Physical Hierarchy Generation with Routing Congestion Control

Chin-Chih Chang*, Jason Cong*, Zhigang (David) Pan†, and Xin Yuan*

ABSTRACT

In this paper, we develop a multi-level physical hierarchy generation (mPG) algorithm integrated with fast incremental global routing for directly updating and optimizing congestion cost during placement. The fast global routing is achieved by 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 also developed for placing objects with significant size variations. Experimental results show that, compared to GORDIAN-L , the wire length driven mPG is $3-6.5$ times faster and generates slightly better wire length for test circuits larger than 100K cells. Moreover, the congestion driven mPG improves 50% wiring overflow with 5% larger bounding box wire length but $3-6\%$ shorter routing wire length measured by graph based A-tree.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*Design Aids*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Placement, routing, congestion, interconnect, physical hierarchy, deep sub-micron

1. 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], Cong proposed a three phase interconnect-centric design flow, including (1) interconnect planning, (2) interconnect synthesis, and (3) interconnect layout in order to emphasize interconnect planning and optimization throughout the entire design process.

	Total Wire Length (meter)						all	net
layers	M1	M2	M3	M4	M5	M6	layers	count
nets <0.5mm	0.04	1.36	2.08	1.41	0.17	0.00	4.1%	43.2%
nets [0.5-5mm]	0.22	4.51	11.81	20.57	12.22	6.83	45.7%	48.7%
nets >5mm	0.01	0.26	2.93	14.43	20.87	23.11	50.2%	8.1%
all nets	0.2%	5.0%	13.7%	29.6%	27.1%	24.4%	100%	100%

Table 1: Wire length distribution on all routing layers of the block level net list of a micro-processor design

The interconnect planning phase is particularly important because it provides early assessments on the 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.

An important step in interconnect planning is *physical hierarchy generation*. There are some recent studies on generating good physical hierarchy from the flattened function and logic hierarchy for performance optimization [2, 3]. However, they have little or no consideration of 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.

The fact that more routing layers are added in the VLSI design suggests that the global interconnect congestion problem is worsening. Table 1 shows the wire length distribution based on the block level net list of a leading high performance microprocessor design from Intel [4], which demonstrates the congestion problem in global interconnects. The data show that over 95% of the wires are from wires longer than $0.5mm$ and over 80% of the wires are on the top three layers. It shows that there is high resource competition among long global interconnects on those top layers (which provide faster connections). Since long wires are usually sized wider for better performance, the top layers are much more congested.

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

Several existing works consider the congestion during placement or floorplan. In [5, 6], it is shown that there is a mismatch between wire length and congestion objectives. In [7], a simple LZ-shape routing is incorporated into a simulated annealing based floorplanning engine to consider congestion. However, there may not be enough global interconnects seen by a floorplanner. In [8], the wiring demand of a net is modeled by a weighted bounding box length. The wiring demand estimation can be fast, though it may

*UCLA Computer Science Department, Los Angeles, CA 90095, Email: {cchang, cong, yuanxin}@cs.ucla.edu

†IBM T.J. Waston Research Center, Yorktown Heights, NY 10598, Email: dpan@watson.ibm.com

ISPD'02, April 7-10, 2002, San Diego, California, USA.

be inaccurate. In [9], pre-computed Steiner tree topologies on a few grid structures are used for wiring demand estimations. This approach is tailored for recursive partition placement and may not foresee congestion problems within each partition. In [10, 11, 12], 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 on reducing congestion. In [13], a post-processing 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 [14], it is shown that a post-processing technique is effective in minimizing congestion because routing congestion correlates with wire length in a global view. In [15], a post-processing with a new congestion model is proposed. The congestion is first estimated by the method in [8] and revised by expanding certain congestion regions (by solving an integer programming problem). This approach improves the accuracy of congestion but 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 multi-level placement engine to provide dynamic routing congestion guidance to the placement engine.

2. PROBLEM FORMULATION

The interconnects of a VLSI circuit are determined by (1) the locations and sizes of logic gates, flip-flops, and buffers; (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 an important problem of identifying and optimizing 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 to estimate net topologies. The finer the bin structure becomes, the more accurate the cell locations and net topologies. We also call our physical hierarchy generation process "coarse placement" because we only place cells in coarse locations (bin centers).

The inputs of the physical hierarchy generation consist of logic hierarchy, design specification, and technology. The logic 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.

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); (3) delay estimations, power estimations, etc.

In this paper, we will only focus on the wire length minimization and routing congestion minimization.

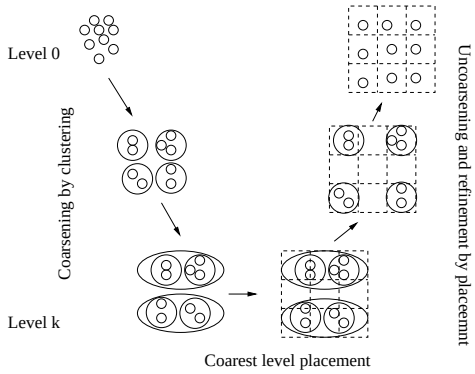

Figure 1: V-shape multi-level simulated annealing coarse placement framework

3. PHYSICAL HIERARCHY GENERATION

High computational complexity is the major challenge for physical hierarchy generation. Inspired by the recent success of the multi-level methods in efficiently handling high complexity designs in the VLSI CAD area [16, 17, 18], the backbone of our system is a multi-level simulated annealing (SA) engine.

3.1 Overview

Figure 1 shows an overview of our multi-level coarse placement framework. It includes a coarsening phase which recursively builds coarsening levels and a refinement phase which refines each coarser level representation to obtain a finer level representation. Our coarsening is done by iterative clustering. We select the *FirstChoice* (FC) clustering algorithm [19] because it experimentally gives us better clustering for coarse placement.

Our refinement is done by a simulated annealing (SA) based placement engine which places each cluster in the current level in a placement bin. We choose to use an SA based placement as in [20, 21] for the flexibility of integrating various design objectives and constraints. We use the *same* placement bin structure for each level.

For each refinement of a coarser level solution, the SA engine moves clusters of the current level (after declustering from the coarser level solution) to optimize bounding box wire length or routing congestion cost, both under area density constraints. The area density constraints are enforced by a hierarchical area density control algorithm. The routing congestion is evaluated by using a fast global router. The details of our algorithms shall be discussed in subsequent sections.

3.2 Hierarchical Area Density Control

Each placement bin has an area bound which is the area that can be used for cells placed in this bin. If the total area of the cells placed in a bin exceeds its area bound, some cells need to be moved to other locations. If the area bound in each placement bin is strictly enforced, a coarse placement solution can be legalized to an overlap-free detailed placement solution without moving any cell out of the placement bin assigned by the coarse placement.

It is difficult, however, to maintain 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 multi-

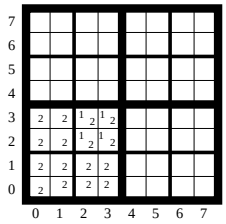

Figure 2: Bin hierarchy for area density control

level coarse placement due to the significant variances 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 looses the accuracy and creates cost jumps when switching to finer bin structures.

We solve this problem by a hierarchical area density control algorithm. Our density control imposes a hierarchy of bin structures on the target bin structure and enforces relaxed area constraints for all the bins in the hierarchy. Subsequently we shall show that the area constraints are gradually tighten in our multi-level framework while allowing more freedom for cluster moves.

The bin hierarchy is formed by recursively grouping adjacent bins to generate the bins in next level. Figure 2 shows an example of a bin hierarchy where boundary lines of different levels of the bin structure are drawn differently. In this figure, there are three bin structures: 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, which is also the summation of all area bounds of the level 0 bins contained in B_b^i. Similarly, denote U_b^i as the current area usage U_b^i for bin B_b^i, which is also the summation of all current area usages of the level 0 bins 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 a bin B_b^0, for any bin $B_{x_i}^i$ on level i that contains the bin B_b^0, the overflow of bin $B_{x_i}^i$ must be smaller than ka_c, where $k \geq 1$ is a user specified parameter ($\forall i (U_{x_i}^i + a_c - A_{x_i}^i) \leq ka_c$).

For example, if a cluster with area a_c is moved to bin $(2, 3)$ in Figure 2, 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 Figure 2, and the level 2 bin marked with 2 in Figure 2.

Using the hierarchical area density control, our placement engine can place clusters with mixed sizes. We will legalize macro cell locations after a few levels from the coarsest levels. As the refinement processes continue, the area constraints will be gradually tightened because the clusters become smaller. By using this method, our annealing engine can efficiently handle mixes of big and small modules 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 sub-regions of the region satisfies the area constraint. We apply this property in our move selection. If the SA engine generates a target location that moves a cell to a location that violates the hierarchical area constraints, we can efficiently find an alternative location. We can first find the smallest bin B in our hierarchy that contains the target location and all the higher level bins that contain this bin also satisfies the area constraint. An alternative location can be found by recursively applies the pigeon hole principle from this bin B.

3.3 Global Interconnect Topology Generation and Layer Assignment

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

The fast global router also includes a fast layer assignment algorithm which assigns nets according to net criticality. More critical nets have higher priority to choose better routing layers and routing topologies satisfying the performance constraints.

The layer assignment is not performed on each SA move in order to save the run time. In our implementation, we only perform layer assignment at the beginning of each SA phase that refines a placement solution from a coarser level to obtain a finer level placement solution.

3.3.1 Routing for Two-pin Nets

We use a fast two-bend routing algorithm to route two-pin nets, with at most two bends. We call the two-bend routing "LZ-routing" and our two-bend router an "LZ-router."

The possible number of configurations of LZ-routing that connects two pins with coordinates (i, j) and $(i + x, i + y)$ is $\mid x \mid + \mid y \mid$. However, with a simple minded implementation, finding an LZ-route requires to calculate $\mid x \mid \times \mid y \mid$ wire usage queries on bin boundaries, which is still quite expensive.

Our LZ-router uses auxiliary data structures (similar to a segment-tree data structure) to find good quality routes by performing a binary search of the possible routes for a two-pin net. For two pins bounded by a rectangle bounding box B, our LZ-router first measures congestion of B and its boundaries on both the horizontal and vertical layer to determine whether horizontal-vertical-horizontal (HVH) routing or vertical-horizontal-vertical (VHV) routing is less congested and should be used.

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

If the complexity for a region query 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 steps to find a route. Given a $g_x \times g_y$ bin structure, any region query used in our LZ-router can be (approximately) answered in $O(log(g_x + g_y))$ using segment-tree-like data structures. Therefore, the complexity for our LZ-routing is $O(log(\mid x \mid + \mid y \mid)log(g_x + g_y))$.

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 $O(log(\mid x \mid + \mid y \mid)log(g_x + g_y))$.*

Due to page limitations, the details of the auxiliary data structures, the complexity analysis, and the proof are omitted. They can be found in the technical report [22].

3.3.2 Incremental Hierarchical A-tree Construction

For a multi-pin net, we would like to construct a rectilinear Steiner arborescence tree (A-tree) for our routing estimation. A rectilinear Steiner arborescence tree (A-tree) is a shortest path rectilinear Steiner tree. There are some heuristics that can construct an n-pin A-tree in $O(n \log n)$ time with a solution no worse than 2x the optimal A-tree solution, e.g., [23, 24]. However, if we reconstruct each A-tree for any of the pin location updates, we may spend $O(n^2 \log n)$ for each n-pin net on a pass of moving all clusters, which is too expensive.

We propose an incremental A-tree (IncA-tree) algorithm that can be efficiently updated. We only explain the construction in the first

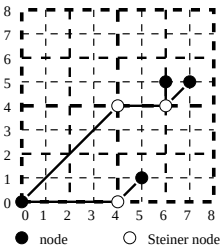

Figure 3: An example of IncA-tree

quadrant because the construction in 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 perform quad-partitioning on the grid structure until it reaches the unit grid.

The lower-left corner of the partition is the root for a subtree connecting all the pins inside this partition. By recursively performing such quad-partitioning, 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. Therefore, any pin insertion (deletion) to location (x, y) only incurs at the 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 the most $\log(x_1+y_1)+\log(x_2+y_2)$ edge changes. An example of an IncA-tree is shown in Figure 3.

With the fast two-pin routing and incremental A-tree routing, for an n-pin net with bounding box length L on a $g_x \times g_y$ bin structure, the complexity for updating a non-root 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 coarse placement, the run time overhead of our congestion cost updating grows slowly due to the low logarithmic complexity.

3.4 Multi-level Simulated Annealing Coarse Placement

The details of the SA engine is described below.

3.4.1 Solution Space:

The *same* bin structure is used for placement on each level. Clusters are placed at bin centers subject to hierarchical area constraints, which is explained in Section 3.2.

3.4.2 Cost Function:

The cost function for our SA engine has two modes: wire length driven and congestion driven. The cost function for the wire length driven mode is the simple summation of all the bounding box wire lengths of all nets.

The fast global router described in Section 3.3 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 routing layers of all bins. This cost is equivalent to the sum of weighted wire length by weighting all the wire segments in each bin by the wire usage of that bin.

This cost function encourages the SA moves that result in shorter wire length and less congestion. This cost function can also be efficiently updated if the wire usage is stored by a segment-tree-like data structure.

Because minimizing wire length is strongly related to congestion minimization, we will run our multi-level SA coarse placement with wire length minimization on coarser levels. We only turn on the congestion optimization at the last few finest levels of optimization. We have a "reduced mode" that only turns on the congestion driven at the finest level when the accepting ratio is lower than a predefined threshold t_r and alternatively runs the congestion driven and wire length driven modes with a fraction of congestion driven runs, denoted as f_c. Our experiments find that $t_r = 0.075$ and $f_c = 80\%$ gives best tradeoff of run time and solution quality.

3.4.3 Neighborhood Structure:

Two moves are used (1) cluster move; (2) I/O pads swap (not used in the experiments of this paper). A cluster move randomly selects a cluster and moves it to another bin. The target location is either randomly generated (within some range limit) or computed to optimize bounding box wire length. The experimental setting of random moves probability is $max(accept\ ratio, 0.6)$. If the generated move violates area constraints, an alternative target location is generated according to the method described in Section 3.2.

3.4.4 Cooling Schedule:

Let n_i be the number of clusters of level i. The cooling schedule is shown below.

- **starting temperature:** The starting temperature for the coarsest level (level k) is calculated by 20 times of the standard deviation of the costs of n_k random moves as suggested by [25]. For level i ($i < k$), it is calculated by binary searching to find the temperature with the expected cost-change of n_i moves close to zero [26].

- **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 $\alpha \leq 0.15$.

- **inner number:** We use two inner numbers $inner_0$ and $inner_1$. For each temperature on level i, we first start a pass with $inner_0 \times n_i$ 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. The values set experimentally are: $inner_0 = 1$, $inner_1 = 5$, and $m = 2$.

- **freezing temperature:** The freezing temperature is computed by $\lambda C/ec$, where C is the current cost; λ is a user input parameter; ec is the net count of the current level. The default value for λ is 0.005.

4. EXPERIMENTAL RESULTS

Our physical hierarchy generation algorithm is implemented in C++/STL. It can be run with three modes: wire length minimization (mPG), congestion cost driven at the finest level (mPG-cg), and the "reduced mode" described in Section *3.4.2* (mPG-cg.rd). Our experiments are conducted on a Sun Blade 1000 workstation running at 750MHz frequency (except the experiments done in Section 4.4).

We obtained our benchmark circuits from different sources: [17], [27], and industrial benchmarks from IBM. [1]

For the wire length comparisons, we use circuits in [17] such that we can compare to [17] and GORDIAN-L [29]. We do not use circuits from [27] because GORDIAN-L can not produce results for some circuits probably due to no connections to I/O pads caused by

[1] Please note that although both [17] and [27] have circuits derived from ISPD98 IBM benchmark suit [28], they are not the same. We rename the circuits in [17] by adding "-p" suffixes to indicate the differences. The circuits in [17] have the same net lists as in [28], however, all the cells have the same size. The circuits in [27] use the cell sizes specified in [28], however, all the big macro cells together with all the nets connecting to them are removed, thus most of the circuits do not have connections to I/O pads.

circuit	#cells	#nets	Gor+Dom		mPG+Dom	
			WL (10^6)	CPU (s)	WL (10^6)	CPU (s)
avqsmall	21854	22124	11.3	857	11.5 (1.02)	694 (0.81)
avqlarge	25114	25384	12.6	925	12.0 (0.95)	764 (0.83)
ibm04-p	27220	31970	6.86	1577	6.59 (0.96)	1397 (0.89)
ibm07-p	45639	48117	10.9	4385	10.3 (0.94)	3434 (0.78)
ibm09-p	53110	60902	11.8	6767	11.6 (0.98)	3364 (0.50)
ibm10-p	68685	75196	18.8	14133	18.9 (1.01)	5526 (0.39)
ibm14-p	147088	152772	40.8	39657	38.8 (0.95)	13131 (0.33)
ibm15-p	161187	186608	52.1	63876	51.7 (0.99)	16091 (0.25)
ibm16-p	182980	190048	55.0	81868	51.5 (0.94)	19979 (0.24)
ibm17-p	184752	189581	67.9	98440	66.2 (0.97)	22281 (0.23)
ibm18-p	210341	201917	53.7	129065	50.0 (0.93)	19944 (0.15)

Table 2: Wirelength comparison with GORDIAN-L

circuit	nets characteristics					#moves	eva. time (s)		speed
	#nets	3pin	4pin	5pin	6^+pin		IncA	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 3: Congestion evaluation time comparison. Two-pin nets are removed.

big modules removals. For the congestion driven experiments, we use circuits in [27]. We do not use circuits in [17] because all the cells have the same area is not reasonable for routing.

4.1 Wirelength Comparison with GORDIAN-L

We compared our wirelength-driven mPG with GORDIAN-L [29] followed by DOMINO [30] on two of the largest circuits (*avqsmall*, *avqlarge*) in 1993 MCNC layout benchmark sets and the ISPD98 IBM benchmark suit offered by the authors of [17] in Table 2. We ran GORDIAN-L followed by DOMINO and reported the BBOX wirelength of the detailed placement results and total runtime in columns titled "Gor+Dom." Also we ran mPG followed by DOMINO and reported the wirelength and total runtime in columns titled "mPG+Dom" and listed the ratio between the wirelength and runtime of mPG and that of GORDIAN-L in parentheses.

It shows that mPG provides a slightly shorter wirelength and significant less run time, especially in circuits larger than 100K.

4.2 Speed-up by Incremental A-tree

The incremental A-tree algorithm enables us to directly integrate a global router into the placement engine without suffering from an overly-long runtime.

In this section, we shall provide the run time comparison between IncA-tree and an implementation that completely routes a net by an A-tree algorithm [23] whenever a pin of this net is moved during the simulated annealing process.

We used some circuits from IBM-PLACE benchmarks suite[27] for this experiment. For each circuit, we first eliminated all the two-pin nets and only kept the multi-terminal nets for testing.[2] For a move generated by SA engine, we used the IncA-tree algorithm to incrementally evaluate the congestion and recorded the runtime for a pre-determined number of moves. The identical set of moves were also evaluated by the A-tree algorithm.

It can be seen in Table 3 that the IncA-tree algorithm can speed up the evaluation process by a factor of at least 5 and even more when the nets with a higher degree become dominant.

[2] We use suffixes "-r" in the circuit names to indicate the two-pin nets are removed from [27].

4.3 Congestion Control Comparison

In order to evaluate effects of the congestion optimization, we implemented a global router based on the GA-tree algorithm [24] 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. It works on a routing graph where nodes represent routing bins and edges correspond to the shared boundaries of adjacent bins.

We can obtain a global routing solution with congestion control by using a slope-based cost function for edge weight to penalize the overflowed/highly congested edges (similar to that used in [31]) and updating the weight after routing each net.

The benchmarks for testing congestion control are chosen from the IBM-PLACE benchmarks suite [27].[3] For each test case, we ran GORDIAN-L followed by DOMINO to get a wirelength-driven placement result and then used our GA-tree based global router to evaluate the congestion. Meanwhile we ran mPG to perform wirelength-driven placement and ran mPG-cg and mPG-cg.rd to perform a congestion-driven placement. All mPG runs used the GA-tree based global router to evaluate the results. For all the test cases, we used two routing layers for global routing evaluation.

In Table 4, we reported the congestion pictures in total overflow (tot. ov), the maximal boundary congestion (max. b.cg), the routing wirelength (routing WL) and total bounding box wirelength for GORDIAN-L/DOMINO (G/D), mPG, mPG-cg, and mPG-cg.rd.

It can be seen that although in terms of bounding box wirelength, wirelength-driven placers (mPG and GORDIAN-L) offer better results in general, their routed wirelength actually becomes larger than that generated by mPG-cg on average. This implies that the bounding box wirelength is no longer a good metric for routability. A similar conclusion was also drawn in [15]. Meanwhile, the mPG-cg reduces any existing total overflow by 53–79% on average and reduces either the routed wirelength or the maximal boundary congestion, demonstrating that the congestion control done 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 mPG-cg with the reduced mode demonstrate the tradeoff between runtime and solution-quality.

4.4 Experiments on Industrial Circuits

We also ran our program on 5 IBM test circuits (named ind1 to ind5, to avoid name confusion with the published IBM benchmark used in the previous section) on a Sun workstation running at 400MHz frequency, followed by IBM's in-house legalization and routing tools. These circuits use IBM ASIC standard cell libraries, with feature size from $0.15\mu m$ to $0.25\mu m$, and some circuits have a number of pre-placed macros (not counted in the cell number).

Table 5 shows the number of placeable cells (#cell), the number of nets (#nets), the grid size, the comparison of routed wirelength, maximum congestion, the number of overflowed edges and the number of nets that overflow.[4] It confirms that the placement results generated by mPG-cg have less congestion than that by mPG, with fewer congested edges and nets, though running much slower. The reduced mode mPG-cg.rd provides a tradeoff between run time and quality of result.

5. CONCLUSIONS

We presented a multi-level simulated annealing physical hierarchy generation algorithm (mPG-cg) integrated with incremental

[3] We removed the dangling cells in the circuits.

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

circuit name	#cells	#nets	grid size	BBOX WL				routing WL				max. b.cg				tot. ov				CPU (s)			
				G/D	mPG	mPG -cg.rd	mPG -cg	G/D	mPG	mPG -cg.rd	mPG -cg	G/D	mPG	mPG -cg.rd	mPG -cg	G/D	mPG	mPG -cg.rd	mPG -cg	G/D	mPG	mPG -cg.rd	mPG -cg
ibm01	12028	11507	32×64	4.95e6	4.37e6	4.66e6	4.77e6	6.19e6	5.37e6	5.26e6	5.14e6	1.36	1.27	1.10	1.06	2284	950	319	114	1363	578	3133	8295
ibm04	26332	26163	64×64	1.70e7	1.66e7	1.74e7	1.73e7	1.96e7	1.97e7	1.96e7	1.87e7	1.20	1.26	1.22	1.06	1949	1266	639	77	3173	1228	9709	27922
ibm07	44811	44394	64×64	4.25e7	3.18e7	3.31e7	3.33e7	6.47e7	4.57e7	4.48e7	4.33e7	2.40	1.78	1.78	1.69	2.97e5	7.51e4	6.39e4	4.74e4	65027	2410	14683	43618
ibm11	68046	67016	64×64	-	4.19e7	4.37e7	4.36e7	-	4.85e7	4.78e7	4.59e7	-	1.25	1.10	1.02	-	3014	811	271	-	3440	17923	72010
ibm13	83806	80906	64×128	-	5.14e7	5.36e7	5.41e7	-	7.31e7	6.82e7	6.40e7	-	1.51	1.39	1.31	-	7.43e4	3.62e4	2.10e4	-	4393	27312	76768
ibm15	161196	157806	128×128	-	1.47e8	1.52e8	1.52e8	-	1.82e8	1.77e8	1.67e8	-	1.14	1.10	1.07	-	3971	1596	288	-	13285	74458	263706
avg.				-	1	1.05	1.05	-	1	0.97	0.94	-	1	0.93	0.87	-	1	0.47	0.21	-	1	6.1	18.9

Table 4: Congestion control comparison between wirelength-driven placement and mPG-cg.

circuit	#cell	#net	grid size	routed WL			max. cg			#edge-ov			#nets-ov			cpu(s)		
				mPG	mPG -cg.rd	mPG -cg	mPG	mPG -cg.rd	mPG -cg	mPG	mPG -cg.rd	mPG -cg	mPG	mPG -cg.rd	mPG -cg	mPG	mPG -cg.rd	mPG -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

Table 5: IBM circuit results.

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 mixed big and small clusters. Our experiments show that our mPG program is both competitive in wire length and run time. The congestion driven mPG (mPG-cg) can significantly reduce routing congestion.

Acknowledgment

This work is supported in part by SRC, an IBM Faculty Partnership Award, a grant from Intel and a grant from Fujitsu under the California MICRO program. The authors would like to thank K. Konigsfeld, M. Mohan, G. Karypis, G. Chen and M. Romesis for their helpful discussion.

6. REFERENCES

[1] J. Cong, "An interconnect-centric design flow for nanometer technologies," *Proceedings of the IEEE*, vol. 89, pp. 505–527, April 2001.

[2] J. Cong and S. K. Lim, "Physical planning with retiming," in *IEEE/ACM International Conference on Computer Aided Design*, pp. 2–7, Nov. 2000.

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

[4] Private communication with Kris Konigsfeld and Mosur Mohan.

[5] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placement?," in *Proceedings 2000 Design Automation Conference*, pp. 477–482, June 2000.

[6] A. B. Kahng, S. Mantik, and D. Stroobandt, "Requirements for models of achievable routing," in *Proceedings International Symposium on Physical Design*, pp. 4–11, April 2000.

[7] H.-M. Chen, H. Zhou, F. Young, D. Wong, H. Yang, and N. Sherwani, "Integrated floorplanning and interconnect planning," in *1999 IEEE/ACM International Conference on Computer-Aided Design*, pp. 354–357, 1999.

[8] C.-L. E. Cheng, "RISA: accurate and efficient placement routability modeling," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 690–695, Nov. 1994.

[9] S. Mayrhofer and U. Lauther, "Congestion-driven placement using a new multi-partitioning heuristic," in *International Conference on Computer-Aided Design*, pp. 332–335, 1990.

[10] 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 *Asia South Pacific Design Automation Conference*, pp. 605–608, 2001.

[11] 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 *Proceedings of the IEEE 1997 Custom Integrated Circuits Conference*, pp. 573–576, 1997.

[12] P. N. Parakh, R. B. Brown, and K. A. Sakallah, "Congestion driven quadratic placement," in *Proceedings 1998 Design Automation Conference*, pp. 275–278, 1998.

[13] R.-S. Tsay, S. Chang, and J. Thorvaldson, "Early wirability checking and 2-D congestion-driven circuit placement," in *International Conference on ASIC*, pp. 50–53, 1992.

[14] M. Wang, X. Yang, and M. Sarrafzadeh, "Congestion minimization during placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 1140–1148, Oct. 2000.

[15] X. Yang, R. Kastner, and M. Sarrafzadeh, "Congestion reduction during placement based on integer programming," in *IEEE/ACM International Conference on Computer-Aided Design*, pp. 573–576, 2001.

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, pp. 69–79, March 1999.

[17] T. F. Chan, J. Cong, T. Kong, and J. R. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proc. IEEE International Conference on Computer Aided Design*, pp. 171–176, November 2000.

[18] J. Cong, J. Fang, and Y. Zhang, "Multilevel approach to full-chip gridless routing," in *Proc. IEEE International Conference on Computer Aided Design.*, pp. 396–403, 2001.

[19] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings 1998 Design Automation Conference*, pp. 343–348, 1998.

[20] C. Sechen and A. Sangiovanni-Vincentelli, "The timberwolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. SC-20, pp. 510–522, April 1985.

[21] W.-J. Sun and C. Sechen, "Efficient and effective placement for very large circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 349–59, March 1995.

[22] C.-C. Chang, J. Cong, Z. D. Pan, and X. Yuan, "Physical hierarchy generation with routing congestion control," Tech. Rep. UCLA-CSD-020010, UCLA Computer Science Department, 2002.

[23] S. Rao, P. Sadayappan, F. Hwang, and P. Shor, "The rectilinear Steiner arborescence problem," *Algorithmica*, vol. 7, no. 2-3, pp. 277–288, 1992.

[24] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 1, pp. 24–39, 1999.

[25] M. Huang, F. Romero, and A. Sangiovanii-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *IEEE International Conference on Computer-Aided Design*, pp. 381–384, 1986.

[26] J. Rose, W. Klebsch, and J. Wolf, "Temperature measurement of simulated annealing placements.," in *IEEE International Conference on Computer-Aided Design*, pp. 514–517, 1988.

[27] `http://www.cs.ucla.edu/~xjyang/Dragon/ibm-place.html`.

[28] `http://nexus6.cs.ucla.edu/~cheese/ispd98.html`.

[29] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, March 1991.

[30] K. Doll, F. Johannes, and G. Sigl, "DOMINO: deterministic placement improvement with hill-climbing capabilities.," *IFIP Transactions A (Computer Science and Technology)*, vol. A-1, pp. 91–100, 1991.

[31] J. Cong and P. H. Madden, "Performance driven multi-layer general area routing for PCB/MCM designs," in *Proceedings 1998 Design and Automation Conference. 35th DAC*, pp. 356–361, 1998.