Multilevel Global Placement With Congestion Control

Chin-Chih Chang, Jason Cong, *Fellow, IEEE*, Zhigang Pan, *Member, IEEE*, and Xin Yuan, *Student Member, IEEE*

***Abstract*—In this paper, we develop a multilevel global placement algorithm (MGP) integrated with fast incremental global routing for directly updating and optimizing congestion cost during physical hierarchy generation. Fast global routing is achieved using a fast two-bend routing and incremental A-tree algorithm. The routing congestion is modeled by the wire usage estimated by the fast global router. A hierarchical area density control is developed for placing objects with significant size variations. Experimental results show that, compared to GORDIAN-L, the wire length-driven MGP is 4–6.7 times faster and generates slightly better wire length for test circuits larger than 100 000 cells. Moreover, the congestion-driven MGP improves wiring overflow by 45%–74% with 5% larger bounding box wire length but 3%–7% shorter routing wire length measured by a graph-based A-tree global router.**

***Index Terms*—Congestion, deep submicrometer, interconnect, physical hierarchy, placement, routing.**

I. Introduction

INTERCONNECT has become the dominating factor in determining overall system performance and reliability. Inevitably, it impacts *all* stages of the design flow. In [1], a three-phase interconnect-centric design flow was proposed. It includes: 1) interconnect planning; 2) interconnect synthesis; and 3) interconnect layout in order to emphasize interconnect planning and optimization throughout the entire design process.

The interconnect planning phase is particularly important in the interconnect-centric design flow because it provides early assessments on system performance, thereby enabling performance optimization in the early design stages. In addition to performance optimization, it is equally important to reduce design uncertainty and ensure that the planned results can be realized in the later design stages without significant deviations.

Although the very large scale integration (VLSI) designs are inevitably hierarchical given their high complexity, the hardware description language (HDL) description provided by the architecture and/or circuit designers usually follows the *logical hierarchy* of the design which reflects the logical dependency and relationship of various functions and components in the design. Such a logical hierarchy may not map well to a two-dimensional layout solution, as it is usually conceived with little or no consideration of the layout information [1]. Therefore, in our interconnect planning stage, we first flatten the circuits in the logical hierarchy to the extent that we are certain about the "physical locality" (i.e., we are certain that the circuits in a module should physically stay together). We then generate a *physical hierarchy* to define the global, semiglobal, and local interconnects (based on their levels in the physical hierarchy).

There are some recent studies on generating a good physical hierarchy from the flattened function and logical hierarchy for performance optimization [2], [3]. However, they have little or no consideration for routing congestion, which may cause uncertainty in later design stages because the planned global interconnects in overly congested areas may be forced to make detours or change layers. Table I shows the wire length distribution of a high performance application specified integrated circuit design from IBM after detailed routing. The chip dimension is about 10 mm $\times$ 10 mm. It has more than 1.3 million nets, with buffers inserted at an early phase for performance optimization. It shows that local interconnects will generally be routed at lower metal layers and global interconnects will be preferably routed at higher metal layers. However, due to timing and congestion constraints, some long nets still have to be routed at lower metal layers and some short nets are actually assigned to higher metal layers.

The above data suggest that the performance estimation in the interconnect planning stage must consider layer assignment and congestion control of global interconnects.

In this paper, we shall discuss our multilevel global placement (MGP) algorithm integrated with fast incremental global routing for directly updating and optimizing the congestion cost for physical hierarchy generation. In particular, we shall discuss in detail our contributions on: 1) the use of multilevel framework for scalability; 2) the use of the hierarchical area density control for placing objects with significant size variations; and 3) use of fast incremental routing for efficient routability estimation. The preliminary results of this paper were published in [4] (where we named our algorithm mPG).

II. Previous Work

Once we have determined how much to flatten the logical hierarchy, the physical hierarchy generation problem is quite similar to the traditional global placement problem. There are

Manuscript received June 1, 2002; revised September 9, 2002. This work was supported in part by Semiconductor Research Corporation under Contracts 98-DJ-605 and 2001-TJ-910, in part by the National Science Foundation under Grant CCR-0096383, in part by a Faculty Partnership Award from IBM Corporation, and in part by grants from Intel Corporation and Fujitsu Laboratories of America under the California MICRO program. This paper was recommended by Guest Editor C. J. Alpert.

C.-C. Chang was with the Computer Science Department, University of California, Los Angeles, CA 90095 USA. He is now with Cadence Design Systems, Inc., San Jose, CA 95134 USA (e-mail:chinchih@cadence.com).

J. Cong and X. Yuan are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; yuanxin@cs.ucla.edu).

Z. Pan is with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: dpan@watson.ibm.com).

Digital Object Identifier 10.1109/TCAD.2003.809661

TABLE I
WIRE LENGTH DISTRIBUTION ON ALL ROUTING LAYERS OF AN INDUSTRIAL ASIC DESIGN

	Total Wire Length (meter)							all	net
layers	M1	M2	M3	M4	M5	M6	M7	layers	count
nets <0.1mm	6.88	10.92	7.15	2.40	0.44	0.11	0.04	27.94	73.85%
nets [0.1-1mm]	8.78	20.01	28.76	27.47	18.79	17.50	12.41	133.74	23.23%
nets >1mm	0.69	2.31	6.53	14.31	15.10	21.32	15.03	75.29	2.92%
all nets	6.9%	14.0%	17.9%	18.6%	14.5%	16.5%	11.6%	100%	100%

several well-known global placement techniques: min-cut partition-based placement (e.g., [5]), analytical methods with recursive partitioning (e.g., [6]), and simulated annealing (SA)-based approaches (e.g., [7]). Many placement tools use hybrids of these techniques to yield better results.

The most noticeable SA-based placement is the Timberwolf placer [7] and its successors (e.g., [8], [9]). The SA-based placement uses a temperature to control the amount of hill climbing. The temperature is gradually reduced according to a cooling schedule.

The placement based on analytical methods with recursive partitioning was first proposed in [10]. In [10], starting from the whole design as a single partition, it recursively applies quadratic placement to each partition to give the optimal module location without considering module area constraints. It then bipartitions the modules by selecting a cutline. In [11], the partition method was simplified by choosing the medium module location as the cut line for the initial solution of a partition. In [6], the placement was formulated as a sequence of quadratic programming problems and the recursively refined partitions were translated to the constraints in the quadratic placement steps. Each quadratic programming sees the entire circuit to avoid making local decisions for each partition. It was further extended for linear wire length by a proper scaling using the edge length from previous iterations [12]. In [13], the recursive partition was done by quadripartition. Each quadripartition is optimally solved to minimize the movements and to satisfy the area constraint of each partition. In [14], extra forces are added to the quadratic placement for overlapping removal.

Recently, a number of hybrid placement methods were proposed targeting the efficient handling for high design complexity and performance optimization. Reference [15] used a recursive bipartition-based method to produce routable placements. The "relaxation-based local search" is used for global placement [16], which iteratively selects a set of modules and solves the optimal locations for them. A placement based on recursive min-cut quadripartition is proposed in [17]. It differs from previous approaches by doing bin swapping placement on each partition level using SA technique. A multilevel placement is proposed in [18]. It recursively clusters circuits and solves the coarsest level placement by nonlinear programming using the interior-point method. The placement solution is then recursively declustered and refined by a greedy algorithm to obtain the global placement solution.

None of the aforementioned placement techniques directly handle routing congestion.[1] Several existing works consider the congestion during the placement or floorplanning stage. In [15] and [19], it is shown that there is a mismatch between wire length and congestion objectives. In [20], a simple LZ-shape routing is incorporated into an SA-based floorplanning engine to consider congestion. However, there may not be enough global interconnects seen by a floorplanner. In [21], the wiring demand of a net is modeled by a weighted bounding box (BBOX) length. The wiring demand estimation can be fast, though it may be inaccurate. In [22], precomputed Steiner tree topologies on a few grid structures are used for wiring demand estimations. This approach is tailored for recursive partition-based placement and may not foresee congestion problems within each partition. In [23]–[27], an indirect cell padding or region growing/shrinking is applied to the placement after congestion analysis. This type of approach will not dynamically monitor the congestion changes and has less control over reducing the congestion. Moreover, if the congestion in a region is not caused by the connection to the pins inside the region, cell padding or region shrinking may not always help.[2] In [28], a postprocessing of moving cells with Steiner tree reconstructions is used. In this approach, the cell movement is limited and reconstructing the Steiner trees on each movement is too expensive. In [29], it is shown that a postprocessing technique is effective in minimizing congestion because routing congestion correlates with wire length in a global view. In [30], a postprocessing placement is proposed using a new congestion model. It estimates the congestion using the method in [21] and expands certain congestion regions (by solving an integer programming problem). This approach improves the accuracy of congestion, though the routing congestion is still not dynamically updated.

In general, the most accurate congestion estimation still comes from the global router itself. However, due to the high computational complexity, most previous works used a variety of simplified approximations to estimate the congestion. Our approach differs from the others by building a fast global router and integrating it with an efficient multilevel placement engine to provide dynamic routing congestion guidance to the placement engine.

III. PROBLEM FORMULATION

The interconnects of a VLSI circuit are determined by: 1) the locations and sizes of logic gates, flip-flops, and buffers

[1]Although [14] claimed that wire density can also be handled, it did not report experimental results using wiring density.

[2]For example, if two heavily connected partitions can only be connected with horizontal passthroughs in a narrow channel with blockages on top and bottom sides. The channel will be very congested. However, no matter how you pad the cells inside the channel or grow the regions, the congestion caused by pass-through wires cannot be eliminated.

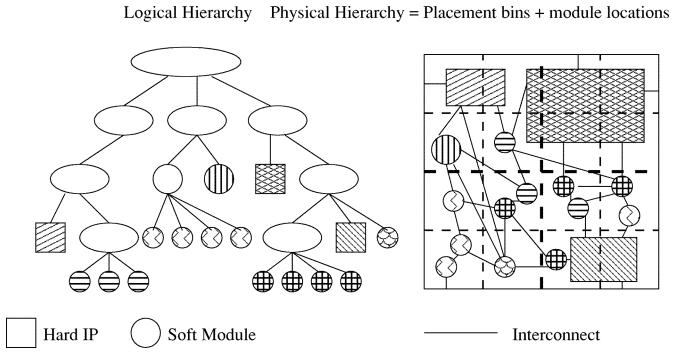


Fig. 1. Illustration of logical and physical hierarchies.

and 2) the interconnect geometry that includes wire locations, layers, and widths. Although interconnect delay is the dominating factor in system performance, only the delays of "long" interconnects are sensitive to wiring geometry. The delays of "short" interconnects are determined mainly by the driver/receiver sizes and are less sensitive to wiring geometry. It is important to identify and optimize global interconnects in early design stages. This important problem of determining global interconnects can be solved by physical hierarchy generation.

The physical hierarchy is represented by a bin structure and cell location assignment. We can use the bin centers to roughly specify cell locations. Global routing can be performed on the bin structure to estimate net topologies. The finer the bin structure becomes, the more accurate the cell locations and net topologies are. We also call our physical hierarchy generation process "coarse/global placement"[3] because we only place cells in coarse locations (bin centers).

The inputs of the physical hierarchy generation consist of the logical hierarchy, design specification, and technology. The logical hierarchy includes a hierarchical net list description consisting of library cells, hard intellectual property (IP) blocks, and soft IP blocks. The width, height, and delay information of library cells and hard IPs are known. The soft IP blocks can be further flattened into other hard IP blocks or library cells. Fig. 1 shows an example of a logical hierarchy and a physical hierarchy generated from that logical hierarchy.

Given the above inputs, the physical hierarchy generation places cells in a bin structure for optimizing the design objectives (delay, area, etc.). The outputs include: 1) block locations specified by bin centers; 2) global nets (inter-bin nets) routing estimations (topology, wire sizing, and layer assignments); and 3) delay estimations, power estimations, etc.

In this paper, we assume that the necessary flattening of the logical hierarchy is complete and focus only on the physical hierarchy generation by global placement for wire length minimization and routing congestion minimization.

IV. mGP Algorithm With Congestion Control

High computational complexity is the major challenge for physical hierarchy generation. Inspired by the recent success of the multilevel methods in efficiently handling high complexity designs in the VLSI computer-aided design (CAD) area [18], [31], [32], the backbone of our system is a multilevel SA engine.

[3]Here, global placement and coarse placement refer to the same thing, therefore, we use these two terms interchangeably in this paper.

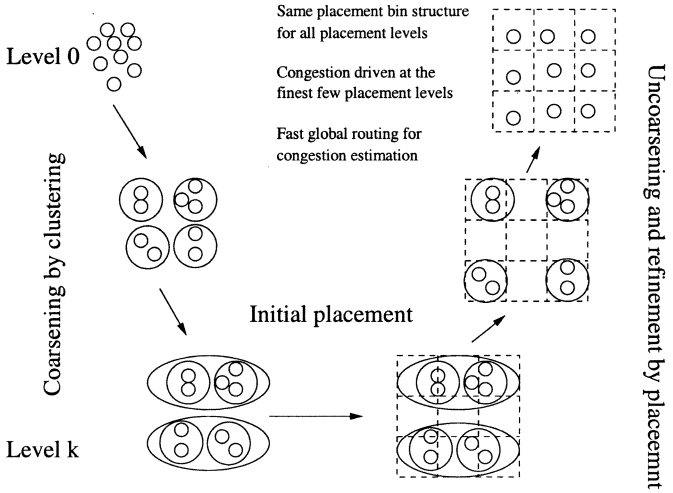


Fig. 2. V-shape multilevel SA coarse placement framework.

The multilevel (or multigrid) method was first developed for solving partial differential equations [33]. The principle is based on the interplay of smoothing and coarse grid correction which complement each other. The multilevel method has been widely used in various applications. A recent survey paper [34] has a good introduction and links to other surveys and resources on the multilevel method.

Applying the multilevel method to the global/coarse placement problem results in a *V-Shape multilevel global/coarse placement* algorithm flow. We first give an overview of this flow.

A. Algorithm Overview: V-Shape mGP

Fig. 2 shows an overview of our multilevel global/coarse placement framework. It includes a coarsening phase which recursively builds coarsening levels, an initial placement step on the coarsest level and a refinement phase which refines each coarser level placement solution to obtain a finer level placement solution.

The details of each phase are explained below.

- *Coarsening by Clustering*: Coarsening can be performed by node clustering. A netlist is coarsened by grouping nodes (or cells) together to form a new netlist with fewer nodes and fewer nets. Sometimes area constraints are considered so that the resulting cluster sizes are more or less balanced. This clustering process is done recursively to generate a netlist that is small enough to be efficiently placed.

  Several clustering algorithms can be used for our purpose. For example, we can apply the connectivity-driven clustering (edge separability-based clustering [ESC] [35], modified hyperedge coarsening [MHEC] [31]), the performance-driven clustering (two-level clustering [TLC] [36]), or even use the original logical hierarchy for clustering. In our current implementation, we use the *FirstChoice* (FC) clustering algorithm [37] because it experimentally gives us a better hierarchy for global placement. (See results in Section V-A2.)

- *Initial Placement on the Coarsest Level*: After the coarsening phase is complete, at the coarsest level a netlist which is much smaller and can be efficiently placed is obtained. Any placement algorithm can be used to generate the initial placement solution. In our system, an SA placement engine is used to generate the initial placement on the coarsest level as in [7] and [9] for its flexibility of integrating various design objectives and handling constraints.
- *Uncoarsening and Placement Refinement*: The uncoarsening phase starts from the coarsest level placement solution and iteratively refines a coarser level solution to a finer level solution. A coarser level placement solution is first declustered to the original netlist in the finer level representation. The subclusters at the finer level assume the locations of their parent clusters in the coarser level. Any iterative placement method can be used to refine the placement solution on the finer level.

  In our system, the refinement is performed by the same SA engine to place clusters in the current level on a placement bin structure. However, the SA process shall not start from a very high temperature as a normal SA-based placement does, because it would be roughly the same as starting from a random solution. The SA engine only starts at a middle or low temperature during the refinement phase. The details of our SA engine are explained in Section IV-D.

There are several features in our multilevel coarse placement that need further explanation. We shall first give an overview of these features and explain the details in the subsequent sections.

- *Placement Cost*: Our coarse placement engine can perform both wire length-driven placement and congestion-driven placement. The congestion-driven placement is usually turned on at the finest few clustering levels. The congestion cost is estimated based on global routing solutions generated by a fast global router. The details of the fast global router are explained in Section IV-C. The placement cost functions are explained in Section IV-D2.
- *Static Placement Bin Structure*: The *same* placement bin structure is used at each level in the multilevel placement to avoid unnecessary cost changes during declustering.[4] (See detailed explanation in Section IV-B.)
- *Hierarchical Area Density Constraints*: To ensure that the coarse placement solution can be legalized to a detailed (overlap-free) placement without significant cell movements, the coarse placement result shall satisfy certain area density constraints. In order to handle significant cluster size variations, we develop a method to handle the area constraints hierarchically. The details are explained in Section IV-B.

B. Hierarchical Area Density Control

In order to legalize a global placement result to a detailed (overlap-free) placement without much wire length increase, each placement bin is usually imposed an area bound constraint, that is, the total area of cells assigned to this bin shall not exceed this bound. It is difficult, however, to maintain a strict area bound in each placement bin during the placement process. The conventional wisdom is to allow some area overflow up to a fixed percentage of the bin area bounds such that a detailed placement solution can be obtained without significant cell movement.

The fixed overflow percentage does not work well in a multilevel coarse placement due to the significant variations in cluster sizes. The clusters in coarser levels may even be larger than a placement bin. One solution is to use coarser bin structures in coarser levels; however, it loses the accuracy and creates unnecessary cost jumps when switching from coarser to finer bin structures. For example, in our early implementations of the multilevel SA process, which are no longer used, we used different placement bin structures for different placement levels in the multilevel placement scheme (coarser level placements with coarser bin structures). For each level, we enforced the area bound constraint by allowing a fixed percentage area balance violation in each bin. The percentage of area balance violation allowed in coarser level placements is greater than or equal to the ones in finer level placements. However, we encountered problems caused by significant cost changes when a coarser level solution was declustered to a finer level solution. We also had problems in move generation with strictly enforced area balance constraints. When the constraints are too tight, it deteriorates both the runtime and the solution quality as the move generation is too restrictive (even with cluster swapping moves).

A cost jump during the optimization means that the optimization engine may not optimize the accurate cost function at higher levels. It usually translates to a less efficient optimization process. In the multilevel placement framework, it is hard to eliminate the cost jump caused by declustering because there are more nets involved in the wire length computation at a finer level than that at a coarser level. However, the cost jump due to switching the grids is unnecessary and can be eliminated by using the same grid structure at each level.

We solve this problem by a hierarchical area density control algorithm. Our density control imposes a density bin hierarchy (DBH) on the target placement bin structure and enforces relaxed area constraints for all the bins in the DBH. Subsequently we shall show that the area constraints are gradually tightened in our multilevel framework while allowing more freedom for cluster moves at coarse levels.

The DBH is formed by recursively grouping adjacent bins to generate the bins in the next level. The bin structure at level 0 is actually the placement bin structure. Fig. 3 shows an example of a DBH where boundary lines of different levels of the bin structure are drawn differently. In this figure, there are three bin structures in the DBH: an 8×8 bin structure at level 0, a 4×4 bin structure at level 1, and a 2×2 bin structure at level 2.

Denote A_b^i as the area bound for a bin B_b^i in level i in the DBH, which is also the summation of all area bounds of bins in level 0 which are contained in B_b^i. Similarly, denote U_b^i as the current area usage for bin B_b^i, which is also the summation of all current area usages of bins in level 0 which are contained in B_b^i. The hierarchical area constraints are enforced on each cluster move. For a cluster move that moves a cluster c of area a_c to bin B_b^0 at level 0 in the DBH, (B_b^0 also is a placement bin), for any bin B_j^i on level i in the DBH that contains bin B_b^0, the

[4]In [18], different placement bin structures are used at each level in the multilevel placement process.

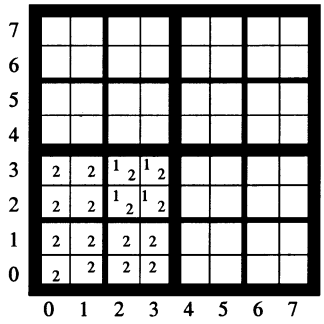


Fig. 3. DBH for area density control.

overflow of bin B_j^i must be smaller than ka_c, where $k \geq 1$ is a user-specified parameter (that is, in DBH, for any B_j^i contains B_b^0, we require $(U_j^i + a_c - A_j^i) \leq ka_c$).

For example, if a cluster with area a_c is moved to bin (2, 3) in Fig. 3, the area constraints of the following bins are enforced: bin (2, 3) on level 0, the level 1 bin covering the region marked with 1 in Fig. 3, and the level 2 bin covering the region marked with 2 in Fig. 3.

Under our hierarchical area density control, a placement bin serves more like a coordinate than a bin. Assigning a cluster to a bin (even if the cluster is larger than the bin) means positioning the cluster center to the bin center. Because of the hierarchical area density control, an overflowed bin always implies empty or less congested bins in its neighborhood. As the refinement is performed from coarser to finer levels, the relaxed area constraints will be gradually tightened due to the size decrease of the clusters. Therefore, there should not be significant area overflow when the refinement on the finest level is complete. Macros will be legalized after a few levels from the coarsest levels. Using this method, our annealing engine can efficiently handle mixes of big and small modules/clusters and will not be stuck due to area constraints.

If the area constraint is satisfied in a region, by applying the pigeon hole principle, at least one of the subregions of the region satisfies the area constraint. We apply this property in the SA move generation. If the SA engine generates a target location bin B_i^0 in the DBH for a cluster c and this move violates the hierarchical area constraints, an alternative location can be efficiently found as follows. In the DBH, we first find the smallest bin B_j^k (in level k) that contains bin B_i^0 and ensure that all of the higher level bins that contain bin B_j^k also satisfy the area constraint. An alternative location can be found by recursively applying the pigeon hole principle from bin B_j^k in the DBH.

C. Global Interconnect Topology Generation and Layer Assignment for Coarse Placement Guidance

Since most of the nets are two-pin nets and a multipin net can be decomposed into two-pin nets, we first build a fast, congestion avoidance two-bend router (LZ-router) for two-pin nets. We use this fast two-pin net routing algorithm with an incremental A-tree generation algorithm for multipin nets to build a fast global router.

The fast global router also includes a fast layer assignment algorithm which assigns nets according to the net criticality. The more critical the nets are, the higher priority they have to choose

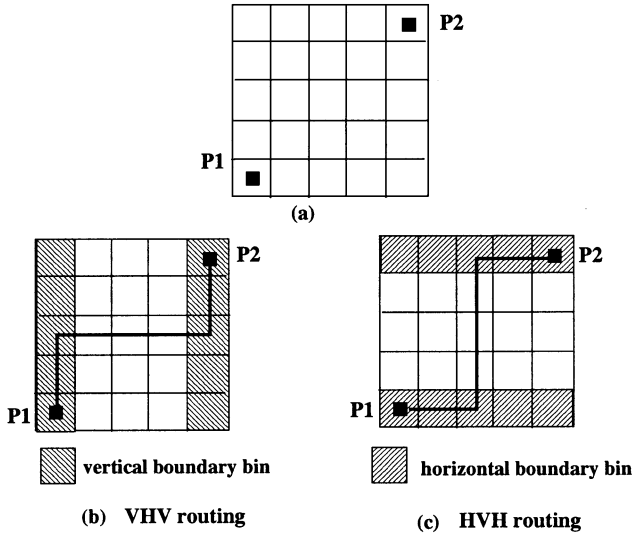


Fig. 4. Illustration of HVH and VHV routing selection of LZ-router.

better routing layers and routing topologies in order to meet the performance constraints. In order to save the runtime, layer assignment is not performed on each SA move in our implementation. Instead, layer assignment is only performed at the beginning of each SA phase on each level.

1) Routing for Two-Pin Nets: We use a fast two-bend routing algorithm to route two-pin nets. We call the two-bend routing "LZ-routing" and our two-bend router an "LZ-router." The possible number of configurations of connecting two pins with coordinates (i, j) and $(i + x, j + y)$ using the LZ-routing is $\mid x \mid + \mid y \mid$. However, the most apparent implementation which needs to calculate $\mid x \mid \times \mid y \mid$ wire usage queries on bin boundaries does not require $O(\mid x \mid + \mid y \mid)$ time, but $O(\mid x \mid \times \mid y \mid)$ time.

Our LZ-router uses auxiliary data structures (similar to a segment-tree data structure, explained in the Appendix) to find good quality routes by performing a binary search of the possible routes for a two-pin net.

The *wire density* of a bin/region is defined as the wire usage of the bin/region divided by its area. For a net connecting two pins P_1 and P_2 which are bounded by a rectangle BBOX B [Fig. 4(a)], if the maximum of the wire density of the vertical (horizontal) boundary bins of B on the vertical (horizontal) layer is $WD_{B_b}^V$ ($WD_{B_b}^H$), the wire density of region B on the horizontal (vertical) layer is $WD_{B_r}^H$ ($WD_{B_r}^V$), then the *possible maximum wire density* of VHV (HVH) routing is the maximum of $WD_{B_b}^V$ ($WD_{B_b}^H$) and $WD_{B_r}^H$ ($WD_{B_r}^V$). The VHV routing pattern [Fig. 4(b)] will be chosen if its possible maximum wire density is smaller than that of HVH, otherwise, the HVH routing pattern [Fig. 4(c)] is chosen.

Assuming the VHV routing is used, our algorithm recursively makes a horizontal cut on B and selects the one with a smaller wire density to route. It stops when the choice narrows to a single row.

If the complexity of a query on wire density in a region is R, the complexity of our LZ-router is $O(\log(\mid x \mid + \mid y \mid)R)$ because it takes at most $O(\log(\mid x \mid + \mid y \mid))$ binary search

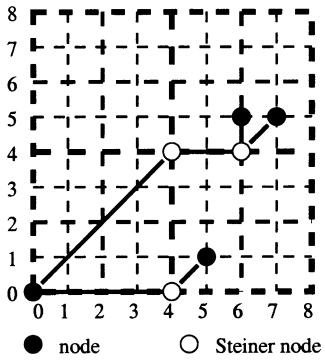


Fig. 5. Illustration of the IncA-tree algorithm.

steps to find a route. Given a $g_x \times g_y$ bin structure, if we do not insist on finding the exact answer of a query used in our LZ-router but use a larger region to answer it approximately (refer to Appendix), it can be answered in $O(\log(g_x+g_y))$ using segment-tree-like data structures. Therefore, the complexity for our LZ-routing is $O(\log(\mid x \mid + \mid y \mid)\log(g_x + g_y))$. Thus, we have

Theorem 1: Given a $g_x \times g_y$ bin structure, the complexity for the LZ-router to route two pins with coordinates (i, j) and $(i + x, j + y)$ is $O(\log(\mid x \mid + \mid y \mid)\log(g_x + g_y))$.

2) Incremental Hierarchical A-Tree Construction: For a multipin net, a rectilinear Steiner arborescence tree (A-tree) is constructed for the routing estimation. An A-tree is a shortest path rectilinear Steiner tree. There are some heuristic algorithms that construct an n-pin A-tree in $O(n \log n)$ time with a solution no worse than 2x of the optimal A-tree solution, e.g., [38] and [39]. However, if an A-tree is reconstructed for any pin location update, the complexity would be $O(n^2 \log n)$ for each n-pin net in a pass of moving all clusters, which is too expensive.

We develop an incremental A-tree (IncA-tree) algorithm that can efficiently update the routing topology for each pin location change. We only explain the construction in the first quadrant because the construction of all quadrants is similar. Given a grid structure consisting of $(2^m + 1) \times (2^m + 1)$ grids on the first quadrant, we can recursively quadripartition the grid structure until reaching the unit grid. For example, the grid structure in Fig. 5 is first quadripartitioned by the cut lines $x = 4$ and $y = 4$ to form four partitions. If there are some pins located inside a partition (including locations on the bottom and left boundaries, but excluding the locations on the right and top boundaries), the lower-left corner of the partition is the root for a subtree connecting all the pins inside this partition. For example, (4, 4) is the root for any pin at location (x, y) with $4 \leq x < 8$ and $4 \leq y < 8$. For a partition with some pins located inside, its root has an edge connecting to the lower-left corner of the previous level quadripartition. In the above example, (4, 4) has an edge connecting to (0, 0). By recursively performing such quadripartition, we can build an A-tree such that each pin at location (x, y) can connect to the origin (0, 0) with $\max(\log x, \log y)$ edges. Any pin insertion (deletion) to location (x, y) only incurs, at most, $\log(x + y)$ edge insertions (deletions). Therefore, each operation of moving a pin from (x_1, y_1) to (x_2, y_2) incurs, at most, $\log(x_1 + y_1) + \log(x_2 + y_2)$ edge changes.

For example, in Fig. 5, assume that at the beginning, only the root pin is inserted. If we insert a pin at (7, 5), it will first connect to its parent, which is (6, 4). It then moves up one level to connect to (4, 4) and then from (4, 4), moves up one level to connect to (0, 0). If we insert a pin on (6, 5), it will connect to (6, 4), sharing the connection from (6, 4) to the root. Similarly, if we insert (5, 1), it goes to (4, 0), which will directly connect to the root.

With the fast two-pin routing and incremental A-tree routing, for an n-pin net with BBOX length L on a $g_x \times g_y$ bin structure, the complexity of updating a nonroot pin move is $O(\log L)$ times the complexity of LZ-routes $O(\log L \log(g_x+g_y))$, which is $O(\log L^2 \log(g_x + g_y))$. For moving the root, the complexity is $O(n \log^2 L \log(g_x+g_y))$. While providing superior guidance for congestion optimization during the coarse placement, the runtime overhead of our congestion cost updating grows slowly due to the low logarithmic complexity.

It is obvious that the IncA-tree may generate routes with longer wire lengths than the A-tree does and using it may overestimate the congestion. However, it is never intended to be used as the final measurement of the placement congestion. Instead, it is used to guide the placement optimization.

D. Multilevel SA Global Placement

The details of the SA engine are described below.

1) Solution Space: A *static* bin structure is used at each level during the multilevel placement. Clusters are placed at bin centers subject to the hierarchical area density constraints, as explained in Section IV-B. The hierarchical area density control makes it possible to place clusters that are much larger than a bin.

2) Cost Function: The cost function for our SA engine has two modes: wire length-driven mode and congestion-driven mode. The cost function for the wire length-driven mode is the simple summation of the BBOX wire lengths of all nets.

The fast global router described in Section IV-C is used to estimate the wire usage in each bin. The cost function for the congestion-driven mode is the quadratic sum of the wire usages of all bins on all the routing layers. This cost is equivalent to the sum of the weighted wire length as it weights all the wire segments in each bin by its wire usage.

On a horizontal routing layer, if a global routing wire segment of width h_s is routed across a bin of width w_b, it contributes $h_s w_b$ wire usage to the bin. If the wire segment starts from or ends at this bin, it contributes $0.5 h_s w_b$ wire usage to this bin. The wire usage on a vertical is similarly estimated.

If a bin has wire usage W, its cost is W^2. The total cost is the summation of all the bins' costs on all the layers. The intuition of this cost function is from a linear weighted cost that was used by many global routers. This cost function encourages the SA moves that can route affected nets with shorter length and less congestion.

For this cost function, if the wire usage of a bin is changed from W to $W + d$, the cost change is $2dW + d^2$. Therefore, if a

wire usage d is added to each bin B_i with wire usage W_i in a set of bins $B_1, B_2, \ldots, B_n$, the cost update is $nd^2 + 2d\sum_{i=1}^{n} W_i$.

This is useful as we can use a tree hierarchy to store the usage data. The above formula shows that if a segment is routed from column i to column j on row k, the wire usage change of each bin is d and the total wire usage is $W_{k,i,j}$ in these bins, then the cost change over the bins is $(j-i+1)d^2+2dW_{k,i,j}$. Therefore, the congestion cost can be efficiently updated by decomposing each global routing change to a set of wire segment insertions and deletions.

Minimizing wire length is strongly related to congestion minimization. Therefore, wire length minimization is performed on coarser levels. Congestion optimization is only turned on at the last few finest levels of refinement. We provide a "reduced mode" where the congestion-driven mode is only turned on at the finest level when the accepting ratio is lower than a predefined threshold t_r and the congestion-driven and wire length-driven modes are alternatively run at a rate of $1-f_c : f_c$. Our experiments find that $t_r = 0.075$ and $f_c = 80\%$ give the best tradeoff between the runtime and the solution quality.

3) Neighborhood Structure: Two moves are used—cluster move and I/O pads swap (not used in the experiments shown in this paper). A cluster move randomly selects a cluster and moves it to another bin. The target location is either randomly selected (within some range limit) or computed to minimize the BBOX wire length. The experimental setting of the random moves probability is $\max(accept\ ratio, 0.6)$. If the generated move violates the hierarchical area density constraints, an alternative target location is generated according to the method described in Section IV-B.

4) Cooling Schedule: Let n_i be the number of clusters in level i. The cooling schedule is shown below.

- *Move accepting probability*: The probability p for accepting a move with cost-change ΔC is $p = e^{-\Delta C/T}$ when $\Delta C > 0$ and $p = 1$ when $\Delta C \leq 0$.
- *Starting temperature*: The starting temperature for the coarsest level (level k) is set to be 20 times the standard deviation of the costs of n_k random moves, as suggested by [40]. For the remaining levels, the SA engine works more like local refinement at lower temperatures. By starting from a lower temperature, the SA engine permits less cost-increasing random walks from the good solutions obtained from the coarser levels, thus speeding up the optimization process. The starting temperature is calculated by using methods similar to [41] which assumes that the temperature of the solution from the previous level is in an equilibrium state such that the expected cost-change of random moves is zero.[5] For level i ($i < k$), this method first generates n_i random moves. A binary search is then performed to find a temperature that makes the expected cost-change close to zero. Given a temperature T, the probability P_T of accepting each of the n_i moves is first calculated. The expected cost-change for T is then calculated based on P_T. A binary search is performed on the temperatures to find the temperature T^* that makes the expected cost-change close to zero. The temperature T^* is our starting temperature.[6]
- *Next temperature calculation*: The next temperature calculation is a function of accepting ratio α. For a given temperature T, the next temperature is $0.5T$ if $\alpha > 0.96$; $0.9T$ if $0.8 < \alpha \leq 0.96$; $0.95T$ if $0.15 < \alpha \leq 0.8$; $0.8T$ if $\alpha \leq 0.15$[42].
- *Inner number*: Two inner numbers $inner_0$ and $inner_1$ are used. For each temperature on level i, the SA process starts with a pass of $inner_0 \times n_i$ moves. If the current pass reduces the total cost, the temperature is repeated with $inner_1 \times n_i$ moves until m cost increase passes are encountered. If the probing pass increases the total cost, the SA engine jumps to the next temperature. By starting with a smaller inner number for the probing pass, the SA engine can quickly skip some temperatures without significantly increasing the total cost if the calculated starting temperature is too high. The reason for running the same temperature with several passes is that our inner number is relatively small compared to the conventional annealing schedule. Enough runs are needed in order to let the SA engine do enough work at each temperature. A pass with increased cost usually indicates that the cost reduction in the current temperature is becoming less effective and a lower temperature should be used. The SA engine should tolerate a few cost increasing passes such that it does not jump to a lower temperature too early. The values are experimentally set to $inner_0 = 1$, $inner_1 = 5$ and $m = 2$.
- *Freezing temperature*: The freezing temperature is set to be $\lambda C/ec$, where C is the current cost; λ is a user-input parameter; ec is the net count of the current level [42]. The default value for λ is 0.005.

V. Experimental Results

Our multilevel global placement algorithm is implemented in C++/STL. It can be run in three modes: wire length minimization (MGP), congestion-driven at the finest level (MGP-cg), and the "reduced congestion-driven mode" (MGP-cg.rd) described in Section IV-D2. Our experiments are conducted on a Sun Blade 1000 workstation running at 750-MHz frequency (except the experiments in Section V-D).

Three sets of placement benchmarks are used for experiments and comparisons in this section. The first set of benchmarks is provided by the authors of [18] and is in the *PROUD* format which can be taken by GORDIAN-L [12] and DOMINO [43]. It contains two of the largest circuits (*avqsmall*, *avqlarge*) in 1993 Microelectronics Center of North Carolina layout benchmark sets and some *ibmXX* circuits derived from the International Symposium on Physical Design (ISPD98) IBM benchmark suite [44]. We call this set of benchmarks as BENCHP. The second set of benchmarks is obtained from [45] in the *GSRC BookShelf* format and is referred as BENCHB. It is also derived from the ISPD98 IBM benchmark suite [44]. The third set of

[5]In extreme cases, where all moves make cost increase or decrease (we never encountered such cases), we can simply set the temperature to the freezing temperature or 20 times the standard deviation of the costs of n_i random moves.

[6]Because the solution from the previous coarser level is not really in an equilibrium state, we have added in other user-input engineering adjustments to fine tune the starting temperature calculation.

benchmarks is a set of industrial benchmarks from IBM and is referred as BENCHI.

Though the *ibmXX* placement benchmarks in BENCHP and BENCHB are all derived from the ISPD98 IBM benchmark suite [44], they are different in both cell library and netlist. As the original ISPD98 IBM benchmark suite [44] was released as partitioning benchmarks, only netlist and cell area were provided. No placement specifications, such as standard-cell row specifications, were provided and most of the benchmarks contain large macros. The *ibmXX* circuits in BENCHP have the same netlist as [44], however, they assume that all the cells have a uniform dimension in order to conform to a standard-cell placement benchmark. In BENCHB, all the big macros are first taken out of the circuits; second, the remaining cells are converted to standard-cells of the same height using the area specified in [44]; third, nets with a degree of less than two are removed due to the removal of macros, thus most of the derived circuits do not have connections to I/O pads.[7] Therefore, the netlists in BENCHB are different from that of [44]. But they share the cell library except that there are no macros in BENCHB. In order to avoid confusion, we rename the circuits *ibmXX* in BENCHP by adding "$-p$" suffixes to indicate the difference. For both sets of benchmarks, placement row specifications are added. However, the *PROUD* format only specifies the number of standard-cell rows, while the *GSRC BookShelf* format specifies not only the number of rows but also the row locations and row spacing. This makes these two formats incompatible in terms of row specification. Even if we convert BENCHB into *PROUD* format and run GORDIAN-L and DOMINO, DOMINO cannot generate rows specified in BENCHB as the program itself automatically sets the row locations and spacing. Therefore, we did not run GORDIAN-L and DOMINO on the circuits in BENCHB and only compared our placer with GORDIAN-L on BENCHP for wire length minimization. We did not use BENCHP for congestion control experiments because all of the cells in *ibmXX* have a uniform size, which is not reasonable for routing.[8] We used BENCHB for congestion control experiments instead.

The characteristics of these two sets of benchmarks, BENCHP and BENCHB, are listed in Table II. Dangling cells are pruned out in BENCHB. The characteristics of BENCHI is included in Table XII.

TABLE II
CHARACTERISTICS OF CIRCUITS IN BENCHP AND BENCHB

BENCHP				BENCHB			
circuit	#cells	#nets	#rows	circuit	#cells	#nets	#rows
avqsmall	21854	22124	80	ibm01	12028	11507	64
avqlarge	25114	25384	86	ibm02	19062	18429	64
ibm04-p	27220	31970	116	ibm03	21879	21621	64
ibm07-p	45639	48117	151	ibm04	26332	26163	64
ibm09-p	53110	60902	162	ibm05	28146	28446	64
ibm10-p	68685	75196	185	ibm06	32018	33354	64
ibm14-p	147088	152772	271	ibm07	44811	44394	64
ibm15-p	161187	186608	283	ibm08	50672	47944	64
ibm16-p	182980	190048	302	ibm09	51382	50393	64
ibm17-p	184752	189581	303	ibm10	66762	64227	64
ibm18-p	210341	201917	324	ibm11	68046	67016	128
				ibm12	68735	67739	128
				ibm13	80906	83806	128
				ibm14	145397	143202	128
				ibm15	157806	161196	128
				ibm16	181581	181188	128
				ibm17	182178	180684	128
				ibm18	210048	200565	128

TABLE III
PLACEMENT GRID IMPACT ON FINAL WIRE LENGTH

circuit	grid	$\frac{\#cells}{bin}$	DP. WL	tot. CPU	$\frac{dpCPU}{totCPU}$
avqsmall	32X32	21	1.22e+07(1.000)	635(1.000)	0.41
	64X64	5	1.16e+07(0.946)	724(1.140)	0.33
avqlarge	32X32	24	1.34e+07(1.000)	611(1.000)	0.33
	64X64	6	1.26e+07(0.942)	766(1.253)	0.25
ibm04-p	32X32	26	6.85e+06(1.000)	1254(1.000)	0.05
	64X64	6	6.5e+06(0.949)	1868(1.490)	0.03
	32X32	44	1.21e+07(1.000)	2048(1.000)	0.13
ibm07-p	64X64	11	1.02e+07(0.846)	2724(1.330)	0.09
	128X128	2	1.06e+07(0.874)	3791(1.851)	0.07
	32X32	51	1.6e+07(1.000)	2186(1.000)	0.17
ibm09-p	64X64	12	1.16e+07(0.728)	3273(1.497)	0.10
	128X128	3	1.16e+07(0.727)	4114(1.882)	0.08
	32X32	67	3.22e+07(1.000)	3429(1.000)	0.15
ibm10-p	64X64	16	1.91e+07(0.594)	4471(1.304)	0.11
	128X128	4	1.92e+07(0.597)	5747(1.676)	0.08
	64X64	35	4.3e+07(1.000)	7673(1.000)	0.09
ibm14-p	128X128	8	3.99e+07(0.928)	9112(1.188)	0.07
	256X256	2	4.01e+07(0.933)	11172(1.456)	0.06
	64X64	39	5.47e+07(1.000)	10080(1.000)	0.10
ibm15-p	128X128	9	5.06e+07(0.925)	13318(1.321)	0.07
	256X256	2	5.11e+07(0.935)	16063(1.593)	0.06
	64X64	44	5.9e+07(1.000)	11623(1.000)	0.11
ibm16-p	128X128	11	5.22e+07(0.884)	14511(1.248)	0.08
	256X256	2	5.3e+07(0.898)	19291(1.660)	0.06
	64X64	45	7.37e+07(1.000)	15908(1.000)	0.09
ibm17-p	128X128	11	6.85e+07(0.930)	18444(1.159)	0.08
	256X256	2	6.89e+07(0.936)	21986(1.382)	0.07
	64X64	51	6.19e+07(1.000)	12574(1.000)	0.13
ibm18-p	128X128	12	5.12e+07(0.828)	16266(1.294)	0.10
	256X256	3	5.13e+07(0.829)	19725(1.569)	0.07

A. Wire Length Minimization Experiments

In this section, we study how the placement grids and different coarsening schemes can affect the placement result for wire length minimization. We compare MGP with GORDIAN-L [12], a well-known quadratic placer and Dragon [17], an advanced SA-based standard-cell placer, on wire length minimization. We also show the effectiveness of the multilevel approach for handling the large-scale placement problem by comparing MGP with its flat version.

1) Placement Grids Impact on Final Wire Length: We studied the impact of placement grid on the final wire length using circuits in BENCHP. We ran MGP followed by DOMINO under different placement grids. Based on the results of multiple runs of MGP/DOMINO, we report the average detailed placement BBOX wire length (DP. WL), the average total runtime in seconds (tot. CPU) and the average percentage of detailed placement runtime in total runtime ($\frac{\text{dpCPU}}{\text{totCPU}}$) in Table III. Data in parentheses are normalized.

From the table, it can be seen that the placement grid should be fine enough so that the wire length estimated in the coarse placement stage is very close to the wire length estimated in the detailed placement stage. If it is set to be too fine, however, it will cause runtime overhead with little wire length improvement. Another observation is that fine grid will lead to runtime

[7]This is because all the I/O pads connect only with the macros which have been taken out in these circuits.

[8]Cells with more pins tend to cause high routing congestion if all of the cells have the same size.

TABLE IV
COARSENING SCHEME IMPACT ON FINAL WIRE LENGTH

circuit	FC			ESC			ESC-fast			MHEC			MHEC-fast		
	#lev	WL	CPU	#lev	WL	CPU	#lev	WL	CPU	#lev	WL	CPU	#lev	WL	CPU
avqsmall	5	1	1	10	0.99	1.46	7	0.97	1.29	13	1.03	1.69	8	1.05	1.34
avqlarge	5	1	1	9	1.03	1.55	7	1.05	1.28	13	1.07	1.71	8	1.08	1.43
ibm04-p	4	1	1	13	0.97	2.11	8	0.97	1.48	14	0.95	2.34	8	0.96	1.61
ibm07-p	5	1	1	10	1.08	1.99	7	1.08	1.45	14	1.18	1.82	9	1.22	1.44
ibm09-p	5	1	1	14	1.09	2.22	8	1.11	1.50	15	1.11	1.98	8	1.12	1.54
ibm10-p	6	1	1	11	0.99	1.61	7	1.11	1.11	13	1.07	1.16	8	1.08	1.04
ibm14-p	6	1	1	11	0.96	1.75	8	0.97	1.29	14	1.03	1.64	9	1.03	1.44
ibm15-p	6	1	1	12	0.97	1.83	8	0.97	1.39	13	1.05	1.85	8	1.03	1.45
ibm16-p	7	1	1	11	1.14	2.01	7	1.15	1.34	14	1.16	1.76	9	1.14	1.62
ibm17-p	7	1	1	10	1.19	2.05	7	1.19	1.48	13	1.20	1.70	8	1.19	1.47
ibm18-p	7	1	1	8	1.03	1.99	6	1.04	1.54	13	1.07	1.47	9	1.08	1.38
avg.	-	1	1	-	1.04	1.87	-	1.05	1.38	-	1.08	1.74	-	1.09	1.43

reduction in the detailed placement, though the total runtime will increase. Through our experiments, we found that for small- to medium-sized circuits, the number of y-partitions of the grid structure should be equivalent to the number of rows to ensure an accurate estimation in the global placement phase. For large circuits, the grid can be set to a limit of no more than 20 cells in each bin.

2) Coarsening Schemes Impact on Final Wire Length: We compared different coarsening schemes to study their impact on the final wire length using circuits in BENCHP. Three clustering algorithms are compared: FC clustering algorithm [37], ESC algorithm [35], and MHEC algorithm [31]. In the FC algorithm, cells/clusters that have the strongest connection are clustered together. ESC exploits more global connectivity information, *edge separability*, to guide the clustering process. *Edge separability* is defined as the minimum cutsize among the cuts separating two cells/clusters in the netlist. MHEC performs net coarsening and tries to find as many groups of cells/clusters (that are netwise independent) as possible. ESC and MHEC algorithms require cluster size constraint so that the generated clusters will not be larger than the given area constraint. We recursively call the clustering algorithm to generate a multilevel hierarchy until the number of clusters on the coarsest level is no less than a user-specified number, such as 300. Due to the clustering size constraints, ESC and MHEC tend to generate more levels than the FC algorithm. Therefore, we skip refinement on some levels generated by ESC and MHEC and call the derived clustering schemes ESC-fast and MHEC-fast, respectively. We ran MGP followed by DOMINO using each coarsening scheme. For each scheme, we report the number of levels of refinement (#lev), the final wire length (WL), and total runtime (CPU) in Table IV. Final wire length and total runtime of ESC and MHEC are normalized with respect to those of FC to one.

From this table, it can be seen that FC provides the best wire length with the least runtime. In addition, it is shown that: 1) it is not always good to have more levels of refinements and 2) the clustering hierarchy matters somewhat. Although two hierarchies may have the same or a similar number of levels for refinement, the better hierarchy will bring better wire length with less runtime (for example FC vs. ESC-fast on *ibm16-p* and *ibm17-p*). This would be due to the following: 1) the independence requirement that MHEC pursues may destroy some clusters of cells that naturally exist in the netlist [46], which brings a poor hierarchy; 2) although ESC exploits global view on edge separability, the cutsize may not correlate very well with optimal wire length; and 3) FC tends to remove a large number of the intercluster nets in successive coarse netlists and thus, makes it easy to find high-quality initial placement results that require little refinement during the uncoarsening phase.

TABLE V
WIRE LENGTH MINIMIZATION COMPARISON WITH GORDIAN-L ON BENCHP

circuit	grid size	Gor+Dom		MGP+Dom	
		WL (10^6)	CPU (s)	WL (10^6)	CPU (s)
avqsmall	64X80	11.3	857	11.5 (1.02)	694 (0.81)
avqlarge	64X86	12.6	925	12.0 (0.95)	764 (0.83)
ibm04-p	64X64	6.86	1577	6.50 (0.95)	1397 (0.89)
ibm07-p	64X64	10.9	4385	10.2 (0.94)	2724 (0.62)
ibm09-p	64X64	11.8	6767	11.6 (0.98)	3273 (0.48)
ibm10-p	64X64	18.8	14133	19.1 (1.02)	4471 (0.32)
ibm14-p	128X128	40.8	39657	39.9 (0.98)	9112 (0.23)
ibm15-p	128X128	52.1	63876	50.6 (0.97)	13318 (0.21)
ibm16-p	128X128	55.0	81868	52.2 (0.95)	14511 (0.18)
ibm17-p	128X128	67.9	98440	68.5 (1.01)	18444 (0.19)
ibm18-p	128X128	53.7	129065	51.2 (0.95)	16266 (0.13)

3) Wire-Length Minimization Comparison With GORDIAN-L on BENCHP: We compared our wire length-driven MGP with GORDIAN-L [12] on circuits in BENCHP in Table V. We report the BBOX wire length of the detailed placement results generated by running GORDIAN-L followed by DOMINO [43] in columns titled "Gor+Dom." Also we report the BBOX wire length of the detailed placement results generated by running MGP followed by DOMINO in columns titled "MGP + Dom." We list the ratio between the wire length and runtime of MGP and those of GORDIAN-L in parentheses.

Table V shows that MGP provides a slightly shorter wire length and significantly less runtime, especially for circuits larger than 100K, where MGP is 4 to 6.7 times faster.

4) Wire Length Minimization Comparison With DRAGON on BENCHB: We compared the wire length-driven MGP with DRAGON (version 2.20) [17] on circuits in BENCHB. In order to generate a detailed placement, we implemented a simple detailed placer which can generate the specified standard-cell rows based on the coarse placement results produced by MGP and perform greedy cell swapping to reduce the wire length.[9]

TABLE VI
WIRE LENGTH MINIMIZATION COMPARISON WITH DRAGON ON BENCHB

circuit	grid	avg-DP-WL		avg-tot-CPU		avg-GP-WL		avg-GP-CPU	
		DRAGON	MGP	DRAGON	MGP	DRAGON	MGP	DRAGON	MGP
ibm01	64x64	4.5e+06	4.8e+06(1.05)	800	483(0.60)	4.5e+06	4.4e+06(0.98)	788	459(0.58)
ibm02	64x64	1.3e+07	1.4e+07(1.03)	1598	1216(0.76)	1.3e+07	1.3e+07(1.00)	1560	1101(0.71)
ibm03	64x64	1.3e+07	1.3e+07(1.04)	1193	869(0.73)	1.3e+07	1.3e+07(1.02)	1155	786(0.68)
ibm04	64x64	1.6e+07	1.7e+07(1.06)	1408	1050(0.75)	1.6e+07	1.6e+07(1.03)	1358	932(0.69)
ibm05	128x64	3.7e+07	3.8e+07(1.03)	3853	1756(0.46)	3.7e+07	3.7e+07(1.02)	3758	1660(0.44)
ibm06	128x64	1.9e+07	2.1e+07(1.06)	2525	1526(0.60)	1.9e+07	2e+07(1.03)	2440	1446(0.59)
ibm07	128x64	3e+07	3.2e+07(1.06)	3234	2363(0.73)	3e+07	3.1e+07(1.04)	3098	2137(0.69)
ibm08	128x64	3.3e+07	3.4e+07(1.05)	4800	3145(0.66)	3.3e+07	3.3e+07(1.02)	4582	2741(0.60)
ibm09	128x64	2.8e+07	2.9e+07(1.03)	3660	2526(0.69)	2.8e+07	2.8e+07(0.99)	3462	2247(0.65)
ibm10	256x64	5.7e+07	6e+07(1.06)	7520	4335(0.58)	5.7e+07	5.8e+07(1.02)	7168	4029(0.56)
ibm11	128x128	4.1e+07	4.3e+07(1.04)	5035	3570(0.71)	4.1e+07	4.2e+07(1.01)	4881	3343(0.69)
ibm12	128x128	7.1e+07	7.7e+07(1.09)	7574	4878(0.64)	7.1e+07	7.5e+07(1.07)	7342	4514(0.61)
ibm13	128x128	5.1e+07	5.4e+07(1.07)	6578	4658(0.71)	5.1e+07	5.3e+07(1.03)	6319	4310(0.68)
ibm14	256x128	1.2e+08	1.3e+08(1.08)	20589	11689(0.57)	1.2e+08	1.2e+08(1.05)	19444	11068(0.57)
ibm15	256x128	1.3e+08	1.5e+08(1.11)	25800	14456(0.56)	1.3e+08	1.4e+08(1.08)	24365	13489(0.55)
ibm16	256x128	1.8e+08	1.8e+08(1.03)	31779	17436(0.55)	1.8e+08	1.8e+08(0.99)	29677	15992(0.54)
ibm17	256x128	2.7e+08	2.7e+08(1.01)	37752	20399(0.54)	2.7e+08	2.7e+08(0.99)	35505	18657(0.53)
ibm18	256x128	1.9e+08	2e+08(1.06)	35327	19829(0.56)	1.9e+08	2e+08(1.04)	32621	18045(0.55)
avg.		1	1.05	1	0.63	1	1.02	1	0.61

The placement grid for MGP is set to be the same as the final partition grid DRAGON used. Based on the results of multiple runs of DRAGON and MGP, we report the average detailed placement BBOX wire length (avg-DP-WL), average total runtime (avg-tot-CPU), average global wire length (avg-GP-WL), and average global placement runtime (avg-GP-CPU) in Table VI. The ratios between the wire length and runtime of MGP and those of DRAGON are listed in parentheses.

It can be seen that MGP produces results with slightly longer wire length (5%) and a 2 times reduction in runtime for large designs when compared with DRAGON.

5) Impacts of the Multilevel Approach: This experiment is designed to show the impact of the multilevel approach on the placement problem. We ran our SA-based placement engine directly on the original netlist without doing any clustering on some circuits in BENCHP. We call this version of MGP flat-MGP. The results are shown in Table VII. The wire length ratio of MGP to flat-MGP is shown in parentheses in column 4. The runtime ratio is shown in parentheses in column 5.

The results show that the multilevel SA-based placement approach generates results with 11%–29% shorter wire length than the approach of running the same SA engine on the flat netlists. We can also see a 10%–52% runtime reduction in most of the test cases. It demonstrates that the multilevel approach is very effective in handling the large-scale placement problem in terms of both runtime and quality.

B. Experimental Results of Incremental A-Tree Algorithm

In this section, we study the performance of the incremental A-tree algorithm for both runtime efficiency and routing wire length quality.

[9]This simple detailed placer is not as good as DOMINO for wire length minimization, therefore, it is not used in the wire length minimization experiments for circuits in BENCHP. As DOMINO can not generate rows specified by the *GSRC BookShelf* format (as mentioned at the beginning of this section), we can not use it as a detailed placer for experiments on BENCHB.

TABLE VII
MULTILEVEL VERSUS FLAT PLACEMENT WIRE LENGTH COMPARISON

circuit	flat-MGP		MGP	
	WL (10^6)	CPU (s)	WL (10^6)	CPU (s)
ibm04-p	7.28	2052	6.50(0.89)	1397(0.90)
ibm07-p	11.7	3826	10.2(0.88)	2724(0.71)
ibm09-p	14.3	5242	11.6(0.81)	3273(0.62)
ibm10-p	21.8	7093	19.1(0.88)	4471(0.63)
ibm14-p	49.9	18881	39.9(0.80)	9112(0.48)
ibm15-p	66.7	21252	50.6(0.76)	13318(0.63)
ibm16-p	67.9	23362	52.2(0.75)	14511(0.62)
ibm17-p	80.7	22378	68.5(0.85)	18444(0.82)
ibm18-p	72.1	25720	51.2(0.71)	16266(0.78)

1) Speedup by Incremental A-Tree: The incremental A-tree (IncA-tree) algorithm enables us to directly integrate a global router into the placement engine without incurring an overly long runtime. In this section, we compare the runtime of using IncA-tree algorithm with that of using an implementation that completely routes a net by an A-tree algorithm [38] whenever a pin of this net is moved during the SA process.

We used some circuits from BENCHB for this experiment. For each circuit, we first eliminated all the two-pin nets and only kept the multiterminal nets for testing.[10] For a move generated by the SA engine, we used the IncA-tree algorithm to incrementally evaluate the congestion and recorded the runtime for a predetermined number of moves. An identical set of moves was also evaluated by the A-tree algorithm. Net characteristics and comparison results are shown in Table VIII.

It can be seen in Table VIII that the IncA-tree algorithm can speed up the evaluation process by a factor of five and even more, when the nets with a higher degree become dominant.

2) Routing Wire Length Comparison Between IncA-Tree and A-Tree: As shown in Section IV-C2, the incremental A-tree algorithm tends to generate routes with longer length due to its

[10]We use suffixes "$-r$" in the circuit names to indicate that the two-pin nets are removed from [45].

TABLE VIII
CONGESTION EVALUATION TIME COMPARISON BETWEEN INCA-TREE AND A-TREE. TWO-PIN NETS ARE REMOVED

circuit	nets characteristics					#moves	eva. time (s)		speed
	#nets	3pin	4pin	5pin	6^+pin		IncA-tree	A-tree	up
ibm01-r	5681	36%	18%	14%	32%	12028	15.35	80.24	5.2X
ibm02-r	8506	20%	22%	23%	35%	19062	26.36	170.74	6.47X
ibm03-r	8137	38%	16%	11%	35%	21879	24.18	174.79	7.20X
ibm04-r	10580	37%	16%	13%	34%	26332	37.83	462.69	12.23X
ibm05-r	10433	11%	0%	23%	66%	28146	60.84	586.87	9.65X
ibm06-r	13968	28%	23%	14%	35%	32018	55.48	623.26	11.23X

TABLE IX
ROUTING WIRE LENGTH COMPARISON BETWEEN INCA-TREE ALGORITHM AND A-TREE ALGORITHM

circuit	ibm01	ibm02	ibm03	ibm04	ibm05	ibm06	ibm07	ibm08	ibm09	ibm10	avg.	max.	min
$\frac{WL_{IncAtree}}{WL_{Atree}}$	1.46	1.50	1.45	1.41	1.51	1.55	1.44	1.55	1.45	1.47	1.48	1.55	1.41

TABLE X
WIRE WIDTH AND SPACING SETTING FOR CONGESTION CONTROL EXPERIMENTS ON BENCHB

circuit	ibm01	ibm02	ibm03	ibm04	ibm05	ibm06	ibm07	ibm08	ibm09
#layers	4	4	2	2	2	4	2	2	2
(w_1, s_1)	(4, 4)	(2.8, 2.8)	(2, 2)	(2, 2)	(1, 1)	(3, 3)	(2, 2)	(2, 2)	(2, 2)
(w_2, s_2)	(4, 4)	(2.8, 2.8)	(2, 2)	(2, 2)	(1, 1)	(3, 3)	(2, 2)	(2, 2)	(2, 2)
(w_3, s_3)	(8, 8)	(3.8, 3.8)				(6, 6)			
(w_4, s_4)	(8, 8)	(3.8, 3.8)				(6, 6)			
circuit	ibm10	ibm11	ibm12	ibm13	ibm14	ibm15	ibm16	ibm17	ibm18
#layers	2	2	2	2	2	2	2	2	2
(w_1, s_1)	(1.7, 1.7)	(1.8, 1.8)	(1.8, 1.8)	(2.1, 2.1)	(1.2, 1.2)	(1, 1)	(1, 1)	(0.9, 0.9)	(1, 1)
(w_2, s_2)	(1.7, 1.7)	(1.8, 1.8)	(1.8, 1.8)	(2.1, 2.1)	(1.2, 1.2)	(1, 1)	(1, 1)	(0.9, 0.9)	(1, 1)

feature that supports incremental tree construction/update. We compared the routing wire length generated by the incremental A-tree algorithm and that generated by the A-tree algorithm [38] using some of the circuits in BENCHB. For a placed circuit, we used the incremental A-tree algorithm and A-tree algorithm to route the nets and compared the total routing wire length produced by these two algorithms in Table IX. Two-pin nets are included in the netlist.

From this table, it can be seen that in terms of total routing wire length, the incremental A-tree algorithm produces routes with about 50% more length than the A-tree algorithm. However, as explained in Section IV-C2, the incremental A-tree construction is never intended to be used as the final measurement of placement congestion. Rather, it is used to guide the placement optimization.

C. Congestion Control Experiments on BENCHB

In order to evaluate the effects of the congestion optimization, we implemented a global router based on the graph-based A-tree (GA-tree) algorithm [39] to evaluate the congestion of a placement solution. When constructing an A-tree topology for a net, the GA-tree algorithm can consider both the congestion and the obstacle information. This algorithm works on a routing graph where nodes represent routing bins and edges correspond to the shared boundaries of adjacent bins.

A global routing solution with congestion control is obtained using a slope-based cost function for the edge weight to penalize the overflowed/highly congested edges (similar to that used in [47]) and by updating the weight after routing each net.

We use BENCHB for the comparison on congestion control between MGP and MGP-cg and MGP-cg.rd.[11] For each circuit, we ran MGP to perform wire length-driven placement and ran MGP-cg and MGP-cg.rd to perform a congestion-driven placement. The GA-tree-based global router is used to evaluate the congestion of the placement results. The wire width and spacing for each layer used in each test case is listed in Table X.[12]

In Table XI, we report the congestion pictures in total overflow (tot. ov), the maximal boundary congestion (max. b.cg), the minimal boundary congestion (min. b.cg), the routing wire length (routing WL) and total BBOX wire length for MGP, MGP-cg and MGP-cg.rd.

It can be seen that although in terms of the BBOX wire length, wire length-driven MGP offers better results. In general, the routing wire length actually becomes larger than that generated

[11]Because there is no timing information in this set of benchmarks, layer assignment based on timing criticality was not performed; instead, a simple layer assignment based on the net length order is performed. However, our algorithm can support timing criticality-driven layer assignment.

[12]As there is no unit for length in the *GSRC BookShelf* format, the width and spacing have no unit. For each circuit, we chose the wire width and spacing to create a congested (or near congested) routing case.

TABLE XI
CONGESTION CONTROL COMPARISON BETWEEN MGP, MGP-CG.RD, AND MGP-CG ON BENCHB

circuit name	grid size	BBOX WL			routing WL			max. b.cg			min. b.cg			tot. ov			CPU (s)		
		MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg
ibm01	32x64	4.71e6	5.04e6	5.23e6	5.76e6	5.66e6	5.67e6	1.47	1.24	1.13	0.00	0.00	0.11	1728	905	420	1039	7244	17313
ibm02	32x64	1.32e7	1.4e7	1.41e7	1.76e7	1.7e7	1.6e7	1.38	1.25	1.14	0.00	0.08	0.08	6844	3104	1247	1898	12113	37658
ibm03	32x64	1.31e7	1.37e7	1.31e7	1.73e7	1.66e7	1.44e7	1.38	1.34	1.09	0.02	0.02	0.26	7031	3494	93	905	5400	27221
ibm04	64x64	1.66e7	1.74e7	1.73e7	1.97e7	1.96e7	1.87e7	1.20	1.22	1.06	0.06	0.14	0.14	1256	639	78	1228	9709	27922
ibm05	64x64	3.71e7	3.88e7	3.93e7	4.7e7	4.56e7	4.29e7	0.99	0.93	0.98	0.02	0.02	0.03	0	0	0	1859	13444	52718
ibm06	64x64	1.93e7	2.05e7	2.09e7	2.77e7	2.68e7	2.6e7	1.65	1.61	1.51	0.00	0.09	0.26	54354	38715	29998	2854	22435	89020
ibm07	64x64	3.18e7	3.31e7	3.33e7	4.57e7	4.48e7	4.33e7	1.78	1.78	1.69	0.03	0.07	0.21	75144	63947	47443	2410	14683	43618
ibm08	64x64	3.31e7	3.49e7	3.56e7	4.89e7	4.61e7	4.46e7	1.50	1.41	1.44	0.02	0.08	0.14	26890	15157	9690	3023	17149	71075
ibm09	128x64	2.74e7	2.88e7	2.93e7	3.49e7	3.43e7	3.26e7	1.23	1.17	1.07	0.00	0.06	0.06	2512	1638	181	2889	17813	69225
ibm10	128x64	5.85e7	6.09e7	6.13e7	7.58e7	7.5e7	7.14e7	1.35	1.41	1.26	0.00	0.00	0.03	12392	8801	4416	4870	27439	104209
ibm11	64x128	4.19e7	4.37e7	4.36e7	4.85e7	4.78e7	4.59e7	1.25	1.10	1.02	0.17	0.19	0.15	3014	811	271	3440	17923	72010
ibm12	64x128	7.7e7	7.69e7	7.95e7	1.14e8	1.08e8	9.86e7	1.52	1.52	1.35	0.04	0.00	0.01	92152	55506	19838	4881	28042	104519
ibm13	64x128	5.14e7	5.36e7	5.41e7	7.31e7	6.82e7	6.4e7	1.51	1.38	1.31	0.02	0.07	0.11	74266	36200	20990	4393	27312	76768
ibm14	128x128	1.31e8	1.36e8	1.36e8	1.49e8	1.47e8	1.43e8	1.01	1.01	1.01	0.00	0.00	0.07	88	17	3	9508	83291	258155
ibm15	128x128	1.47e8	1.52e8	1.52e8	1.82e8	1.77e8	1.67e8	1.14	1.10	1.07	0.05	0.07	0.04	3971	1595	288	13285	74458	263706
ibm16	256x128	1.77e8	1.79e8	1.85e8	2.1e8	2.08e8	2.01e8	1.14	1.09	1.02	0.00	0.00	0.00	316	68	3	19548	114612	633760
ibm17	256x128	2.61e8	2.68e8	2.68e8	3.01e8	2.98e8	2.86e8	1.10	1.01	1.01	0.00	0.00	0.00	2187	1745	955	20603	160752	676954
ibm18	256x128	1.92e8	2e8	2.02e8	2.37e8	2.32e8	2.21e8	1.30	1.22	1.16	0.00	0.01	0.14	10717	4930	805	19904	162056	805802
avg.		1	1.04	1.05	1	0.97	0.93	1	0.95	0.90	-	-	-	1	0.55	0.26	1	6.62	24.91

TABLE XII
CONGESTION CONTROL COMPARISON BETWEEN MGP, MGP-CG.RD, AND MGP-CG ON FIVE REAL INDUSTRIAL DESIGNS FROM IBM

circuit	#cells	#nets	grid size	routed WL			max. cg			#edges-ov			#nets-ov			cpu (s)		
				MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg	MGP	MGP -cg.rd	MGP -cg
ind1	1099	1179	8×8	113	112	101	1.0	1.0	1.0	0	0	0	0	0	0	57	147	739
ind2	30997	32027	64×32	5520	5494	5369	1.1	1.1	1.1	1014	754	426	3608	3096	2566	1927	10247	40642
ind3	72940	73386	64×64	11601	11940	11863	1.3	1.1	1.03	9	3	1	133	57	24	5722	18527	59340
ind4	141862	153708	128×128	180094	180998	179473	2.53	2.53	2.53	5255	4783	4315	2809	2798	2634	58929	128036	361480
ind5	216111	221133	128×128	69545	69362	69188	1.7	1.7	1.7	1396	1310	1145	724	652	629	38773	92109	288096

by MGP-cg on average. This implies that the BBOX wire length is no longer a good metric for routability. A similar conclusion was also drawn in [30]. Meanwhile, the MGP-cg reduces any existing total overflow by 45%–74% on average and reduces either the routing wire length or the maximal boundary congestion, demonstrating that the congestion control performed in the placement phase can benefit the routing phase. It is also shown that by properly placing the modules/blocks/cells in the placement phase, good interconnect planning can be carried out in the routing phase. The results of MGP-cg with the reduced mode demonstrate the tradeoff between the runtime and the solution quality.

D. Congestion Control Experiments on Industrial Circuits

We also ran our program on five real industrial circuits from IBM (named ind1 to ind5, to avoid name confusion with the published IBM benchmark used in the previous section) on a Sun workstation running at 400-MHz frequency, followed by IBM's in-house legalization and routing tools. These circuits use IBM ASIC standard cell libraries, with feature size ranging from 0.15 to 0.25 μm. Some circuits have a number of preplaced macros (not counted in the numbers of cells).

Table XII shows the number of placeable cells (#cells), the number of nets (#nets), the grid size, the comparison of routed wirelength (in mm), maximum congestion, the number of overflowed edges,[13] and the number of nets that overflow. It confirms that the placement results generated by MGP-cg have less congestion than that by MGP, with fewer congested edges and nets, though it runs much slower. The reduced-mode MGP-cg.rd provides a tradeoff between the runtime and the quality of result.

VI. SUMMARY

We present a multilevel SA-based global placement algorithm (MGP) with congestion control integrated with an incremental A-tree algorithm and fast LZ-routing for fast congestion evaluation and optimization. Our placement engine also has a hierarchical area density control, which allows us to place both big and small clusters together. Our experiments show that our MGP program is competitive in both wire length and runtime. The congestion-driven MGP (MGP-cg) can significantly reduce routing congestion.

APPENDIX
DATA STRUCTURE FOR STORING WIRE USAGE

We only discuss the auxiliary data structure for storing segments on a horizontal routing layer because the data structure for a vertical routing layer can be similarly built. To simplify

[13]IBM tool reports the number of overflowed edges, not the total overflow.

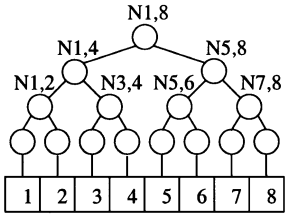


Fig. 6. Wire usage storing hierarchy in a row.

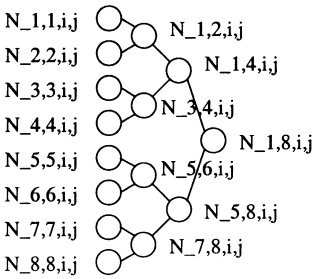


Fig. 7. Wire usage storing hierarchy combining multiple rows.

the discussion, we assume that each wire has a unit width. We first discuss how the wire usages in a given row are stored. In Fig. 6, a tree is formed to store the wire usages. Each node $N_{i,j}$ in the tree has a variable $U_{i,j}$ to store the number of segments that start from bin i and end at bin j. For example, node $N_{3,4}$ has $U_{3,4}$ to store all segments starting from bin 3 and ending at bin 4. If a wire segment cannot be properly stored in one node, it will be broken into a set of segments where each segment can fit into a node. We always choose the set with minimum cardinality. For example, a segment starting from bin 2 and ending at bin 8 will be stored at nodes $N_{2,2}$, $N_{3,4}$, and $N_{5,8}$. It is easy to see that for a row of 2^n bins, it takes less than $2n-2$ nodes to store a segment. For each node, we also have another variable $S_{i,j}$ that stores the summation of all wire usages of the nodes under it. For example, for node $N_{1,4}$, we will also store the summation of wire usages of nodes $N_{1,1}$, $N_{2,2}$, $N_{3,3}$, $N_{4,4}$, $N_{1,2}$, and $N_{3,4}$ ($S_{1,4} = U_{1,1} + U_{2,2} + U_{3,3} + U_{4,4} + U_{1,2} \times 2 + U_{3,4} \times 2$). It also takes, at most, $2n-1$ updates for inserting or deleting a segment.

The above data structure only helps to efficiently answer a query in a single row. For queries involving multiple rows, we need another auxiliary data structure. If we have 2^m rows, we also build a similar tree to store the sum of the usages of the adjacent rows. We use $N_{k,k,i,j}$ to represent the node $N_{i,j}$ on row k. We will build a similar hierarchy on these nodes to represent multiple-row usage. We define $S_{k,l,i,j} = \sum_{x=k}^{l} S_{x,x,i,j}$.

For example, in Fig. 7, we show the hierarchy generated from each node $N_{i,j}$ in every row from rows 1 and 8. For each update of a node, it requires m updates on the second multiple-row hierarchy. Therefore, the total updates for a segment is at most $2n-1+(2n-2)m$.

Given the above data structure, we can efficiently answer some usage queries. For a query asking for the usage of bins between rows k and l and columns i and j, if $N_{k,l,i,j}$ is a node stored in our data structure, it can be calculated by adding $S_{k,l,i,j}$ and all the $U_{k,l,x,y} \times (j-i+1)$ for all the ancestors of node $N_{k,l,i,j}$. For example, if we want to query the usage between rows 1 and 4, columns 1 and column 4, we only need to add $S_{1,4,1,4}$, $U_{1,4,1,4} \times 4$ and $U_{1,4,1,8} \times 4$. In general, given $2^m \times 2^n$ bins, if a query has a corresponding node in our tree hierarchy, it takes at most n lookups to find all the U values. For a query that can be broken down to $N_{k,l,i_1,j_1}, N_{k,l,j_1+1,j_2}, \ldots, N_{k,l,j_r+1,j}$, it takes at most $2n$ lookups to find all the U values. Finally, for any query, it takes at most $4mn$ node lookups to find all the U values.

Because the complexity of a query depends on whether the row numbers can fit into the tree hierarchy, we always use the minimum range that can cover our range to form the query in our LZ-router and use its density as an approximation of the queried density.

ACKNOWLEDGMENT

The authors would like to thank K. Konigsfeld and M. Mohan from Intel Corporation, G. Karypis from the University of Minnesota, Minneapolis, G. Chen and M. Romesis from the University of California, Los Angeles, for their helpful discussion, and S. Gao from IBM EDA for generating wire length distribution in Table I.

REFERENCES

[1] J. Cong, "An interconnect-centric design flow for nanometer technologies," *Proc. IEEE*, vol. 89, pp. 505–527, Apr. 2001.
[2] J. Cong and S. K. Lim, "Physical planning with retiming," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 2–7.
[3] J. Cong, S. K. Lim, and C. Wu, "Performance driven multi-level and multiway partitioning with retiming," in *Proc. Design Automation Conf.*, 2000, pp. 274–279.
[4] C.-C. Chang, J. Cong, Z. D. Pan, and X. Yuan, "Physical hierarchy generation with routing congestion control," in *Proc. Int. Symp. Physical Design*, 2002, pp. 36–41.
[5] M. Breuer, "Min-cut placement," *J. Design Automation Fault-Tolerant Computing*, vol. 1, no. 4, pp. 343–362, 1977.
[6] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 356–365, Mar. 1991.
[7] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 510–522, Apr. 1985.
[8] W. Swartz and C. Sechen, "New algorithms for the placement and routing of macro cells," in *Proc. Int. Conf. Computer-Aided Design*, 1990, pp. 336–339.
[9] W.-J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 349–359, Mar. 1995.
[10] C.-K. Cheng and E. S. Kuh, "Module placement based on resistive network," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 218–225, July 1984.
[11] R.-S. Tsay, E. S. Kuh, and C.-P. Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design Test Comput.*, vol. 5, pp. 44–56, Dec. 1988.
[12] G. Sigl, K. Doll, and F. Johannes, "Analytical placement: A linear or a quadratic objective function?," in *Proc. Design Automation Conf.*, 1991, pp. 427–432.
[13] J. Vygen, "Algorithms for large-scale flat placement," in *Proc. Design Automation Conf.*, 1997, pp. 746–751.
[14] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. Design Automation Conf.*, 1998, pp. 269–274.
[15] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placement?," in *Proc. Design Automation Conf.*, June 2000, pp. 477–482.

[16] S.-W. Hur and J. Lillis, "Mongrel: Hybrid techniques for standard cell placement," in *Proc. Int. Conf. Computer-Aided Design*, 2000, pp. 165–170.
[17] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Fast standard-cell placement for large circuits," in *Proc. Int. Conf. Computer-Aided Design*, 2000, pp. 260–263.
[18] T. F. Chan, J. Cong, T. Kong, and J. R. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 171–176.
[19] A. B. Kahng, S. Mantik, and D. Stroobandt, "Requirements for models of achievable routing," in *Proc. Int. Symp. Physical Design*, Apr. 2000, pp. 4–11.
[20] H.-M. Chen, H. Zhou, F. Young, D. Wong, H. Yang, and N. Sherwani, "Integrated floorplanning and interconnect planning," in *Proc. Int. Conf. Computer-Aided Design*, 1999, pp. 354–357.
[21] C.-L. E. Cheng, "RISA: Accurate and efficient placement routability modeling," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 690–695.
[22] S. Mayrhofer and U. Lauther, "Congestion-driven placement using a new multi-partitioning heuristic," in *Proc. Int. Conf. Computer-Aided Design*, 1990, pp. 332–335.
[23] W. Hou, H. Yu, X. Hong, Y. Cai, W. Wu, J. Gu, and W. H. Kao, "A new congestion-driven placement algorithm based on cell inflation," in *Proc. Asia South Pacific Design Automation Conf.*, 2001, pp. 605–608.
[24] T. Sadakane, H. Shirota, K. Takahashi, M. Terai, and K. Okazaki, "A congestion-driven placement improvement algorithm for large scale sea-of-gates arrays," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1997, pp. 573–576.
[25] P. N. Parakh, R. B. Brown, and K. A. Sakallah, "Congestion driven quadratic placement," in *Proc. Design Automation Conf.*, 1998, pp. 275–278.
[26] U. Brenner and A. Rohe, "An effective congestion driven placement framework," in *Proc. Int. Symp. Physical Design*, 2002, pp. 6–11.
[27] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Routability driven white space allocation for fixed-die standard-cell placement," in *Proc. Int. Symp. Physical Design*, 2002, pp. 42–47.
[28] R.-S. Tsay, S. Chang, and J. Thorvaldson, "Early wirability checking and 2D congestion-driven circuit placement," in *Proc. Int. Conf. ASIC*, 1992, pp. 50–53.
[29] M. Wang, X. Yang, and M. Sarrafzadeh, "Congestion minimization during placement," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1140–1148, Oct. 2000.
[30] X. Yang, R. Kastner, and M. Sarrafzadeh, "Congestion reduction during placement based on integer programming," in *Proc. Int. Conf. Computer-Aided Design*, 2001, pp. 573–576.
[31] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 69–79, Mar. 1999.
[32] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proc. Int. Conf. Computer-Aided Design*, 2001, pp. 396–403.
[33] A. Brandt, "Multi-level adaptive solutions to boundary-value problems," *Math. Comput.*, vol. 31, pp. 333–390, Apr. 1977.
[34] T. F. Chan and W. Wan, "Robust multigrid methods for nonsmooth coefficient elliptic linear systems," *J. Comput. Appl. Math.*, vol. 123, pp. 323–352, Nov. 2000.
[35] J. Cong and S.-K. Lim, "Edge separability based circuit clustering with application to circuit partitioning," in *Proc. Asia South Pacific Design Automation Conf.*, Jan. 2000, pp. 429–434.
[36] J. Cong and M. Romesis, "Performance-driven multi-level clustering with application to hierarchical FPGA mapping," in *Proc. Design Automation Conf.*, June 2001, pp. 389–394.
[37] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. Design Automation Conf.*, 1998, pp. 343–348.
[38] S. Rao, P. Sadayappan, F. Hwang, and P. Shor, "The rectilinear Steiner arborescence problem," *Algorithmica*, vol. 7, no. 2–3, pp. 277–288, 1992.
[39] J. Cong, A. B. Kahng, and K.-S. Leung, "Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 24–39, Jan. 1999.
[40] M. Huang, F. Romero, and A. Sangiovanii-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Proc. Int. Conf. Computer-Aided Design*, 1986, pp. 381–384.
[41] J. Rose, W. Klebsch, and J. Wolf, "Temperature measurement of simulated annealing placements," in *Proc. Int. Conf. Computer-Aided Design*, 1988, pp. 514–517.
[42] V. Betz and J. Rose, "VPR: A new packing placement and routing tool for FPGA research," in *Proc. ACM/SIGDA Int. Symp. FPGA*, 1997, pp. 213–222.
[43] K. Doll, F. Johannes, and G. Sigl, "DOMINO: Deterministic placement improvement with hill-climbing capabilities," *IFIP Trans. A: Comput. Sci. Technol.*, vol. A-1, pp. 91–100, 1991.
[44] [Online]. Available: http://nexus6.cs.ucla.edu/~cheese/ispd98.html
[45] [Online]. Available: http://er.cs.ucla.edu/Dragon
[46] G. Karypis, "Multilevel hypergraph partitioning," Comput. Sci. and Eng. Dept., Univ. Minnesota, Minneapolis, Tech. Rep. 02-25, 2002.
[47] J. Cong and P. H. Madden, "Performance driven multi-layer general area routing for PCB/MCM designs," in *Proc. Design Automation Conf.*, 1998, pp. 356–361.

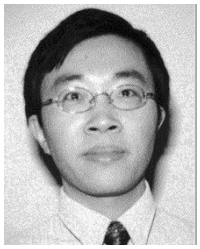
Chin-Chih Chang received the B.S. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1989, the M.S. degree from the State University of New York, Stony Brook, in 1993, and the Ph.D. degree from the University of California, Los Angeles, in 2002, all in computer science.

He joined Cadence Design Systems, Inc., San Jose, CA, in 2002. His research interests include VLSI CAD algorithms on placement and routing.

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.



Zhigang Pan (S'97–M'00) received the B.S. degree in geophysics from Peking University, Beijing, China, in 1992, and M.S. degrees in atmospheric sciences and computer science and the Ph.D. degree in computer science from the University of California, Los Angeles, in 1994, 1998, and 2000, respectively.

He has been a Research Staff Member at the IBM T. J. Watson Research Center, Yorktown Heights, NY, since December 2000. He was with Magma Design Automation during the summer of 1999. He has published 20 papers and has one U. S. Patent issued. His current research interests are focused on VLSI interconnect modeling, synthesis, planning and their interaction with physical design and logic synthesis, and low power designs.

Dr. Pan served as the Local Arrangement Chair of the 2002 ACM Great Lakes Symposium on VLSI (GLSVLSI) and the Publicity Co-Chair of the 2003 ACM International Workshop on System-Level Interconnect Prediction. He is also a Technical Program Committee Member of the Asia South Pacific Design Automation Conference, in 2003, and the GLSVLSI, in 2003. He received the Best Paper in Session Award from the SRC Techcon in 1998, an IBM Research Fellowship in 1999, the Dimitris Chorafas Foundation Award in 2000, and Outstanding Ph.D. Award from the Computer Science Department, University of California, Los Angeles, in 2001.

Xin Yuan (S'99) received the B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 1996 and 1998, respectively. She is currently working toward the Ph.D. degree in computer science at the University of California, Los Angeles.

Her research interests include VLSI physical design, placement, global routing, interconnect optimization, and technology mapping for FPGAs.