Routability-Driven Placement and White Space Allocation

Chen Li, Min Xie, Cheng-Kok Koh, *Senior Member, IEEE,* Jason Cong, *Fellow, IEEE,* and Patrick H. Madden, *Member, IEEE*

***Abstract*—We present a two-stage congestion-driven placement flow. First, during each refinement stage of our multilevel global placement framework, we replace cells based on the wirelength weighted by congestion level to reduce the routing demands of congested regions. Second, after the global placement stage, we allocate appropriate amounts of white space into different regions of the chip according to a congestion map by shifting cut lines in a top-down fashion and apply a detailed placer to legalize the placement and further reduce the half-perimeter wirelength while preserving the distribution of white space. Experimental results show that our placement flow can achieve the best routability with the shortest routed wirelength among publicly available placement tools on IBM v2 benchmarks. Our placer obtains 100% successful routings on 16 IBM v2 benchmarks with shorter routed wirelengths by 3.1% to 24.5% compared to other placement tools. Moreover, our white space allocation approach can significantly improve the routability of placements generated by other placement tools.**

***Index Terms*—Circuit placement, design automation, routability, white space allocation.**

I. Introduction

PLACEMENT plays a fundamental and critical role in the physical design of integrated circuits. The traditional objectives of placement are to minimize area and wirelength. Among these two objectives, area optimization becomes less useful in the fixed-die design mode as we are given an upper bound on the available area. Total wirelength is not of direct interests to a circuit designer; of greater importance are the routability, delay, and power objectives during the placement stages due to timing and layout closure problems brought by the aggressive scaling of semiconductor technologies. Nevertheless, half-perimeter wirelength (HPWL), also called bounding-box (BBOX) wirelength, is the most typical objective of a placement tool as it is easy and simple to compute. Most importantly, HPWL is positively related to the actual interconnect wirelength, which has a strong correlation with routability and whose parasitics directly determine the timing and power of a circuit. Unfortunately, a placement with shorter HPWL may still be unroutable because the routing resources and routing demands at some parts of the chip are not matched. Therefore, achieving routability becomes one of the most difficult tasks of the placement flow of current designs.

Routability of a placement is determined by two factors, routing demands and routing resources throughout the chip. Consequently, routability can be optimized by either reducing routing demands or increasing routing resources at congested regions in a placement. We shall review routability improvement techniques in these two categories in the following. Please also refer to [19] for a good survey of routability-driven placement.

One approach to routability control is to reduce routing demands. Minimizing the wirelength of a placement may reduce the total routing demand of the placement. However, routing failures can still occur in some congested regions. Reducing the routing demands in congested regions is usually performed in the global placement stage, as the cell locations are adjusted only slightly in the detailed placement step. In addition, net topology manipulation can also be considered for optimization, so that good routability can be obtained without much increase in wirelength cost. In [25], the congestion of a placement bin, which is estimated by Rent rule implicitly, is incorporated into the HPWL cost function in a simulated annealing flow. The approach reserves routing resources for global nets by avoiding excessive usage by local nets. However, the congestion contributed by internal nets within a bin is ignored; hence, the accuracy of the congestion estimation is reduced. Furthermore, the topology of the global nets is also ignored. In **APlace** [28], the congestion estimated by a probability-based approach is incorporated into the density penalty function, which is used in conjunction with a logarithm-sum-exponential wirelength objective function, to obtain a placement with cells roughly evenly distributed and with improved routability. A postprocessing step of moving cells with Steiner tree reconstructions during placement is used in [40]. The movement of a cell is limited, and reconstructing a Steiner tree for each cell movement is still expensive. The **mPG** placer [15], which follows a multilevel simulated annealing flow, incorporates into its cost function routing congestion estimated by a tree-based global router. It

Manuscript received June 7, 2005; revised September 30, 2005 and December 25, 2005. This work was supported in part by the Semiconductor Research Corporation under Contract 2003-TJ-1091 and Project 947.1, in part by the National Science Foundation under Awards CCF-0430077 and CCR-9984553, and in part by an IBM Faculty Partnership Award. This paper was presented in part at ICCAD, 2004. This paper was recommended by Associate Editor J. Lillis.

C. Li was with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA. He is now with Magma Design Automation, Inc., Santa Clara, CA 95054 USA (e-mail: chenli@magma-da.com).

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

C.-K. Koh is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: chengkok@ecn.purdue.edu).

P. H. Madden is with the Department of Computer Science, Binghamton University, Binghamton, NY 139002 USA (e-mail: pmadden@cs.binghamton.edu).

Digital Object Identifier 10.1109/TCAD.2006.884575

reduces the routing overflow by 50%. However, the runtime is increased by at least five times due to the high complexity involved in the construction of routing trees.

Orthogonal to routing demand reduction is the approach that increases routing resources in congested regions by expanding the region. In fixed-die placement, where white space is typically present, allocating white space to increase routing resources in congested regions is a common method to relieve congestion. White space allocation can be done both during the global placement stage (or at the end of global placement) or during the detailed placement stage. Depending on the specific stage, the congestion estimation method as well as the bin granularity may differ. Allocating white space to a congested region can be achieved by inflating cells [6], [24]. In BonnPlace [6], congestion levels of initial partitions are first estimated by taking both interregion nets and intraregion nets into account. Congestion due to interregion nets is estimated by a probabilistic method using a routing grid structure, and congestion due to intraregion nets is estimated by pin density within this region. After that, white space is allocated by expanding cell areas in the congested regions. In [37], congestion in the horizontal or vertical direction is relieved by expanding the region in that direction and shrinking the region in the orthogonal direction in a quadratic placement framework. In [45], expansion of congested regions is formulated as an integer-programming problem such that a subset of congested regions is selected to be expanded with proper amount of white space. Congestion-driven Dragon [44] allocates white space in two steps. White space is first distributed into rows and then into bins within a row. As that may increase the row imbalance, Dragon imposes a lower bound and an upper bound on the amount of white space available in a row. In all these approaches, providing white space to relieve congestion and reducing wirelength become conflicting objectives. Therefore, these approaches improve routability of placements, at the expense of longer wirelengths.

We also observe that modern designs may contain various amount of white space. It is obvious that evenly distributed or randomly distributed white space may increase the wirelength. To avoid that, white space management methods in [5] and [1] were proposed. In [5], an analytical placement technique is used to generate the constraints for the partitioner during the partitioning-based placement flow. In [1], some part of white space is inserted as filler cells before global placement. However, as these white space management methods are not guided by congestion information, they may not be effective in reducing congestion.

In this paper, we present a two-stage routability-driven placement flow. We propose a congestion-driven multilevel global placement method that enhances the routability during global placement by replacing cells to avoid congested regions. Experiments showed that compared with its wirelength driven mode, it significantly reduces the global routing overflow and enhances the detailed routing completion rate. Final routed wirelengths are also reduced. We also propose a congestion-driven white space allocation method that allocates white space after the global placement stage to provide appropriate routing resources to congested regions without a great perturbation on wirelength. Experiments show that the proposed white space allocation flow can significantly improve routability of placements generated by various latest placement tools. Combining the proposed routability-driven global placement and white space allocation methods, we achieve placements with the best routability among the publicly available placement tools, with all IBM-Dragon version 2 easy and hard benchmark circuits [44] successfully routed. Routed wirelengths are also reduced by 2.3% to 24.5% compared to other placement tools.

II. Congestion Estimation

Existing congestion estimation methods can be divided into two categories as in [18], topology-based methods (TP-based), where routing trees are explicitly constructed on some routing grid, and topology-free methods (TP-free), where no explicit routing is needed.

Among the two categories, TP-free modeling is usually faster. This category includes BBOX-based modeling [17], probabilistic analysis-based modeling [24], [31], Rent's rule-based modeling [46], and pin density-based modeling [6]. TP-based modeling methods usually construct a Steiner tree for each net in the netlist. Such modeling method can generate an upper bound for the routability estimation. If the topology generated by a TP-based modeling method is similar to what the after-placement-router does, high fidelity of the modeling can be expected. The Steiner trees can be either precomputed [34], or constructed dynamically [15]. More detailed discussions of congestion methods are available in [19]. In this paper, we take the TP-based approach developed in [15] and construct spanning tree-based routing topology to estimate congestion.

A. Routing Resource Estimation

In our workflow, we calculate the resource in a routing region as

$$\mathrm{RR}_r = \sum_{i=1}^{n} \frac{A_i}{w_i + s_i}$$

where n is the number of routing layers, w_i and s_i are width and space of metal wires in layer i, respectively. A_i denotes the area of layer i available for routing over the region r. In particular, layer 1 and layer 2 might be utilized for routing within the cells; thus, part of these two layers might not be available for signal routing. Layer 3 and above are typically available for signal routing. We do not consider prerouted nets when measuring A_i because the benchmarks that we used in our experiments do not contain such information. However, our approaches can be easily extended to consider different routing resources for different regions.

B. Routing Demand Estimation

To estimate the congestion of a placement, we build a routing grid of $m \times n$ over the chip on each level. As for routing demand estimation, the most accurate value usually comes from global routing itself. However, due to its complexity, global routing cannot be performed very frequently, as the placement

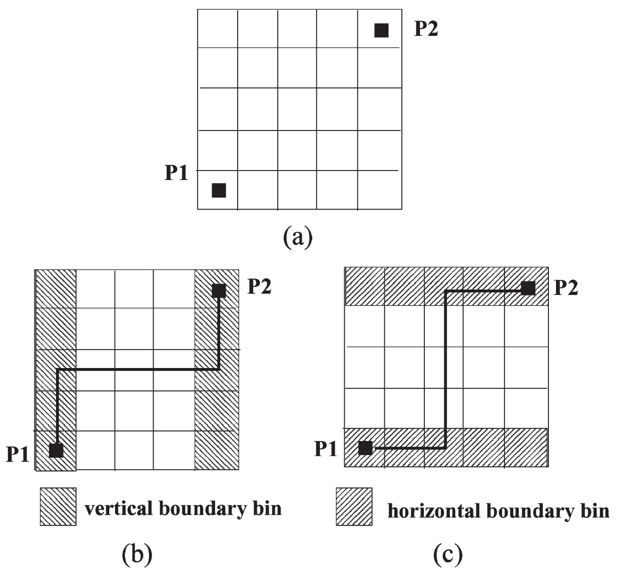

Fig. 1. Illustration of HVH and VHV routing selection of LZ router.

may go through many changes in each stage, changing the routing congestion at the same time. Therefore, full-blown global routing at every step is both expensive and unnecessary. A congestion estimator with good fidelity and high sensitivity to placement changes is better suited for our purpose. In our framework, we start with a minimum spanning tree for each net, and decompose multipin nets into two-pin connections. Then, we use the two-bend LZ router developed in [15] to determine the topology for each two-pin connection.

Fig. 1 gives an illustration of the LZ router. It uses auxiliary data structures to find good quality routes by performing a binary search of the possible routes for a two-pin net. For a net connecting two pins, P_1 and P_2, which are bounded by a rectangle bounding box B [Fig. 1(a)], the vertical–horizontal–vertical (VHV) routing pattern [Fig. 1(b)] is chosen if its possible maximum wire density is smaller than that of horizontal–vertical–horizontal (HVH); otherwise, the HVH routing pattern [Fig. 1(c)] is chosen. Suppose the VHV routing pattern is chosen, the router makes a horizontal cut on B and selects the partition with a lower wire density to route the horizontal segment. The router performs the cut-and-select process on the lower wire density partition recursively, as in binary search, until the location of the horizontal segment narrows to a single row. Note that this method may not find the location with the smallest congestion level. Despite its simplicity, the fidelity of LZ-shaped routing has been confirmed by a previous study [41], which shows that a large portion of the nets in the netlist are routed using L- or Z-shaped topology. Comparing our approach to the approach proposed in [42], the major difference is that our approach tries to avoid congestion for the routing trunk through a hierarchical density query, which takes $\log(m)$ time, where m is the total number of locations to be searched. Throughout the congestion estimation, we commit to only one routing configuration, instead of using any probability. It was shown in [16] that the complexity to route a two pin net on a given g_x by g_y bin structure for a two pin net that spans x by y bins is $\mathrm{O}(\log(x+y)\times\log(g_x+g_y))$. However, the complexity of using a congestion stamp might still be much higher due to the updates that have to be performed on all the bins in a probabilistic approach.

Once the topology for each net is determined, the routing demand by net_k on grid cell gc_{ij} is calculated as

$$\mathrm{RD}_{\mathrm{net}_k,\mathrm{gc}_{ij}} = \begin{cases} w_{\mathrm{net}_k}, & \text{if net}_k \text{ crosses gc}_{ij} \\ \alpha\times\mathrm{pin}_{\mathrm{net}_k}\times w_{\mathrm{net}_k}, & \text{if net}_k \text{ is within gc}_{ij} \\ 0, & \text{otherwise.} \end{cases}$$

Here, w_{net_k} is the wire width of net_k, α is a user specified constant, and $\mathrm{pin}_{\mathrm{net}_k}$ is the number of pins in net_k. We use $\alpha = 0.25$ in our experiments. The routing demand on grid cell gc_{ij} is calculated as

$$\mathrm{RD}_{\mathrm{gc}_{ij}} = \sum_{\text{for}} \text{all net}_k \mathrm{RD}_{\mathrm{net}_k,\mathrm{gc}_{ij}}.$$

We compute the routing demand for any rectangular region r as

$$\mathrm{RD}_r = \sum_{\mathrm{gc}_{ij}\in r} \mathrm{RD}_{\mathrm{gc}_{ij}}.$$

Then, we define the overflow on a grid cell gc_{ij} as

$$\mathrm{OVL}_{\mathrm{gc}_{ij}} = \max(0, \mathrm{RD}_{\mathrm{gc}_{ij}} - \eta\mathrm{RR}_{\mathrm{gc}_{ij}})$$

where η is a multiplier that is assigned different values during the global routing stage and the white space allocation phase.

Intuitively, a grid cell is considered to be congested if its routing demand is larger than its routing resource. Consequently, during the global placement stage, we set η to be 1, so that the global placer tries to even out the routing demand only if the routing demand exceeds its routing resources. While it is important to know the "absolute" congestion levels during the global routing phase, relative congestion levels are of greater importance in the white space allocation stage. During the white space allocation stage, the purpose is to spread out the routing demand over the entire placement region such that the routing demand in a grid cell matches its routing resource. As it is important for the white space allocation phase not to make drastic changes to the routing tree structures, we set the multiplier η according to the following formula:

$$\eta = \frac{\mathrm{RD}_{r_P}}{\mathrm{RR}_{r_P}}$$

where RD_{r_P} and RR_{r_P} are the routing demand and resource of the whole placement region. Note that $\eta \leq 1$ is a necessary condition for a placement to be routable.

We also define the overflow of any region r as

$$\mathrm{OVL}_r = \sum_{\mathrm{gc}_{ij}\in r} \mathrm{OVL}_{\mathrm{gc}_{ij}}$$

instead of the difference between the routing demand and resource of region r to consider the possible uneven distribution of routing demand on these grid cells.

We define the overflow caused by net_k as

$$\mathrm{OVL}_{\mathrm{net}_k} = \sum_{\{\mathrm{gc}_{ij}|\mathrm{OVL}_{\mathrm{gc}_{ij}}>0\}} \mathrm{RD}_{\mathrm{net}_k,\mathrm{gc}_{ij}}.$$

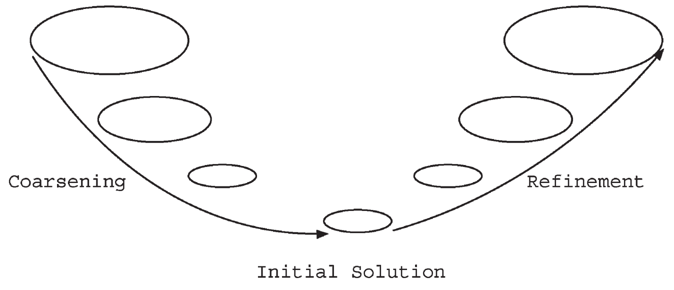

Fig. 2. Multilevel global placement. It consists of a coarsening phase, an initial solution generation, and a refinement phase.

The overflow of a placement P is defined as

$$\mathrm{OVL}_P = \sum_{i=1}^{m}\sum_{i=1}^{n}\mathrm{OVL}_{\mathrm{gc}_{ij}}.$$

Note that according to our definition, the sum of $\mathrm{OVL}_{\mathrm{net}_k}$ over all nets may be greater than OVL_P due to double counting.

III. Routability Control in Global Placement

The global placement in our workflow is built upon mPL [11], [12], a multilevel standard cell placement engine. Fig. 2 shows its flow, including a coarsening phase in which cells are recursively aggregated into clusters, an initial placement generation at the coarsest level, and a refinement phase that refines each coarser level placement solution to obtain a finer level solution.

Compared with other state-of-the-art placement tools, mPL produces competitive placement results in terms of HPWL. However, mPL gives little consideration for routing congestion. The placement it produces may be overly congested for subsequent routing. Since refinement is the stage where the cell locations as well as the net topologies are determined, our focus for routability control is mainly on the refinement on each level.

The congestion driven refinement during global placement begins with a congestion estimation using a fast LZ router. This is followed by a normal wirelength minimization step. In the end, a subset of cells is chosen and replaced to adjust the topology of the nets incident on them, so that the routing demand for current placement can be reduced.

To reduce the routing demand, we selectively replace a subset of the cells after the wirelength minimization step, so that the topology of the nets incident on them can be adjusted. At all levels except the finest, we are actually replacing the cells. In other words, we treat the clustered nets incident on these clusters as normal ones with similar width and spacing requirements.

A secondary objective during this process is to reduce the wirelength. Migrating cells for routability enhancement has been proposed in [15] and [25]. However, in [25], the focus is mainly on reducing local net routing demands, and the routing topology of global nets are ignored. The approach in [15] is quite indiscriminate about which cells should be replaced. As a consequence, there are many candidate cells for replacement, resulting in prohibitive runtime. In our workflow, candidates for cell replacement are selected based on the routing topology of nets incident on them.

Algorithm: Congestion Driven Cell Re-placement

```
Sort nets in descending order of overflow;
Pick the first $s$ nets such that $\sum_{i=1}^{s} OVL_{net_i} \geq OVL_P$;
for $i$ = 1 to $s$ do
    for each cell $c$ connected by $net_i$ do
        $WL_{min}$ = $\infty$;
        Determine grid cell $g_{ij}$ for $c$'s optimal HPWL;
        for each neighboring grid cell $gc_{i'j'}$ of $g$ do
            Place $c$ in $gc_{i'j'}$;
            Re-determine the topology of the nets incident
              on $c$;
            Evaluate $WL_c = \sum wgt_{net_k} \times WL_{net_k}$;
            if $WL_{min} > WL_c$ do
                $gc_{min}$ = $gc_{i'j'}$;
                $WL_{min}$ = $WL_c$;
            end if
        end for
        Place $c$ in $gc_{min}$;
    end for
end for
```

Fig. 3. Algorithm for congestion driven cell replacement.

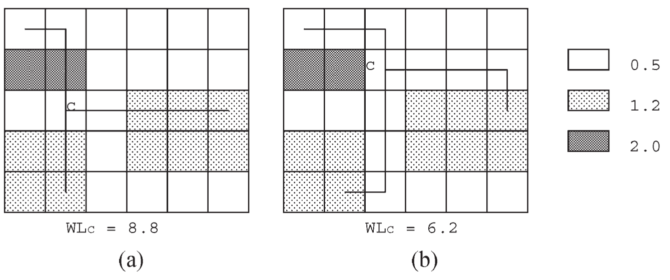

Fig. 4. Congestion driven cell replacement. The legend corresponds to the congestion in different routing regions. (a) Original placement of cell c in the optimal location for HPWL gives a weighted wirelength of 8.8. (b) Replacement of c in a neighboring region gives a weighted wirelength of 6.2.

Fig. 3 gives a description of this process. We sort the nets according to the overflow they cause in descending order, and pick the first s nets, such that the sum of their overflow is more than the total overflow of the current placement. The cells connected by these nets will be replaced.

For a cell c, we first determine the grid cell gc_{ij} corresponding to its optimal location for HPWL. The set of candidate grid cells $\{\mathrm{gc}_{i'j'}\}$ in which c can be replaced are within a certain distance d from the optimal location of c, i.e., $|i'-i| + |j'-j| \leq d$. To evaluate the cost associated with placing c in a candidate grid cell, the topology for the nets incident on c is redetermined using the LZ router. Then, the cost is computed using a weighted wirelength of all the nets incident on c as follows:

$$\mathrm{WL}_c = \sum \mathrm{wgt}_{\mathrm{net}_k} \times \mathrm{WL}_{\mathrm{net}_k}$$

where $\mathrm{wgt}_{\mathrm{net}_k}$ is the weight on net_k, calculated as the average congestion of the grids cell net_k crosses, and $\mathrm{WL}_{\mathrm{net}_k}$ is the HPWL of net_k. In the end, c is placed at the center of the region that results in the shortest weighted wirelength.

Fig. 4 shows an example of this process. The congestion levels in different routing regions are shaded accordingly

Algorithm: White Space Allocation
Construct a slicing tree
Estimate congestion of nodes of the tree
Adjust cut line locations from the top downward
Legalize placement and locally minimize HPWL

Fig. 5. Overview flow of white space allocation.

(see the legend). Starting from the optimal location for HPWL, we search the neighborhood for a region that gives the shortest weighted wirelength. In this example, the optimal location for the HPWL gives a weighted wirelength of 8.8 due to the congestion on each route, whereas a neighboring region will give a smaller weighted wirelength of 6.2.

The total cell area in a routing region might exceed its capacity after this procedure. To slightly relieve this, we lock the recently replaced cells and try to rebalance the area density with ripple move [26]. This replacement process is repeated for several passes before proceeding to the next level of the refinement stage.

IV. White Space Allocation

In our global placement flow, the amount of white space in a region may not accurately match its routing demand. Therefore, we further apply a white space allocation step to allocate appropriate amounts of white space into congested regions. In the context of fixed-die placement, this step does not increase the chip area as white space is already present.

In Dragon [44], based on the total congestion of each row, white space is first distributed into rows proportionally. Then, white space is allocated into various locations within each row. However, cells between different rows might sheer due to different amount of white space inserted in different location. Dragon further applies simulated annealing technique to adjust cell locations to reduce this effect of sheering. However, if the amounts of white space in rows range widely or some rows contain far too less white space, the effectiveness of simulated annealing deteriorates (noting that simulated annealing process should still roughly maintain the amount of white space in each row). Therefore, Dragon imposes a lower bound constraint and an upper bound constraint on the amount of white space distributed to each row, which, on the other hand, deteriorates the matching between white space and congestion. Unlike Dragon's two-step allocation approach [44], we assign appropriate amounts of white space to congested regions in a hierarchical flow, which can mitigate the effect of sheering.

The flow of our congestion-driven white space allocation is shown in Fig. 5. We first construct a slicing tree based on the geometric locations of all cells. We estimate the congestion level at each node of the tree and then adjust the cut line location at each node in a top–bottom fashion to distribute the white space to two child nodes. After cut line adjustment, we apply a detailed placer (legalization and local minimization) to remove overlaps and further reduce HPWL while preserving the white space distribution.

By using this slicing tree-based white space allocation, we observe two advantages of this approach. First, the slicing tree captures the relative locations among cells; thus, it maintains the quality of the original placement. Second, this approach is able to adjust the routing resources of the whole chip region even when nearby resources are not enough for local congestion.

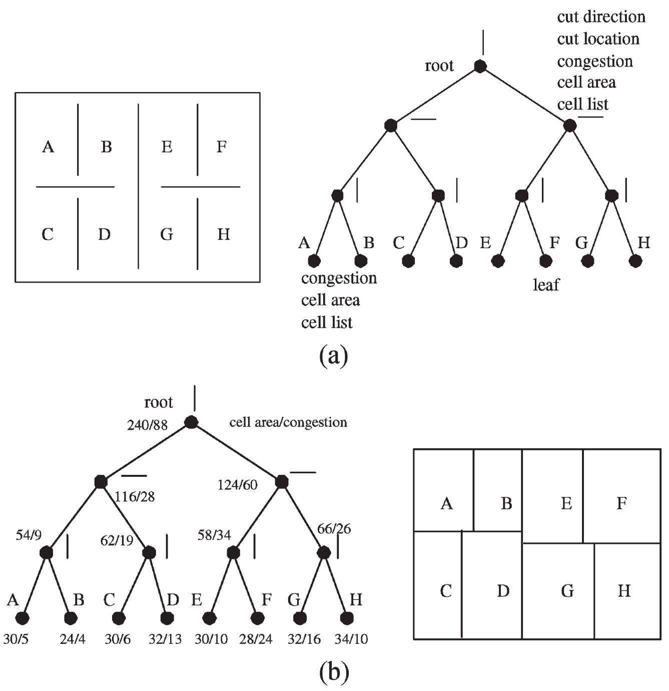

Fig. 6. (a) Slicing tree and its corresponding cut lines and regions. (b) A slicing tree after congestion estimation and regions after cut lines adjustment.

A. Slicing Tree and Congestion Estimation

Given any global or detailed placement, we first construct a slicing tree using a method that is similar to a partitioning-based global placement flow. The difference is that the partitioning here is performed based on the geometric locations of the cells (instead of the minimization of cut size).

We recursively partition the placement, starting from the full chip level until every region contains a small number of cells. Cut directions are determined by comparing the aspect ratio of this region with a fixed value. Each cut line geometrically bisects a region evenly. For a region, once its cut direction and cut location are determined, all cells whose centers are located at the left of the cut line (if cut vertically) or above the cut line (if cut horizontally) form the left child of that tree node. The remaining cells in the region form the right child of that node. This is essentially the slicing tree data structure used to capture a slicing floorplan [as shown in Fig. 6(a)]. Every node in the tree maintains its cut direction, cut location, congestion, total cell area as well as cell list.

Now, we estimate the congestion level of the nodes in the slicing tree in a bottom-up fashion. The congestion level of a leaf node can be estimated by the total routing overflow of the grid cells contained in this leaf node. The congestion level of an internal node can then be computed through a postorder traversal of the tree by adding up the congestion levels of two child nodes.

B. Cut Line Adjustment

After performing congestion analysis on all tree nodes, we shift cut line locations in the nodes of the slicing tree by traversing the nodes in a top-down fashion such that the amounts of white space allocated to the two child nodes are linearly proportional to their congestion levels. Consider a region r with lower left corner (x_0, y_0), upper right corner (x_1, y_1), and the original vertical cut direction at $x_{\mathrm{cut}} = (x_0 + x_1)/2$. Thus, the area of this region is $A_r = (x_1 - x_0)(y_1 - y_0)$. Assume that the total area of cells for left subregion r_0 and right subregion r_1 are S_0 and S_1, and the corresponding congestion levels are OVL_0 and OVL_1, respectively. We want to distribute the total amount of white space, which is $(A_r - S_0 - S_1)$, to the two subregions such that the amounts of white space in the two subregions are linearly proportional to their congestion levels. Thus, the amount of white space allocated to subregion r_0 is $(A_r - S_0 - S_1)[\mathrm{OVL}_0/(\mathrm{OVL}_0 + \mathrm{OVL}_1)]$. Then, the new cut line location x'_{cut} can be derived as follows:

$$\gamma = \frac{S_0 + (A_r - S_0 - S_1)\frac{\mathrm{OVL}_0}{\mathrm{OVL}_0 + \mathrm{OVL}_1}}{A_r}$$

$$x'_{\mathrm{cut}} = \gamma x_1 + (1 - \gamma) x_0$$

where γ is the ratio of the left subregion area to A_r after the cut line adjustment.

This step is similar to a top-down partitioning-based global placement except that the cut direction, the cut location and subnetlists are all known. With the cut line being shifted, cell locations in two subregions are also scaled accordingly.[1] After cut line adjustment and cell location scaling, we obtain a global placement that contains overlaps even if we start with a legal placement. Moreover, cells may not be placed along a row.

The cut line shifting approach can be traced back to [7] or earlier. In [7] and their recent work [10], both horizontal and vertical cut lines are shifted to satisfy area constraints. Due to the restriction of a horizontal cut line be located at a row boundary, however, the area constraints may not be met completely. Furthermore, they did not consider congestion during cut line shifting. In our approach, cut line adjustment approach is performed in the same spirit as fractional cut [3], where horizontal cuts are not aligned with row boundaries, and can be shifted to any location within a row. We illustrate the region changes before and after cut line adjustment by an example in Fig. 6(b). In this example, we show the total cell area and congestion level at every tree node of the slicing tree. Cut lines are adjusted from top to bottom such that the amounts of white space in the subregions are proportional to their congestion levels.

C. Legalization and Local Minimization

After cut line adjustment and cell location scaling, the cells may overlap and they may not be placed along rows. A detailed placer is required to legalize the placement. This detailed placer should be able to maintain the locations of white space such that the white space is not greatly redistributed. As the detailed placer DOMINO [22] generates packed placements to minimize HPWL, it does not fit our purpose.

We propose a detailed placer to preserve white space distribution and further reduce HPWL. This detailed placer contains two steps, the legalization step and the local minimization step. We first adopt a greedy legalization algorithm to remove the overlaps. This legalization algorithm was first proposed in [23] and was extended in [29]. Then, we locally minimize HPWL using a sliding window approach [3], [9].

Although we do not explicitly consider routability in the detail placement stage, we observe that minimizing HPWL (without great perturbation to white space) during this stage leads to better routability. Congestion-weighted HPWL as in Section III can be used to further address routability.

1) Legalization Step: In the legalization step, all cells are aligned to sites and overlaps between cells are removed. We sort all cells in the increasing order of x-coordinates of their centers. Starting from the leftmost cell, we determine one cell at a time the row and site that the cell belongs to by minimizing the relocation cost. In this paper, relocation cost is defined as the cost incurred in moving a cell from the original location to a new location, which is the distance between the original location and new location. Details of this legalization step can be found in [29].

2) Local Minimization by Permutation: In the local minimization step, the wirelength is further reduced by searching locally the minimum wirelength of a window containing a small number of cells and possible white space. We slide the window over the entire chip from left to right, and from bottom to top, overlapping with the previous window. This approach has been used in various placers, such as **Feng Shui** [3] and **CAPO** [8]. **Feng Shui** applied a dynamic programming approach to assign cells to single row or multiple rows, while **CAPO** applied a branch and bound approach to consider several cells in a one-row window. We propose a near-exhaustive approach to permute cells in a one-row and two-row window.[2] In our experiments, each window contains a total of six real cells and/or white-space cells, where a white-space cell occupies contiguous empty sites within a prespecified width in a row (a contiguous white space that is too wide is represented by several white-space cells). Each window overlaps with a previous window by three real cells and/or white-space cells.

If a window covers only one row, all permutations of cells are feasible; we permute real cells as well as white-space cells to find a best ordering of them in a row in terms of HPWL. The permutation of cells in a two-row window is not as straightforward. Let $N(\leq 6)$ denote the number of real cells (hereafter referred to as "cells") in the window. We now present our proposed heuristic for the permutation of N cells in a two-row window, which is given in Fig. 7, in the remainder of this section.

[1]In our preliminary work [30], cells in a region were temporarily located at the center of that region during the process of cut-line shifting.

[2]In our preliminary work [30], we used a heuristic to permute cells in a window based on estimated HPWL. Our proposed approach in this paper obtains 1.0% shorter routed wirelength at a penalty of 9.4% longer total runtime of our two-stage placement flow. However, the total placement and routing time is reduced by 5.4%.

```
Algorithm: Partition(C, C_0, last_selected, best_cost, r_i, x_i)
$LB_0 = TotalWidth(C) - w_1$;
$UB_0 = w_0$;
if ($TotalWidth(C_0) >= LB_0$)
    $cost = EvalCost(C_0, C - C_0)$;
    if ($cost < best\_cost$)
        $best\_cost = cost$;
        save cell locations to $r_i$, $x_i$;
    endif
endif
for ($k = last\_selected + 1$; $k <= N$; $k++$)
    if ($TotalWidth(C_0) + Width(c_k) <= UB_0$)
        Partition($C$, $C_0 \cup \{c_k\}$, $k$, $best\_cost$, $r_i$, $x_i$);
    else
        break;
    endif
endfor
```

(a)

```
Algorithm: Local Minimization by Two-Row Permutation
for ($r = 0$ to $(LastRow - 1)$)
    initialize window ($C$, $w_0$, $w_1$, $x_0$, $x_1$) to cover the
        leftmost of row $r$ and $(r + 1)$;
    while (window is not outside the right boundary of
        the chip)
        sort $C$ in increasing order of their widths;
        initialize $best\_cost = \infty$, $C_0 = \emptyset$;
        Partition($C$, $C_0$, 0, $best\_cost$, $r_i$, $x_i$);
        slide window right, overlapping with current
            window;
    endwhile
endfor
```

(b)

Fig. 7. (a) Algorithm for partitioning N cells in a two-row window. (b) Algorithm for performing local minimization by two-row permutation.

In this algorithm, $C = \{c_i\}$, $i \in \{1, \ldots, N\}$ is a set of cells contained in the window, sorted in an increasing order based on their widths. Let w_0 and w_1 (x_0 and x_1), respectively, be the widths (x-coordinates of the first sites) of the lower row (r_0) and upper row (r_1) in the window. In order to accommodate all cells in the window, $(w_0 + w_1)$ should be larger than or equal to TotalWidth(C), i.e., the total width of cells in C. A lower bound LB$_0$ and an upper bound UB$_0$ of total widths of cells in r_0 can be easily found, which are $(\text{TotalWidth}(C) - w_1)$ and w_0, respectively. These two bounds can be used to determine the feasibility of a partitioning solution.

When $(w_0 + w_1) > \text{TotalWidth}(C)$, there is white space inside this window. The existence of white space allows for more feasible partitions of cells into two rows. Thus, we can explore more partitioning solutions to reduce wirelength. From our experience, although white space is redistributed by local optimization within this window and each window is overlapped with adjacent windows, it is unlikely that white space would migrate over a long distance. Thus, we can maintain the original distribution of white space in the macroscopic view (see Fig. 8).

Let C_0 and C_1 $(C_1 = C - C_0)$ denote two disjoint sets of C, that are assigned to rows r_0 and r_1, respectively. In this algorithm, we construct all feasible partitionings recursively. Given a feasible partitioning, C_0 and C_1, let $c_{\text{last\_selected}}$ be the widest cell last added to C_0. The algorithm pushes a candidate cell c_k, which is wider than $c_{\text{last\_selected}}$, into C_0. If the upper bound constraint is not violated, this partitioning is feasible, and its wirelength cost is evaluated by EvalCost(C_0, C_1). To evaluate the wirelength cost of a partitioning C_0 and C_1, we exhaustively permute C_0 in row r_0 and C_1 in row r_1 to find the best wirelength. For efficiency, cells in row r_0 are permuted independently of the permutation of cells in row r_1. This method provides an acceptable runtime with little sacrifice in placement quality. In summary, we exhaustively search the best permutation in terms of HPWL among all feasible partitionings of cells into two rows.

V. Experimental Results

In the following, we use **mPL-R** (Routability-driven mPL) to denote our global placement flow, **WSA** (White Space Allocation) our white space allocation flow, **mPL-R+WSA** the combined flow.

We evaluate the effectiveness of our tools (**mPL-R**, **WSA** and **mPL-R+WSA**) in achieving routability by comparing them with several state-of-the-art academic tools, including **Dragon** 3.01 [44], **CAPO** 9.3 [8], **Feng Shui** 5.1 [3], and **mPG** 1.0 [16], a leading-edge industrial placement tool **Cadence QPLACE** (in SEULTRA 5.3). **Dragon** is run with -fd option for congestion-driven mode (therefore shown as Dragon-fd in the following tables). **CAPO** is run with -uniformWS and -tryHarder for better routability (thereby with longer runtime). **mPG** is run using the wirelength objective instead of the congestion minimization objective due to the huge runtime and memory usage of congestion-driven **mPG**.[3] As **mPG** generates global placements with overlaps, we apply **QPLACE ECO** mode to obtain the final placements. Routability and routed wirelength of placements generated by these tools are evaluated by **Cadence WROUTE** (in SEULTRA 5.3) with all settings on default.[4] As we are unable to obtain the binary of routability-driven **APlace** for comparison, we use the partial routability results published in [28] (on ibm01/02/07/08 easy and hard benchmarks) for comparison.

All experiments are performed on the complete set of IBM-Dragon version 2 easy and hard benchmarks. These benchmarks were converted from ISPD98 [4] by mapping cells to commercial standard cell library by authors of **Dragon** [44] to evaluate routability of placements. The characteristics of these benchmarks, including number of cells, number of nets, ratio of white space to the chip area, and routing layers, are shown on Table I. The relative amount of white space in these benchmarks varies from about 5% to 15%.

Dragon, **CAPO**, and **mPG** are each run for five times for each benchmark. **Feng Shui**, **QPLACE**, and our tools are

[3]For congestion-driven mPG, we obtain placements only for ibm01 and ibm02.

[4]We found that routing results obtained from WROUTE in SE 5.3 were better than those obtained from Nanoroute (V4.10-p554.001 in SOC Encounter 4.1) based on placement solutions from Dragon and our flow. However, Nanoroute or newer version of WROUTE might produce better routing results on placements generated by other placers [39]. As WROUTE in SE 5.3 with the same configuration is used throughout our experiments, we believe that it can serve as a tool to evaluate routability of placements, although the results may not be representative of what can be achieved by other routers.

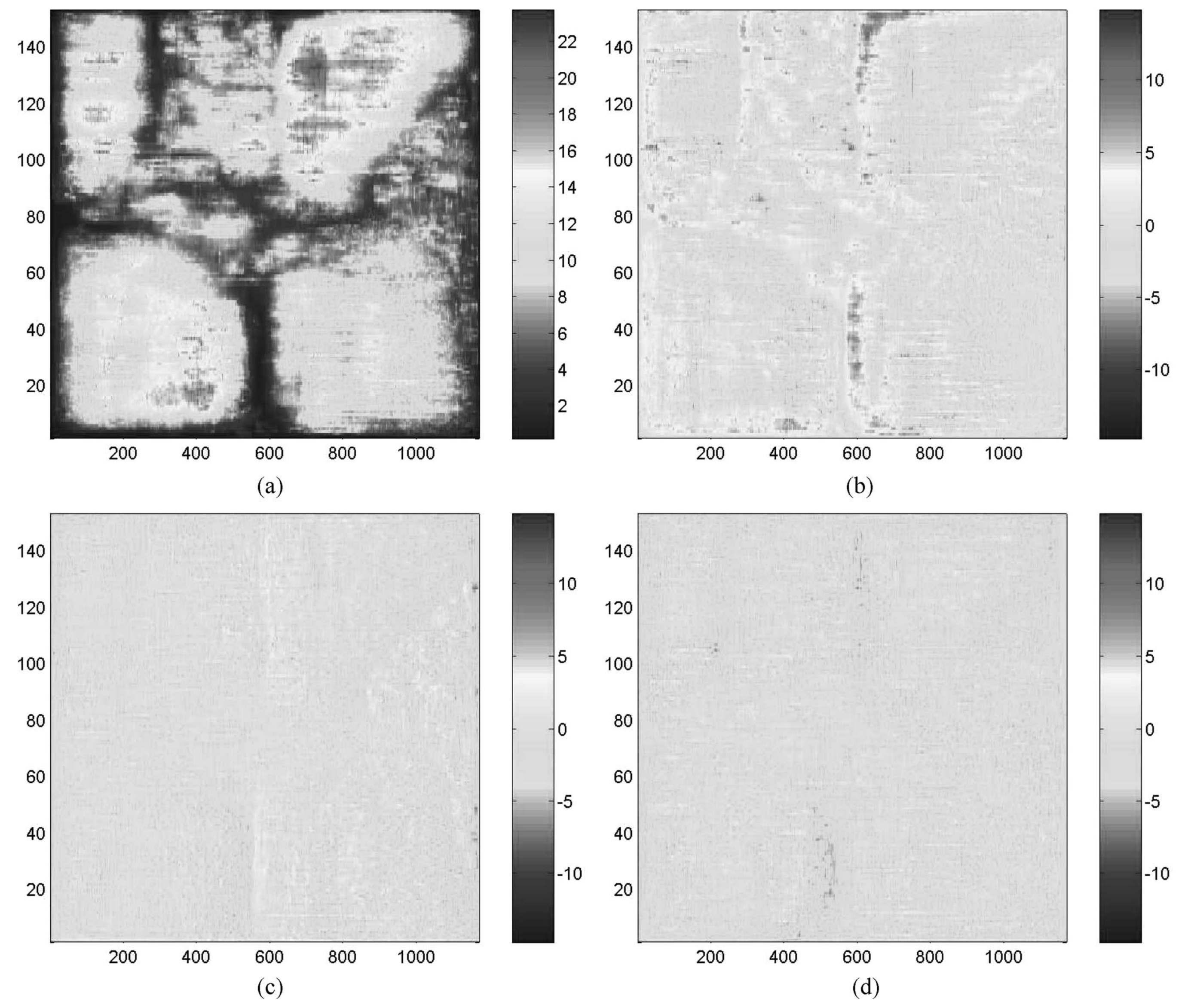

Fig. 8. (a) Congestion map of ibm02-easy benchmark after applying mPL-R. (b) Incremental changes in the congestion levels after applying the cut-line adjustment step of WSA. (c) Incremental changes in the congestion levels after legalization. (d) Incremental changes in congestion levels after the local minimization step.

TABLE I
CHARACTERISTICS OF IBM VERSION 2 BENCHMARKS

benchmarks	cells	nets	ws%	layers
ibm01-easy	12,208	11,753	14.88	4
ibm01-hard			12.00	4
ibm02-easy	19,062	18,688	9.58	5
ibm02-hard			4.72	5
ibm07-easy	44,811	44,681	10.05	5
ibm07-hard			4.70	5
ibm08-easy	50,672	48,230	9.97	5
ibm08-hard			4.84	5
ibm09-easy	51,382	50,678	9.76	5
ibm09-hard			4.88	5
ibm10-easy	66,762	64.971	9.78	5
ibm10-hard			4.92	5
ibm11-easy	68,046	67,422	9.89	5
ibm11-hard			4.67	5
ibm12-easy	68,735	68,376	14.78	5
ibm12-hard			9.94	5

run once as they generate the same placement for different runs. **Dragon**, **CAPO**, and **Feng Shui** are run on a Pentium 4 2.6-GHz CPU with 512-MB memory running Linux. **mPG**, **Cadence QPLACE**, and **WROUTE** are run on an UltraSpacII 450-MHz CPU with 1-GB memory.

The results are given in Tables II–V. We use "Dragon-fd" in these tables to emphasize **Dragon** in the congestion-driven mode instead of its default wirelength-driven mode. All data are obtained by averaging over the runs except the column "S/V/F." In Tables II, IV, and V, the columns "r-WL," "vias," "vlts," "o.c.%" and "r-time" show the routed wirelengths, numbers of vias, numbers of violations, percentages of overcapacity gcells and routing times reported by **WROUTE**, respectively.

The column "S/V/F" shows status of the routing results. The status of a routing result can be one of the following: successful routing without violation (denoted as "S"), finished routing with some violations (denoted as "V"), and failed routing due

TABLE II
ROUTABILITY COMPARISON OF OUR TOOL WITH VARIOUS PLACEMENT TOOLS. ALL WIRELENGTHS ARE SCALED BY 10^6 MICROMETER

bench-	tools	routed by Cadence WROUTE						bench-	tools	routed by Cadence WROUTE					
marks		S/V/F	r-WL	vias	vlts	o.c.%	r-time	marks		S/V/F	r-WL	vias	vlts	o.c.%	r-time
ibm01 -easy	Dragon-fd	1/0/4	0.929	146773	5346	5.64	1:26:47	ibm01 -hard	Dragon-fd	0/0/5	0.904	145770	6998	6.63	N/A
	CAPO	0/0/5	0.871	138132	10426	6.87	N/A		CAPO	0/0/5	0.870	140299	12740	8.09	N/A
	FS	0/0/1	0.847	139740	22833	9.83	N/A		FS	0/0/1	0.836	139103	21673	9.67	N/A
	mPG+ECO	0/0/5	1.003	147230	23225	10.40	N/A		mPG+ECO	0/0/5	1.001	148382	29628	11.95	N/A
	QPLACE	1/0/0	0.875	132250	0	3.21	0:28:01		QPLACE	0/1/0	0.831	133469	20	3.90	1:18:17
	APlace	1/0/0	0.80	152489	0	—	3:15:00		APlace	1/0/0	0.75	150947	0	—	4:13:00
	mPL-R+WSA	1/0/0	0.765	125321	0	1.54	0:21:46		mPL-R+WSA	1/0/0	0.735	126219	0	2.15	0:25:50
ibm02 -easy	Dragon-fd	5/0/0	2.18	313289	0	1.07	1:17:19	ibm02 -hard	Dragon-fd	2/2/1	2.24	321523	2045	2.59	2:48:18
	CAPO	0/0/5	2.33	316938	14414	5.21	N/A		CAPO	0/0/5	2.35	322585	21150	7.31	N/A
	FS	0/0/1	2.35	321215	24041	8.32	N/A		FS	0/0/1	2.33	316104	15686	5.10	N/A
	mPG+ECO	0/0/5	2.55	338704	31894	9.14	N/A		mPG+ECO	0/0/5	2.59	349984	58670	13.89	N/A
	QPLACE	1/0/0	2.11	289985	0	0.27	0:31:02		QPLACE	0/1/0	2.20	317981	1	1.59	1:49:20
	APlace	1/0/0	2.05	299306	0	—	0:48:00		APlace	1/0/0	2.14	315786	0	—	2:56:00
	mPL-R+WSA	1/0/0	1.88	283798	0	0.40	0:39:32		mPL-R+WSA	1/0/0	1.98	293535	0	0.99	0:46:53
ibm07 -easy	Dragon-fd	4/1/0	4.55	593637	4.4	0.68	1:46:48	ibm07 -hard	Dragon-fd	4/1/0	4.77	624005	0.8	2.51	3:34:02
	CAPO	1/4/0	4.74	607589	74.8	2.84	10:11:47		CAPO	1/1/3	4.84	626522	906.2	3.90	8:28:19
	FS	0/0/1	4.51	605145	2295	3.27	N/A		FS	0/0/1	4.71	603100	3356	4.72	N/A
	mPG+ECO	0/0/5	5.67	666646	83489	10.91	N/A		mPG+ECO	0/0/5	5.54	677474	113178	12.29	N/A
	QPLACE	0/1/0	4.67	561838	2	0.31	1:06:59		QPLACE	1/0/0	5.07	607972	0	2.17	2:43:00
	APlace	1/0/0	4.18	559354	0	—	1:35:00		APlace	0/1/0	4.29	586129	1	—	3:58:00
	mPL-R+WSA	1/0/0	4.26	544531	0	0.38	1:21:02		mPL-R+WSA	1/0/0	4.36	569806	0	1.55	2:15:13
ibm08 -easy	Dragon-fd	5/0/0	4.78	696404	0	0.08	1:01:55	ibm08 -hard	Dragon-fd	3/2/0	4.71	721215	0.4	0.28	2:01:41
	CAPO	1/4/0	4.93	711666	18	0.72	4:42:19		CAPO	0/5/0	5.04	751687	67.2	1.46	11:52:40
	FS	0/0/1	5.20	755160	1884	1.72	N/A		FS	0/1/0	4.90	745774	42	1.49	10:26:33
	mPG+ECO	0/1/4	5.56	784389	23700	4.28	13:17:23		mPG+ECO	0/0/5	5.50	788772	45339	5.95	N/A
	QPLACE	1/0/0	5.32	706212	0	0.37	1:27:59		QPLACE	1/0/0	5.18	713999	0	0.28	1:44:54
	APlace	1/0/0	4.58	681884	0	—	1:47:00		APlace	1/0/0	4.63	699441	0	—	2:45:00
	mPL-R+WSA	1/0/0	4.54	654051	0	0.05	0:53:17		mPL-R+WSA	1/0/0	4.45	678905	0	0.22	1:17:33
ibm09 -easy	Dragon-fd	5/0/0	3.81	594621	0	0.03	0:49:46	ibm09 -hard	Dragon-fd	5/0/0	3.70	603149	0	0.04	0:58:36
	CAPO	4/1/0	3.68	578888	0.2	0.05	1:05:14		CAPO	4/1/0	3.71	590540	0.2	0.08	1:07:53
	FS	1/0/0	3.63	588963	0	0.09	1:40:46		FS	1/0/0	3.53	580433	0	0.07	0:54:33
	mPG+ECO	5/0/0	4.00	616150	0	0.21	1:16:47		mPG+ECO	4/1/0	4.28	653824	2.2	0.86	2:21:01
	QPLACE	1/0/0	4.04	561859	0	0.01	0:47:27		QPLACE	1/0/0	3.90	578085	0	0.03	1:03:50
	APlace	—	—	—	—	—	—		APlace	—	—	—	—	—	—
	mPL-R+WSA	1/0/0	3.48	543582	0	0.02	0:47:16		mPL-R+WSA	1/0/0	3.61	563511	0	0.04	0:53:45
ibm10 -easy	Dragon-fd	3/2/0	7.46	934786	7.2	0.04	1:53:32	ibm10 -hard	Dragon-fd	5/0/0	7.16	939199	0	0.08	1:43:38
	CAPO	2/3/0	7.13	900907	9.6	0.10	2:47:00		CAPO	3/2/0	7.32	934833	7.6	0.30	4:22:06
	FS	1/0/0	6.86	914029	0	0.18	3:52:24		FS	0/1/0	6.93	928940	47	0.19	5:23:40
	mPG+ECO	3/1/1	7.91	990337	376	1.03	4:16:29		mPG+ECO	1/1/3	8.26	1051147	8752	2.27	6:46:32
	QPLACE	1/0/0	7.32	877598	0	0.02	1:13:10		QPLACE	1/0/0	7.47	916207	0	0.06	2:01:55
	APlace	—	—	—	—	—	—		APlace	—	—	—	—	—	—
	mPL-R+WSA	1/0/0	6.78	864527	0	0.02	1:19:13		mPL-R+WSA	1/0/0	6.69	889200	0	0.06	1:40:41
ibm11 -easy	Dragon-fd	2/3/0	5.68	780344	0.8	0.05	1:09:08	ibm11 -hard	Dragon-fd	4/1/0	5.57	795088	0.2	0.17	1:29:21
	CAPO	4/1/0	5.30	752747	0.2	0.13	1:40:51		CAPO	3/2/0	5.30	766920	0.6	0.25	1:55:29
	FS	1/0/0	5.65	768119	0	0.25	2:10:57		FS	0/1/0	5.69	778811	1	0.68	2:02:38
	mPG+ECO	3/2/0	5.93	813960	0.4	0.89	2:36:53		mPG+ECO	0/4/1	6.54	902662	6958	3.70	12:55:47
	QPLACE	1/0/0	6.05	745184	0	0.08	1:04:44		QPLACE	1/0/0	6.10	779713	0	0.22	1:50:30
	APlace	—	—	—	—	—	—		APlace	—	—	—	—	—	—
	mPL-R+WSA	1/0/0	5.10	706533	0	0.03	0:58:13		mPL-R+WSA	1/0/0	5.11	740125	0	0.20	1:26:44
ibm12 -easy	Dragon-fd	3/2/0	10.61	1141524	0.4	0.22	2:36:51	ibm12 -hard	Dragon-fd	1/4/0	10.50	1162650	0.8	0.48	3:44:46
	CAPO	1/4/0	10.27	1144195	23.4	0.87	11:42:32		CAPO	0/1/4	10.40	1176167	1085	1.92	20:07:32
	FS	0/0/1	10.05	1181418	658	2.15	N/A		FS	0/0/1	10.51	1199790	50288	5.66	N/A
	mPG+ECO	0/0/5	13.25	1397195	266038	14.63	N/A		mPG+ECO	0/0/5	12.95	1437522	357235	18.22	N/A
	QPLACE	0/1/0	11.98	1210849	122	2.40	9:44:26		QPLACE	1/0/0	11.09	1174369	0	1.46	4:48:27
	APlace	—	—	—	—	—	—		APlace	—	—	—	—	—	—
	mPL-R+WSA	1/0/0	10.43	1110546	0	0.37	4:46:02		mPL-R+WSA	1/0/0	10.03	1090540	0	0.33	3:04:42
summary	Dragon-fd	52/18/10	1.098×	1.088×	900	1.80×	1.58×	Except for "vlts", which is the average number of violations per run, results in the summary row are normalized with respect to mPL-R+WSA. †Routing results of APlace are from [28]. The routing is performed by SE 5.4 on a SUN Untra10 400MHz CPU.							
	CAPO	24/29/27	1.095×	1.079×	3808	5.52×	3.77×								
	FS	4/3/9	1.085×	1.085×	8925	8.92×	3.01×								
	mPG+ECO	16/10/54	1.257×	1.193×	66663	28.8×	6.76×								
	QPLACE	12/4/0	1.136×	1.052×	9.1	2.19×	1.42×								
	APlace†	7/1/0	1.031×	1.084×	0.13	—	3.85×								
	mPL-R+WSA	16/0/0	1.000	1.000	0	1.00	1.00								

to too many violations or too long a routing time (denoted as "F"). For example, "1/0/4" means that there are 1 successful routing, 0 finished routing, and 4 failed routings in five runs. The column "r-time" shows the average for only successful or finished routings. Failed routings are excluded in the routing time comparison, as they have either a very short routing time or a time of 24 h (the runtime limit for the router), which if included would skew the comparison in favor of this paper. The row "summary" summarizes the results of these placement tools on this suite of benchmarks. The column "S/V/F" in this row shows the total numbers of successful routings, finished routings, and failed routings on easy and hard benchmarks. The column "vlts" in this row shows the average numbers of violations of all easy and hard benchmarks.

Among these columns, "S/V/F" is a straightforward indicator of the routability of the placement generated by a placer. The routing completion rate is the ratio of the number of successful routings over the total number of routings, i.e., $\mathrm{S}/(\mathrm{S}+\mathrm{V}+\mathrm{F})$

TABLE III
RUNTIME COMPARISON AMONG VARIOUS PLACEMENT TOOLS. THE RUNTIMES ARE IN FORMATS OF "HR:MM:SS" OR "MM:SS"

benchmarks	Dragon-fd		CAPO		FS		mPG		QPLACE		m PL		mPL-R+WSA		
	p	p+r	p	p+r	p	p+r	p	p+r	p	p+r	p	p+r	p(mPL-R)	p(WSA)	p+r
ibm01easy	10:30	1:37:17	4:10	N/A	2:04	N/A	10:49	N/A	2:13	30:14	4:16	N/A	5:40	1:42	29:08
ibm01hard	10:23	N/A	4:06	N/A	2:08	N/A	11:04	N/A	2:45	1:21:02	4:28	N/A	5:45	1:33	33:08
ibm02easy	16:20	1:33:40	9:03	N/A	3:54	N/A	25:51	N/A	4:51	35:53	9:32	N/A	20:03	3:00	1:02:35
ibm02hard	16:16	3:04:34	8:44	N/A	3:56	N/A	22:06	N/A	6:05	1:55:25	9:55	N/A	18:43	2:34	1:08:10
ibm07easy	40:11	2:26:18	22:18	10:34:05	9:54	N/A	40:16	N/A	10:05	1:17:04	20:48	N/A	29:02	5:33	1:55:37
ibm07hard	26:08	4:00:10	21:14	8:49:33	9:24	N/A	48:18	N/A	12:28	2:55:28	21:52	N/A	27:54	4:45	2:47:52
ibm08easy	1:39:40	2:41:35	26:19	5:08:38	10:45	N/A	58:37	14:16:00	13:17	1:41:16	28:51	16:55:27	40:44	7:45	1:41:46
ibm08hard	1:04:54	3:06:35	25:34	12:18:14	11:22	10:37:55	1:04:44	N/A	17:07	2:02:01	30:24	N/A	40:19	6:35	2:04:27
ibm09easy	1:13:38	2:03:24	23:33	1:28:47	11:09	1:51:55	54:57	2:11:44	11:20	58:47	24:26	5:39:03	25:22	6:28	1:19:06
ibm09hard	49:57	1:48:33	22:43	1:30:36	10:31	1:05:04	52:42	3:13:43	14:34	1:18:24	24:45	7:24:03	25:31	5:33	1:24:49
ibm10easy	2:05:00	3:58:32	36:54	3:23:54	15:52	4:08:16	1:20:03	5:36:33	17:50	1:31:00	38:24	7:27:20	53:57	9:57	2:23:07
ibm10hard	1:21:34	3:05:12	36:27	4:58:33	15:16	5:38:56	1:23:55	8:10:27	23:05	2:25:00	39:42	15:00:26	50:59	8:23	2:40:03
ibm11easy	1:23:17	2:32:23	34:41	2:15:32	14:37	2:25:34	1:18:19	3:55:12	15:35	1:20:19	30:21	11:04:24	38:52	6:53	1:43:58
ibm11hard	54:12	2:23:33	33:22	2:28:51	14:46	2:17:24	1:10:39	14:06:27	19:02	2:09:32	32:29	11:13:43	37:38	6:37	2:10:59
ibm12easy	2:22:28	4:59:19	42:49	12:25:21	15:12	N/A	1:23:06	N/A	19:42	10:04:08	43:09	N/A	1:19:25	12:59	6:18:26
ibm12hard	1:32:41	5:17:27	34:54	20:42:26	16:34	N/A	1:28:52	N/A	20:17	5:08:44	44:53	N/A	1:16:16	11:01	4:31:59
summary	1.41	1.57	0.58	2.67	0.26	1.94	1.36	3.79	0.32	1.07	0.61	5.68	0.84	0.16	1.00

TABLE IV
IMPACTS OF VARIOUS ROUTABILITY OPTIMIZATION TECHNIQUES IN OUR FLOW. ALL WIRELENGTHS ARE SCALED BY 10^6 MICROMETER

bench-marks	tools	routed by Cadence WROUTE						bench-marks	tools	routed by Cadence WROUTE					
		S/V/F	r-WL	vias	vlts	o.c.%	r-time			S/V/F	r-WL	vias	vlts	o.c.%	r-time
ibm01 -easy	mPL	0/0/1	0.886	143828	16055	7.83	N/A	ibm01 -hard	mPL	0/0/1	0.884	143940	16124	8.33	N/A
	mPL-R	1/0/0	0.821	137894	0	3.22	1:07:36		mPL-R	1/0/0	0.789	137511	0	3.45	1:30:38
	mPL+WSA	1/0/0	0.808	130513	0	2.60	1:08:19		mPL+WSA	1/0/0	0.758	127165	0	2.11	0:40:09
	mPL-R+WSA	1/0/0	0.765	125321	0	1.54	0:21:46		mPL-R+WSA	1/0/0	0.735	126219	0	2.15	0:25:50
ibm02 -easy	mPL	0/0/1	2.26	318566	13405	4.66	N/A	ibm02 -hard	mPL	0/0/1	2.34	326682	24554	8.50	N/A
	mPL-R	1/0/0	1.96	297454	0	0.42	0:37:40		mPL-R	0/1/0	2.08	320952	45	2.17	4:31:47
	mPL+WSA	1/0/0	2.01	289135	0	0.87	0:48:36		mPL+WSA	1/0/0	2.03	292191	0	1.62	1:07:59
	mPL-R+WSA	1/0/0	1.88	283798	0	0.40	0:39:32		mPL-R+WSA	1/0/0	1.98	293535	0	0.99	0:46:53
ibm07 -easy	mPL	0/0/1	4.51	654818	1887	3.29	N/A	ibm07 -hard	mPL	0/0/1	5.10	642796	30539	6.85	N/A
	mPL-R	1/0/0	4.35	574541	0	0.34	1:15:28		mPL-R	1/0/0	4.52	607782	0	1.98	3:18:34
	mPL+WSA	1/0/0	4.04	525640	0	0.17	1:03:36		mPL+WSA	1/0/0	4.13	537549	0	0.33	1:07:16
	mPL-R+WSA	1/0/0	4.26	544531	0	0.38	1:21:02		mPL-R+WSA	1/0/0	4.36	569806	0	1.55	2:15:13
ibm08 -easy	mPL	0/1/0	5.13	806862	69	2.39	16:26:36	ibm08 -hard	mPL	0/0/1	5.44	791936	47979	6.08	N/A
	mPL-R	1/0/0	4.65	693206	0	0.07	0:57:52		mPL-R	1/0/0	4.61	717951	0	0.39	2:50:48
	mPL+WSA	1/0/0	4.70	666489	0	0.18	1:13:57		mPL+WSA	1/0/0	4.85	679489	0	0.49	1:42:13
	mPL-R+WSA	1/0/0	4.54	654051	0	0.05	0:53:17		mPL-R+WSA	1/0/0	4.45	678905	0	0.22	1:17:33
ibm09 -easy	mPL	0/1/0	3.95	645890	82	0.07	5:14:37	ibm09 -hard	mPL	0/1/0	4.00	673932	77	0.17	6:59:18
	mPL-R	1/0/0	3.57	580005	0	0.02	0:49:55		mPL-R	1/0/0	3.74	601755	0	0.04	0:57:17
	mPL+WSA	1/0/0	3.63	548431	0	0.02	0:54:33		mPL+WSA	1/0/0	3.60	550154	0	0.03	0:55:21
	mPL-R+WSA	1/0/0	3.47	543582	0	0.02	0:47:16		mPL-R+WSA	1/0/0	3.61	563511	0	0.04	0:53:45
ibm10 -easy	mPL	0/1/0	6.98	947526	42	0.17	6:48:56	ibm10 -hard	mPL	0/1/0	8.05	1071552	55	1.27	14:20:44
	mPL-R	1/0/0	6.94	910034	0	0.02	1:22:47		mPL-R	1/0/0	6.85	933290	0	0.06	1:53:46
	mPL+WSA	1/0/0	6.61	850293	0	0.02	1:33:25		mPL+WSA	0/1/0	7.27	881128	1	0.05	1:52:10
	mPL-R+WSA	1/0/0	6.78	864527	0	0.02	1:19:13		mPL-R+WSA	1/0/0	6.69	889200	0	0.06	1:40:41
ibm11 -easy	mPL	0/1/0	5.62	866097	132	0.40	10:34:03	ibm11 -hard	mPL	0/1/0	6.07	918744	91	1.39	10:41:14
	mPL-R	1/0/0	5.25	757252	0	0.03	0:59:50		mPL-R	0/1/0	5.28	786186	1	0.09	1:25:06
	mPL+WSA	1/0/0	5.11	705506	0	0.04	1:08:07		mPL+WSA	1/0/0	5.19	711791	0	0.04	1:13:17
	mPL-R+WSA	1/0/0	5.10	706533	0	0.03	0:58:13		mPL-R+WSA	1/0/0	5.11	740125	0	0.20	1:26:44
ibm12 -easy	mPL	0/0/1	11.16	1208277	2569	2.65	N/A	ibm12 -hard	mPL	0/0/1	11.61	1262711	64083	6.07	N/A
	mPL-R	1/0/0	10.48	1155284	0	0.41	4:21:03		mPL-R	1/0/0	10.27	1151207	0	0.67	4:27:59
	mPL+WSA	1/0/0	10.19	1070362	0	0.17	3:04:40		mPL+WSA	0/1/0	10.63	1125882	1	0.71	6:19:17
	mPL-R+WSA	1/0/0	10.43	1110546	0	0.37	4:46:02		mPL-R+WSA	1/0/0	10.03	1090540	0	0.33	3:04:42
summary	mPL	0/7/9	1.000	1.000	3564	1.00	1.00	Except for "vlts", which is the average number of violations per run, results in the summary row are normalized with respect to mPL.							
	mPL-R	14/2/0	0.906×	0.915×	2.9	0.17×	0.13×								
	mPL+WSA	14/2/0	0.894×	0.854×	0.13	0.13×	0.14×								
	mPL-R+WSA	16/0/0	0.875×	0.859×	0	0.13×	0.12×								

by using entries in the column "S/V/F." Overcapacity gcells ("o.c.%") reflects the congestion level of a placement, which is evaluated by the global router of **WROUTE**.

A. Routability Comparison

From Table II, we observe that our placement flow of **mPL-R+WSA** obtains successful routings for all 16 benchmarks, whereas other tools can only have partial successful routing results. For example, for five runs of **Dragon** and **CAPO** on each of the 16 benchmarks, **Dragon** obtains 65.0% (58 out of 80) successful, 22.5% (18) finished and 12.5% (10) failed routings,[5] and **CAPO** obtains 30.0% (24) successful, 36.3%

[5]One possible reason that routability of placement solutions obtained from Dragon reported in Table II differs from that in [44] is that the results from [44] were obtained using a different set of tuning parameters in the algorithm [43], which we cannot reproduce in this paper.

TABLE V
ROUTABILITY RESULTS AFTER APPLYING OUR APPROACH WSA TO PLACEMENTS GENERATED BY VARIOUS TOOLS. ALL WIRELENGTHS ARE SCALED BY 10^6 MICROMETER

bench-marks	tools	routed by Cadence WROUTE						bench-marks	tools	routed by Cadence WROUTE					
		S/V/F	r-WL	vias	vlts	o.c.%	r-time			S/V/F	r-WL	vias	vlts	o.c.%	r-time
ibm01 -easy	Dragon-fd+WSA	5/0/0	0.769	126654	0	2.09	0:46:53	ibm01 -hard	Dragon-fd+WSA	3/2/0	0.778	128765	0.6	3.05	1:22:11
	CAPO+WSA	4/1/0	0.769	126495	0.6	2.30	0:50:27		CAPO+WSA	3/2/0	0.773	128905	7.4	3.67	1:45:02
	FS+WSA	0/1/0	0.810	132076	12	2.86	1:52:17		FS+WSA	1/0/0	0.797	128534	0	4.17	1:55:35
	mPG+WSA	3/2/0	0.862	133267	19.8	3.55	2:12:51		mPG+WSA	0/1/4	0.850	132059	5587	5.24	3:38:16
	QPLACE+WSA	1/0/0	0.808	129264	0	2.14	0:36:09		QPLACE+WSA	0/1/0	0.775	129735	1	2.58	0:48:08
ibm02 -easy	Dragon-fd+WSA	4/1/0	2.06	291692	8.8	1.10	2:32:33	ibm02 -hard	Dragon-fd+WSA	4/1/0	2.11	299019	0.2	2.27	2:29:13
	CAPO+WSA	4/1/0	2.17	300109	0.4	1.48	1:35:50		CAPO+WSA	0/2/3	2.23	310135	8406	4.63	4:17:15
	FS+WSA	0/1/0	2.21	298786	1	2.17	1:49:48		FS+WSA	1/0/0	2.21	301630	0	2.47	2:33:34
	mPG+WSA	3/1/1	2.25	310687	2177	2.33	2:29:49		mPG+WSA	0/0/5	2.30	314014	15744	6.03	N/A
	QPLACE+WSA	1/0/0	2.04	286501	0	0.45	0:37:07		QPLACE+WSA	0/1/0	2.14	304311	2	1.94	1:23:55
ibm07 -easy	Dragon-fd+WSA	3/2/0	4.10	533518	17	0.26	1:41:03	ibm07 -hard	Dragon-fd+WSA	3/2/0	4.40	566598	6.8	1.72	3:00:05
	CAPO+WSA	5/0/0	4.23	529183	0	0.26	1:15:25		CAPO+WSA	3/2/0	4.42	561516	13.2	1.34	2:59:19
	FS+WSA	1/0/0	3.94	513394	0	0.15	1:00:13		FS+WSA	0/1/0	4.22	548636	1	1.44	3:37:06
	mPG+WSA	1/4/0	5.03	605921	19	2.87	5:44:20		mPG+WSA	0/1/4	4.99	627549	8273	6.26	20:07:24
	QPLACE+WSA	0/1/0	4.39	544466	1	0.33	1:25:37		QPLACE+WSA	1/0/0	4.84	582107	0	1.85	2:06:27
ibm08 -easy	Dragon-fd+WSA	4/1/0	4.42	637987	0.2	0.05	0:59:31	ibm08 -hard	Dragon-fd+WSA	3/2/0	4.39	664695	0.4	0.16	2:31:41
	CAPO+WSA	4/1/0	4.65	657066	4.2	0.17	1:25:24		CAPO+WSA	3/2/0	4.75	693723	1.8	0.43	3:29:01
	FS+WSA	0/1/0	4.82	651588	1	0.15	1:12:41		FS+WSA	0/1/0	4.70	658978	1	0.24	1:45:18
	mPG+WSA	4/1/0	4.98	695851	8.8	0.44	3:14:10		mPG+WSA	1/4/0	4.92	729977	21	1.21	7:28:37
	QPLACE+WSA	0/1/0	5.11	686148	2	0.34	1:54:12		QPLACE+WSA	1/0/0	5.00	691508	0	0.43	1:41:40
ibm09 -easy	Dragon-fd+WSA	4/1/0	3.46	539591	0.2	0.02	0:52:38	ibm09 -hard	Dragon-fd+WSA	4/1/0	3.39	547891	0.4	0.03	1:03:41
	CAPO+WSA	4/1/0	3.55	541794	0.2	0.03	0:54:40		CAPO+WSA	3/2/0	3.55	551237	0.4	0.03	1:01:54
	FS+WSA	1/0/0	3.58	535684	0	0.02	1:00:23		FS+WSA	0/1/0	3.43	539832	1	0.03	0:58:05
	mPG+WSA	3/2/0	3.69	558569	0.4	0.03	1:06:40		mPG+WSA	5/0/0	3.71	574078	0	0.09	1:06:18
	QPLACE+WSA	1/0/0	3.88	555051	0	0.03	0:51:55		QPLACE+WSA	0/1/0	3.74	563470	1	0.03	0:58:15
ibm10 -easy	Dragon-fd+WSA	3/2/0	6.82	853929	0.4	0.02	1:27:12	ibm10 -hard	Dragon-fd+WSA	4/1/0	6.58	861099	7.4	0.06	2:19:25
	CAPO+WSA	3/2/0	6.89	847951	0.6	0.02	1:35:21		CAPO+WSA	2/3/0	6.95	879383	1.4	0.12	3:00:45
	FS+WSA	1/0/0	6.74	832639	0	0.01	1:31:09		FS+WSA	0/1/0	6.70	854247	1	0.05	1:56:39
	mPG+WSA	4/1/0	7.04	879150	0.2	0.08	2:15:39		mPG+WSA	3/2/0	7.24	920737	14.4	0.34	3:57:46
	QPLACE+WSA	1/0/0	7.02	863250	0	0.04	1:47:26		QPLACE+WSA	0/1/0	7.16	894009	1	0.10	2:04:49
ibm11 -easy	Dragon-fd+WSA	5/0/0	5.15	702568	0	0.04	1:06:19	ibm11 -hard	Dragon-fd+WSA	4/1/0	5.08	718324	0.2	0.14	1:32:53
	CAPO+WSA	4/1/0	5.14	697548	0.2	0.03	1:08:07		CAPO+WSA	5/0/0	5.07	712433	0	0.11	1:27:35
	FS+WSA	1/0/0	5.46	693569	0	0.04	1:09:56		FS+WSA	1/0/0	5.38	710435	0	0.15	1:33:48
	mPG+WSA	5/0/0	5.33	722299	0	0.08	1:24:55		mPG+WSA	3/2/0	5.56	759126	0.6	0.40	2:17:06
	QPLACE+WSA	1/0/0	5.75	725456	0	0.08	1:13:28		QPLACE+WSA	1/0/0	5.83	758826	0	0.30	1:58:20
ibm12 -easy	Dragon-fd+WSA	3/2/0	9.71	1039890	0.6	0.11	2:18:01	ibm12 -hard	Dragon-fd+WSA	1/4/0	9.54	1059384	9.4	0.24	5:14:30
	CAPO+WSA	2/3/0	9.83	1056007	1.4	0.17	3:32:07		CAPO+WSA	0/5/0	9.85	1076948	11	0.49	5:34:32
	FS+WSA	1/0/0	9.82	1016333	0	0.08	2:11:07		FS+WSA	0/1/0	9.97	1075354	2	0.59	4:53:32
	mPG+WSA	0/1/4	12.32	1259799	40757	5.23	16:59:54		mPG+WSA	0/0/5	12.08	1287987	87444	9.30	N/A
	QPLACE+WSA	0/1/0	11.53	1134817	1	0.65	4:18:44		QPLACE+WSA	1/0/0	10.71	1132941	0	0.84	5:15:38
summary	Dragon-fd+WSA	57/23/0	0.910×	0.908×	3.3	0.64×	1.06×	Except for "vlts", which is the average number of violations per run, results in the summary row are normalized with respect to each original tool without applying WSA. See Table II for the detailed results of the original tools.							
	CAPO+WSA	49/28/3	0.924×	0.914×	932	0.33×	0.57×								
	FS+WSA	8/8/0	0.950×	0.897×	125	0.23×	0.44×								
	mPG+WSA	35/22/23	0.890×	0.896×	10004	0.26×	1.34×								
	QPLACE+WSA	9/7/0	0.955×	0.971×	0.6	1.22×	1.03×								

(29) finished and 33.8% (27) failed routings.[6] For one run on each of the 16 benchmarks, **QPLACE** obtains 75.0% (12 out of 16) successful and 25.0% (4) finished routings. Moreover, among all tools, our placement flow obtains the shortest routed wirelengths, whereas those of **Dragon** and **QPLACE** are 9.8% and 13.6% longer. **APlace** produces one finished routing among ibm01-08 benchmarks and 3.1% longer routed wirelength compared to our combined flow. We also obtain fewer vias and fewer overcapacity gcells. We conclude that our placement flow of **mPL-R+WSA** is the best in terms of routability and routed wirelength.

The runtimes of various placement tools including **mPL-R** and **WSA** on these benchmarks are shown in Table III in the columns with "p," "p(mPL-R)" and "p(WSA)." For each tool, we also show the total runtime of placement and routing (by **WROUTE**) in the columns "p + r." In the row "summary," we show the ratios of placement runtime and P&R runtime of these tools with respect to those of our flow on both easy and hard benchmarks, respectively. From Table II, we find that **Dragon**, **QPLACE**, and **mPL-R+WSA** achieve better routability on these IBM benchmarks. Among these three tools with good routability, **QPLACE** is the best in terms of placement runtime, thus is most scalable and **Dragon** the least scalable.

Within our combined flow, **mPL-R** contributes 84% of the total runtime, and **WSA** takes 16% of the total runtime. Compared to the wirelength-oriented **mPL**, our combined flow increases the total runtime by 1.64×. However, our runtime is still competitive to other tools especially when we consider the total placement and routing runtime on top of the quality of solutions obtained by our flow. When the total runtime of both placement and routing is considered, our flow is comparable to that of **QPLACE** and is much shorter than other placers, with the caveat that **QPLACE**, **mPG**, and **WROUTE** are run

[6]Possible reasons for the difference in the routability of placement solutions obtained from CAPO in Table II and in [39] are: 1) noting that CAPO runs faster than most placement tools, each run of CAPO in [39] produces three independent solutions, among which the one with the best wirelength is picked and 2) a newer version of WROUTE (in SOC 4.1) with the option "frouteAutoStop FALSE" being used [33].

on SUN 450-MHz CPU and others are run on Linux 2.6-GHz CPU. Recall that we exclude failed routings in the calculation of the total placement and routing time, which if considered, the total placement and routing time of our flow would have been even better than what is shown in this table.

B. Impacts of Various Routability Optimization Techniques

Table IV compares the impacts of each technique in our workflow. We compare the routability of flow **mPL**, **mPL-R**, **mPL+WSA**, and **mPL-R+WSA**. Note that we did not compare with recent placement packages **mPL5** [14] and **mPL6** [13], as they are wirelength optimization only. The results in the summary row of Table IV are normalized to those of **mPL**. It can be seen that both routability control technique during global placement stage and white space allocation technique after global placement stage are effective in relieving routing congestion. Compared to **mPL**, these two techniques can reduce the overflowed global routing cells by 83% and 87%, respectively. As a result, both techniques improve the completion rate from zero successful routing to 14 successful routings out of 16 routings. Both techniques also reduce the routed wirelength by 9.4% and 10.6%, respectively. The combined flow has successful routings on all these benchmarks. Routed wirelengths are improved significantly by 12.5%. Overall, the combined workflow is the best in terms of final completion rate and routed wirelength. We also conclude that both techniques are necessary in this congestion-driven placement flow.

C. Impacts of White Space Allocation

We further evaluate the impacts of the white space allocation technique by applying it on the placements generated by other tools. Routability results are shown in Table V.

Table V shows that our approach can greatly improve the routability of a placement, especially on those placements generated by wirelength-driven placement tools. After applying our approach, **Dragon** obtains 71.3% (57 out of 80) successful, 28.8% (23) finished and 0 failed routing results and **CAPO** obtains 61.3% (49) successful and 35.0% (28) finished and 3.8% (3) failed routing results; similar results are obtained for **Feng Shui**, **mPG**, and **QPLACE**. Compared to results obtained by these tools without our white space allocation technique (see the summary row in Table V), we consistently reduce the routed wirelength by 4.5% to 11.0%. We also reduce the number of vias, violations, and significantly reduce the overcapacity gcells (with the exception of **QPLACE**). We conclude that routability and routed wirelength can be simultaneously improved with our white space allocation technique. Although **Dragon_fd+WSA** has a similar average routed wirelength as **mPL-R+WSA**, its routability is much worse.

During the top-down cut line adjustment process, there are changes in the congestion levels of the placement in different regions. However, we observe that the changes of congestion levels are fairly local and minimal; consequently, it is not necessary to update the congestion map during this top-down flow or readjust cut line location based on a new congestion map. We show the changes in the congestion map during our combined flow **mPL-R+WSA** on a particular benchmark ibm02-easy in Fig. 8. We show in Fig. 8(a) the congestion map after applying **mPL-R**. Hot spots in Fig. 8(a) correspond to congested regions in the global placement generated by **mPL-R**. In Fig. 8(b), we show the changes in the congestion levels made by the first step of **WSA**, i.e., cut line adjustment. As indicated by the "hot spots" in Fig. 8(b), the neighboring regions of the true hot spots in Fig. 8(a) have significant increases in their congestion levels after cut-line adjustment. It is obvious that white space allocation by cut line adjustment migrates routing demands from congested regions to less congested neighboring areas. Fig. 8(c) and (d) shows the incremental changes in the congestion levels made by the legalization and local minimization steps of **WSA**. It is evident that the legalization and local minimization steps only slightly perturb the white space distribution and hence congestion distribution. We conclude that: 1) it is not necessary to update the congestion map during white space allocation step and 2) the legalization step and detailed placement step do not redistribute white space greatly.

VI. Conclusion and Future Work

Routability is one of the key problems in the placement stage. It can be achieved by either reducing routing demands or increasing routing resources in congestion regions. We propose a congestion-driven global placement that enhances the routability by considering routing resource demand reduction. We also propose a congestion-driven white space allocation step that can further allocate appropriate amounts of white space into congestion regions to increase routing resources without large wirelength overhead. Experimental results show that our placement flow can greatly improve routability and reduce routed wirelength. We achieve successful routings on all IBM easy and hard benchmark circuits with the best routed wirelength and competitive runtime. Compared to congestion-driven **Dragon**, which generates 65.0% successful routings out of 80 placements on IBM version 2 easy and hard benchmarks, our combined flow generates 100% successful routings with 9.8% shorter routed wirelength, much fewer vias, and shorter routing runtime. Compared to other academic tools, such as **CAPO**, **Feng Shui** and **mPG**, improvements obtained from the proposed flow are even more significant.

One of our ongoing efforts is to extend the combined flow of **mPL-R+WSA** for mixed-size placement. The latest wirelength-driven version of **mPL** (**mPL6** [13]) effectively handles macroblocks and standard cells simultaneously. However, **WSA** introduces overlaps among standard cells and macroblocks; the legalization step in **WSA** may greatly change the cell locations with the existence of blocks [2]. Recall also that **WSA** scales all cell locations in one region as white space is allocated. The presence of fixed blocks, as in many real-life designs, will affect the effectiveness of this scaling. Irregular (instead of straight horizontal or vertical) cut line adjustment, diffusion-based method [38], or computational geometry-based placement migration [32] may alleviate such difficulties. More sophisticated heuristics combining wirelength reduction and placement migration (by cut line adjustment or other methods) and legalization techniques should be considered to handle movable or fixed blocks. The complete support of mixed-size placement with routability control is beyond the scope of this paper, as the mixed-size placement by itself is a very difficult problem and under active research [13], [20], [27], [35], [36]. In an upcoming work, we develop an efficient algorithm for legalization of mixed-size placement [21] and develop a framework of multilevel legalization. We plan to use this framework to incorporate WSA and support routability control in mixed-size placement.

Acknowledgment

The authors would like to thank X. Yang from Synplicity, Inc., and Prof. I. Markov and J. A. Roy from University of Michigan for their comments on the routing results. Prof. I. Markov also provided us the binary of CAPO 9.3.

References

[1] S. Adya, I. Markov, and P. Villarrubia, "On whitespace and stability in mixed-size placement and physical synthesis," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 311–318.

[2] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov, "Unification of partitioning, placement and floorplanning," in *Proc. Int. Conf. Comput.-Aided Des.*, 2004, pp. 550–557.

[3] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, "Fractional cut: Improved recursive bisection placement," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 307–310.

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

[5] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia, "Effective free space management for cut-based placement via analytical constraint generation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 10, pp. 1343–1353, Oct. 2003.

[6] U. Brenner and A. Rohe, "An effective congestion driven placement framework," in *Proc. Int. Symp. Phys. Design*, 2002, pp. 6–11.

[7] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Relaxed partitioning balance constraints in top-down placement," in *Proc. IEEE ASIC Conf.*, 1998, pp. 229–232.

[8] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placements?" in *Proc. Des. Autom. Conf.*, 2000, pp. 477–482.

[9] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Optimal partitioners and end-case placers for standard-cell layout," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 11, pp. 1304–1313, Nov. 2000.

[10] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hierarchical whitespace allocation in top-down placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 11, pp. 716–724, Nov. 2003.

[11] T. Chan, J. Cong, T. Kong, and J. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 171–176.

[12] T. Chan, J. Cong, T. Kong, J. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proc. Int. Conf. Comput.-Aided Des.*, 2003, pp. 299–306.

[13] T. Chan, J. Cong, M. Romesis, J. R. Shinnerl, K. Sze, and M. Xie, "mPL6: A robust multilevel mixed-size placement engine," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 227–229.

[14] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 185–192.

[15] C.-C. Chang, J. Cong, Z. Pan, and X. Yuan, "Physical hierarchy generation with routing congestion control," in *Proc. Int. Symp. Phys. Des.*, 2002, pp. 36–41.

[16] C.-C. Chang, J. Cong, Z. Pan, and X. Yuan, "Multilevel global placement with congestion control," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 4, pp. 395–409, Apr. 2003.

[17] C.-L. E. Cheng, "RISA: Accurate and efficient placement routability modeling," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 1994, pp. 690–695.

[18] J. Cong, T. Kong, J. Shinnerl, M. Xie, and X. Yuan, "Large-scale circuit placement: Gap and promise," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2003, pp. 883–890.

[19] J. Cong, T. Kong, J. Shinnerl, M. Xie, and X. Yuan, "Large-scale circuit placement," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 10, no. 2, pp. 389–430, Apr. 2005.

[20] J. Cong, M. Romesis, and J. R. Shinnerl, "Robust mixed-size placement under tight white-space constraints," in *Proc. Int. Conf. Comput.-Aided Des.*, 2005, pp. 165–172.

[21] J. Cong and M. Xie, "A robust detailed placement for mixed-size IC designs," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2006, pp. 188–194.

[22] K. Doll, F. M. Johannes, and K. J. Antreich, "Iterative placement improvement by network flow methods," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 10, pp. 1189–1200, Oct. 1994.

[23] D. Hill, "Method and system for high speed detailed placement of cells within an integrated circuit design," U.S. Patent No. 6 370 673, Apr. 9, 2002.

[24] W. Hou, H. Yu, X. Hong, Y. Cai, W. Wu, J. Gu, and W. Kao, "A new congestion-driven placement algorithm based on cell inflation," in *Proc. Asia South Pacific Design Autom. Conf.*, 2001, pp. 605–608.

[25] B. Hu and M. Marek-Sadowska, "Congestion minimization during placement without estimation," in *Proc. Int. Conf. Comput.-Aided Des.*, 2002, pp. 739–745.

[26] S.-W. Hur and J. Lillis, "Mongrel: Hybrid techniques for standard cell placement," in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 165–170.

[27] A. B. Kahng, S. Reda, and Q. Wang, "Architecture and details of a high quality, large-scale analytical placer," in *Proc. Int. Conf. Comput.-Aided Des.*, 2005, pp. 891–898.

[28] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 18–25.

[29] A. Khatkhate, C. Li, A. Agnihotri, M. Yildiz, S. Ono, C.-K. Koh, and P. Madden, "Recursive bisection based mixed block placement," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 84–89.

[30] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," in *Proc. Int. Conf. Comput.-Aided Des.*, 2004, pp. 394–401.

[31] J. Lou, S. Krishnamoorthy, and H. Sheng, "Estimating routing congestion using probabilistic analysis," in *Proc. Int. Symp. Phys. Des.*, 2001, pp. 112–117.

[32] T. Luo, H. Ren, C. J. Alpert, and D. Z. Pan, "Computational geometry based placement migration," in *Proc. Int. Conf. Comput.-Aided Des.*, 2005, pp. 41–47.

[33] I. Markov and J. A. Roy, Nov. 2005. Private Communication.

[34] S. Mayrhofer and U. Lauther, "Congestion-driven placement using a new multi-partitioning heuristic," in *Proc. Int. Conf. Comput.-Aided Des.*, 1990, pp. 332–335.

[35] A. N. Ng, R. Aggarwal, V. Ramachandran, and I. L. Markov, "Solving hard instances of floorplacement," in *Proc. Int. Symp. Phys. Des.*, 2006, pp. 170–177.

[36] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *Proc. Int. Conf. Comput.-Aided Des.*, 2005, pp. 48–55.

[37] P. N. Parakh, R. B. Brown, and K. A. Sakallah, "Congestion driven quadratic placement," in *Proc. Des. Autom. Conf.*, 1998, pp. 275–278.

[38] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia, "Diffusion-based placement migration," in *Proc. Des. Autom. Conf.*, 2005, pp. 515–520.

[39] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov, "Capo: Robust and scalable open-source min-cut floorplacer," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 224–226.

[40] R.-S. Tsay, S. Chang, and J. Thorvaldson, "Early wirability checking and 2-D congestion-driven circuit placement," in *Proc. Int. Conf. ASIC*, 1992, pp. 50–53.

[41] J. Westra, C. Bartels, and P. Groeneveld, "Probabilistic congestion prediction," in *Proc. Int. Symp. Phys. Des.*, 2004, pp. 204–209.

[42] J. Westra and P. Groeneveld, "Is probabilistic congestion estimation worthwhile?" in *Proc. Int. Workshop Syst. Level Interconnect Prediction*, 2005, pp. 99–106.

[43] X. Yang, Nov. 2005. Private Communication.

[44] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Routability-driven white space allocation for fixed-die standard-cell placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 4, pp. 410–419, Apr. 2003.

[45] X. Yang, R. Kastner, and M. Sarrafzadeh, "Congestion reduction during placement based on integer programming," in *Proc. Int. Conf. Comput.-Aided Des.*, 2001, pp. 573–576.

[46] X. Yang, R. Kastner, and M. Sarrafzadeh, "Congestion estimation during top-down placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 21, no. 1, pp. 72–80, Jan. 2002.

Chen Li received the B.Eng. and M.Eng. degrees in electronic engineering and microelectronics, both from Tsinghua University, Beijing, China, in 1998 and 2001, respectively, and the Ph.D. degree from the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in 2006.

He is currently with Magma Design Automation, Inc., Santa Clara, CA. His interests include physical design of very large scale integration (VLSI)/computer-aided design (CAD), with an emphasis on wire length, congestion, timing, and power.

Min Xie received the B.E. degree from Tongji University, Shanghai, China, in 1997 and the M.S. degree from Tsinghua University, Beijing, China, in 2001. He is currently working toward the Ph.D. degree at University of California, Los Angeles, all in computer science.

His research interests include VLSI physical design, placement, and global routing.

Cheng-Kok Koh (S'92–M'98–SM'06) received the B.S. (with first-class honors) and M.S. degrees, both from National University of Singapore, Kent Ridge, Singapore, in 1992 and 1996, respectively, and the Ph.D. degree from University of California, Los Angeles (UCLA), in 1998, all in computer science.

Currently, he is an Associate Professor of electrical and computer engineering with Purdue University, West Lafayette, IN. His research interests include physical design of VLSI circuits and modeling and analysis of large-scale systems.

Dr. Koh received the Lim Soo Peng Book Prize for Best Computer Science Student from the National University of Singapore in 1990, and the Tan Kah Kee Foundation Postgraduate Scholarship in 1993 and 1994. He received the GTE Fellowship and the Chorafas Foundation Prize from the UCLA in 1995 and 1996, respectively. He received the Association for Computing Machinery (ACM) Special Interest Group on Design Automation (SIGDA) Meritorious Service Award in 1998, the Chicago Alumni Award from Purdue University in 1999, the National Science Foundation (NSF) CAREER Award in 2000, the ACM/SIGDA Distinguished Service Award in 2002, and the Semiconductor Research Corporation (SRC) Inventor Recognition Award in 2005.

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, both from University of Illinois at Urbana–Champaign, in 1987 and 1990, respectively, all in computer science.

Currently, he is a Professor and the Chairman of the Computer Science Department of University of California, Los Angeles (UCLA). He is also a Codirector of the VLSI CAD Laboratory. He served on the ACM SIGDA Advisory Board, the Board of Governors of the IEEE Circuits and Systems Society, and the Technical Advisory Board of a number of electronic design automation (EDA) and silicon IP companies, including Atrenta, eASIC, Get2Chip, Magma Design Automation, and Ultima Interconnect Technologies. He was the Founder and President of Aplus Design Technologies, Inc., until it was acquired by Magma Design Automation in 2003. Currently, he serves as the Chief Technologist Advisor at Magma. Additionally, he has been a Guest Professor at Peking University since 2000. His research interests include CAD of VLSI circuits and systems, design and synthesis of system-on-a-chip, programmable systems, novel computer architectures, nanosystems, and highly scalable algorithms. He has published over 230 research papers and led over 30 research projects supported by Defense Advanced Research Projects Agency, NSF, SRC, and a number of industrial sponsors in these areas.

Dr. Cong 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 at Urbana–Champaign in 1989. He received the NSF Young Investigator Award in 1993, the Northrop Outstanding Junior Faculty Research Award from UCLA in 1993, and the ACM/SIGDA Meritorious Service Award in 1998. He has received three best paper awards including the 1995 IEEE Transactions on CAD Best Paper Award, the 2005 International Symposium on Physical Design Best Paper Award, and the 2005 ACM Transaction on Design Automation of Electronic Systems Best Paper Award. He also received SRC Inventor Recognition Awards in 2000 and 2006, and the SRC Technical Excellence Award in 2000. He served on the technical program committees and executive committees of many conferences, such as Asia and South Pacific Design Automation Conference, Design Automation Conference (DAC), Field-Programmable Gate Arrays, International Conference on Computer Aided Design, International Symposium on Circuits and Systems, International Symposium on Physical Design (ISPD), and International Symposium on Low Power Electronics and Design (ISLPED), and several editorial boards, including the IEEE TRANSACTIONS ON VLSI SYSTEMS and the ACM Transactions on Design Automation of Electronic Systems.

Patrick H. Madden (S'95–M'98) received the B.S. and M.S. degrees, both from New Mexico Institute of Mining and Technology, Socorro, NM, and the Ph.D. degree from University of California, Los Angeles (UCLA).

He is currently an Associate Professor of computer science with SUNY Binghamton, Vestal, NY. His primary research interests are on large scale combinatorial optimization, with applications to VLSI physical design. He is also actively engaged in research on cryptography.

Dr. Madden is the Vice-Chair of ACM/SIGDA. He is also the Technical Program Chair for ISPD and an IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Associate Editor.