement wire length leads to a better final wire length.

TABLE III
IMPACT OF CLUSTER-SIZE CONTROL ON TOTAL HPWL FOR DENSE DESIGNS

Bench	Stnd			Automatic			Hard			Soft		
	Max	Avg	WL(%)	Max	Avg	WL(%)	Max	Avg	WL(%)	Max	Avg	WL(%)
AD	14823	171.4	0.00	1140	160.4	-0.88%	364	169.9	0.47%	1668	169.6	0.56%
BD	28600	150.0	0.00	1140	114.6	3.71%	405	147.9	5.89%	1520	147.9	4.86%
CD	9060	113.5	0.00	610	109.8	30.05%	280	116.1	29.11%	1075	114.9	34.16%
Avg.	-	-	**0.00**	-	-	**10.96%**	-	-	**11.82%**	-	-	**13.19%**

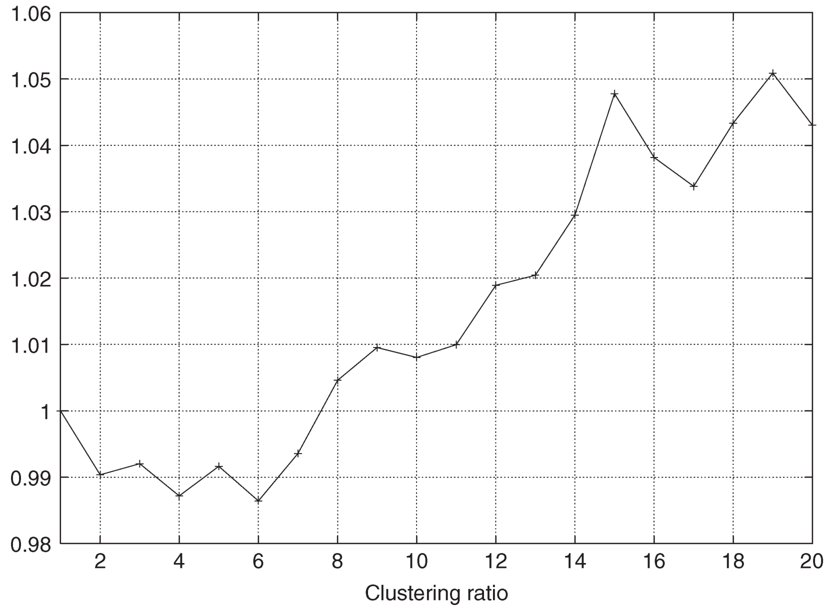

Fig. 16. Normalized hATP placement wire length.

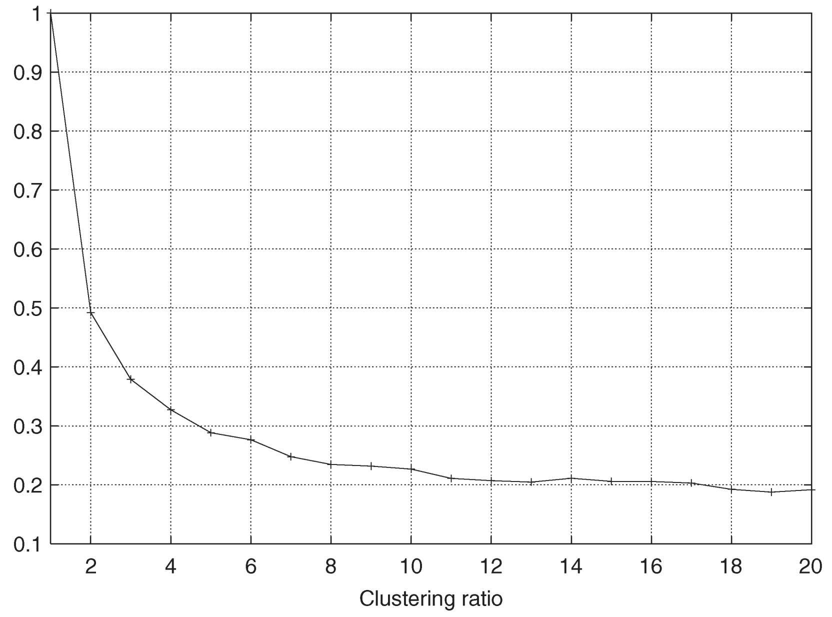

Fig. 17. Normalized hATP placement CPU time.

partly due to the linearization effect of clustering. Once a group of cells is clustered together in the semipersistent clustering flow, those cells will be placed in the same vicinity until the unclustering happens. This reduces the gap between quadratic and linear wire lengths, because the wire lengths among objects in the same cluster are considered to be near zero. However, at the same time, this is why high-quality clustering is required up front.

TABLE IV
COMPARISON BETWEEN FLAT (NO CLUSTERING) AND BC + LAZY-UPDATE CLUSTERING CPLACE RUNS

Bench	WL(%)	CPU	CL-CPU
AL	2.09%	x0.40	1.17%
BL	-4.28%	x0.52	1.35%
CL	3.27%	x0.51	1.14%
DL	0.87%	x0.45	1.35%
EL	1.59%	x0.33	1.10%
FL	1.41%	x0.46	1.68%
AD	8.23%	x0.50	0.98%
BD	-0.34%	x0.47	0.94%
CD	-0.36%	x0.69	0.51%
Avg.	**1.39%**	**x0.48**	**1.14%**

TABLE V
WIRE-LENGTH COMPARISON WITH AREA-BASED SELECTIVE UNCLUSTERING CONTROLS

	1%	5%	10%	15%	20%	25%	30%
AL	0.99	1.00	1.01	1.04	1.06	1.08	1.09
BL	1.01	1.00	1.00	1.01	1.04	1.09	1.15
CL	1.00	1.00	1.00	1.01	1.02	1.02	1.04
DL	1.00	1.00	1.04	1.07	1.07	1.08	1.04
EL	0.99	1.00	1.00	1.08	1.24	1.43	1.53
FL	0.99	1.00	1.05	1.05	1.09	1.07	1.09

Table IV presents the detailed performance of the hATP with clustering ratio $\alpha = 10$. The hATP run with BC–lazy-update clustering speeds up the overall placement by $2.1x$ on the average. More interestingly, with BC clustering, the hATP is able to produce 1.39% better final wire lengths on the average. Column "CL-CPU" shows the portion of the clustering CPU time in the overall CPLACE run. Although BC–lazy-update clustering takes $3.77x$ more CPU time than EC as shown in Table II, it takes only 1.14% of the overall placement CPU time, which is negligible. Please note that the hATP produces better wire lengths than the flat ATP algorithm on six out of nine circuits.

Another important consideration in the hATP is the unclustering strategy. Table V demonstrates that area-based selective unclustering strategy can affect the quality of the solution. The top row represents the unclustering area threshold. If a cluster size is bigger than "x%" of the available free space within a bin, the clustered object is dissolved. The larger the "x%" value, the less the unclustering operation is executed. The wire lengths are normalized to those of the "5%" threshold value. The results show that the more clusters are preserved, the higher potential damage are made to final placement wire lengths. As expected, this contrasts to the fact that more preserved clustered objects lead to a higher runtime speedup.

VI. CONCLUSION

In this paper, we developed a new multilevel ATP algorithm called hATP. It takes advantage of the bottom-up BC clustering algorithm in a semipersistent context to accomplish significant runtime speedup over its flat version without loss of wire length. We also observe that an appropriate amount of clustering actually improves the wire-length quality due to the linearization effect of clustering. We also explored various aspects of clustering/unclustering strategies within the analytic placement, such as the clustering cost function and the area-control methods during clustering and unclustering. We believe that an even higher quality of placement solutions is achievable by a better handling of fixed blocks during clustering, and this remains as a future study.

REFERENCES

[1] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proc. Int. Symp. Physical Design*, Monterey, CA, 1998, pp. 80–85.
[2] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "Effective free space management for cut-based placement," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 10, pp. 1343–1353, Oct. 2003.
[3] C. J. Alpert, J. H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, Anaheim, CA, 1997, pp. 530–533.
[4] C. J. Alpert and A. B. Kahng, "A general framework for vertex orderings, with application to netlist clustering," in *Proc. ACM/IEEE Design Automation Conf.*, San Diego, CA, 1994, pp. 63–67.
[5] T. N. Bui, "Improving the performance of Kernighan–Lin and simulated annealing graph bisection algorithms," in *Proc. ACM/IEEE Design Automation Conf.*, Las Vegas, NV, 1989, pp. 775–778.
[6] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placements?," in *Proc. ACM/IEEE Design Automation Conf.*, Los Angeles, CA, 2000, pp. 477–482.
[7] T. Chan, J. Cong, T. Kong, and J. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, 2000, pp. 171–176.
[8] T. Chan, J. Cong, T. Kong, J. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, 2003, pp. 299–305.
[9] C.-C. Chang, J. Cong, D. Pan, and X. Yuan, "Multilevel global placement with congestion control," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 4, pp. 395–409, Apr. 2003.
[10] C.-C. Chang, J. Cong, Z. Pan, and X. Yuan, "Physical hierarchy generation with routing congestion control," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Del Mar, CA, 2002, pp. 36–41.
[11] J. Cong, L. Hagen, and A. B. Kahng, "Random walks for circuit clustering," in *Proc. IEEE Int. Conf. ASIC*, Rochester, NY, 1991, pp. 14.2.1–14.2.4.
[12] J. Cong and S. K. Lim, "Edge separability-based circuit clustering with application to multilevel circuit partitioning," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 3, pp. 346–357, Mar. 2004.
[13] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. ACM/IEEE Design Automation Conf.*, San Francisco, CA, 1998, pp. 269–274.
[14] B. Hu and M. M. Sadowska, "Fine granularity clustering-based placement," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 4, pp. 527–536, Apr. 2004.
[15] D. J.-H. Huang and A. B. Kahng, "Partitioning based standard cell global placement with an exact objective," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Napa Valley, CA, 1997, pp. 18–25.
[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. ACM/IEEE Design Automation Conf.*, Anaheim, CA, 1997, pp. 526–529.
[17] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, New Orleans, LA, 1999, pp. 343–348.
[18] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, no. 3, pp. 356–365, Mar. 1991.
[19] G.-J. Nam, C. J. Alpert, P. G. Villarrubia, B. Winter, and M. Yildiz, "The ISPD2005 placement contest and benchmark suite," in *Proc. ACM/IEEE Int. Symp. Physical Design*, San Francisco, CA, 2005, pp. 216–220.
[20] D. M. Schuler and E. G. Ulrich, "Clustering and linear placement," in *Proc. ACM/IEEE Design Automation Conf.*, Dallas, TX, 1972, pp. 50–56.

[21] W.-J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 3, pp. 349–359, Mar. 1995.
[22] N. Viswanathan and C.-N. Chu, "FastPlace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Phoenix, AZ, 2004, pp. 26–33.
[23] J. Vygen, "Algorithms for large-scale flat placement," in *Proc. ACM/IEEE Design Automation Conf.*, Anaheim, CA, 1997, pp. 746–751.
[24] M. Wang, X. Yang, and M. Sarrafzadeh, "DRAGON2000: Standard-cell placement tool for large industry circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2001, pp. 260–263.
[25] M. Yildiz and P. Madden, "Global objectives for standard-cell placement," in *Proc. IEEE Great Lakes Symp. VLSI*, West Lafayette, IN, 2001, pp. 68–72.

Gi-Joon Nam (S'99–M'01) received the B.S. degree in computer engineering from Seoul National University, Seoul, Korea, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Michigan, Ann Arbor.

Since 2001, he has been with the International Business Machines Corporation Austin Research Laboratory, Austin, TX, where he is currently working on the physical design space, particularly placement and timing closure flow. His general interests include computer-aided design algorithms, combinatorial optimizations, very large scale integration system designs, and computer architecture.

Dr. Nam has been serving on the technical program committee for the Association for Computing Machinery (ACM)/IEEE International Symposium on Physical Design, International Conference on Computer Design, and the International System-on-Chip Conference. He is a member of ACM.

Sherief Reda (S'01) received the B.Sc. and M.Sc. degrees in electrical and computer engineering from Ain Shams University, Cairo, Egypt, in 1998 and 2000, respectively, and is currently working towards the Ph.D. degree at University of California at San Diego, La Jolla.

He has over 20 refereed publications in the areas of physical design, very-large-scale-integration (VLSI) test and diagnosis, combinatorial optimization, and computer-aided design (CAD) for deoxyribonucleic-acid chips (DNA) chips.

Mr. Reda received the Best Paper Award at the 2002 Design, Automation and Test in Europe Conference and Exhibition and the First Place Award at the 2005 International Symposium on Physical Design placement contest.

Charles J. Alpert (S'92–M'96–SM'02) received the B.S. and B.A. degrees from Stanford University, Stanford, CA in 1991 and the Ph.D. degree in computer science from the University of California, Los Angeles, in 1996.

He is the Technical Lead of the Design Tools Group at the International Business Machines Corporation Austin Research Laboratory, Austin, TX. His research interests include physical synthesis and design closure.

Dr. Alpert has served as the General Chair and Technical Program Chair for the International Symposium on Physical Design and the Tau Workshop on Timing Issues in the Synthesis and Specification of Digital Systems. He has also served on the technical program committees for the Association for Computing Machinery (ACM)/IEEE Design Automation Conference and International Conference on Computer-Aided Design. He has received three Best Paper Awards from the ACM/IEEE Design Automation Conference and was awarded the Semiconductor Research Corporation's Mahboob Khan Mentor Award in 2001.

Paul G. Villarrubia received the B.S. degree in electrical engineering from Louisiana State University, Baton Rouge, in 1981 and the M.S. degree from the University of Texas, Austin, in 1988.

He is currently a Senior Technical Staff Member at International Business Machines Corporation Austin Research Laboratory, Austin, TX, where he leads the development of placement and timing closure tools and where he has worked in the areas of physical design of microprocessors, physical design tools development, and tools development for application-specific integrated-circuit timing closure. His research interests include placement, synthesis, buffering, signal integrity, and extraction. He is the author or coauthor of more than 18 publications and is the holder of 21 patents.

Mr. Villarubia received one Design Automation Conference Best Paper Award. He is a member of the 2005 International Conference on Computer Aided Design on Transition Pattern Coding and was an Invited Speaker at the 2002 and 2004 International Symposium on Physical Design Conference.

Andrew B. Kahng (A'89–M'03) received the A.B. degree in applied mathematics from Harvard University, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from the University of California at San Diego, La Jolla.

From 1989 to 2000, he was a member of the Computer Science Faculty, University of California, Los Angeles. Since 1997, he has been defining the physical design roadmap for the International Technology Roadmap for Semiconductors (ITRS). Since 2001, he has been the Chair of U.S. and international working groups for the design technology for the ITRS. He has been active in the Microelectronics Advanced Research Corporation Gigascale Silicon Research Center since its inception. He is currently a Professor of computer science and engineering and electrical and computer engineering at the University of California, San Diego. He is the author of more than 200 papers in the very large scale integration (VLSI) computer-aided-design (CAD) literature. His research includes physical design and performance analysis of VLSI, as well as the VLSI design manufacturing interface. Other research interests include combinatorial and graph algorithms and large-scale heuristic global optimization.

Dr. Kahng was the founding General Chair of the Association for Computing Machinery (ACM)/IEEE International Symposium on Physical Design and was the Cofounder of the ACM Workshop on System-Level Interconnect Planning. He received three Best Paper Awards and a National Science Foundation (NSF) Young Investigator Award.