FAST HIERARCHICAL FLOORPLANNING WITH CONGESTION AND TIMING CONTROL

A. Ranjan, K. Bazargan and M. Sarrafzadeh

(abhi,kiarash,majid)@ece.nwu.edu
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208-3118

ABSTRACT

We propose fresher looks into already existing hierarchical partitioning based floorplan design methods and their relevance in providing faster alternatives to conventional approaches. We modify the existing partitioning based floorplanner to handle congestion and timing. We also explore the applicability of traditional Sizing Theorem for combining two modules based on their sizes and interconnecting wirelength. The results show that our floorplanning approach can produce floorplans hundred times faster and at the same time achieving better quality (on average 20% better wirelength, better congestion and better timing optimization) than that of pure Simulated Annealing based floorplanner.

1. INTRODUCTION

Floorplanning of the chip modules has been an extensively studied problem. Several strategies have been proposed for floorplan optimization. The hierarchical approach to floorplanning has been reported in [17][20][11][14]. In particular, the partitioning based hierarchical floorplanning has been studied in [5][9][10]. Floorplanning algorithms for general rectilinear blocks have been reported in [7][8]. However, till date the best floorplanning algorithm has been the one based on Simulated Annealing [19][18]. Simulated Annealing has proven to be very effective for area-optimization because others (like hierarchical) have difficulty matching the shapes.

Floorplanning being the first stage of VLSI Physical Design is the most suited for early optimization of timing, congestion and routability. Some recent researches have been directed towards doing interconnect planning [2] [4] and timing optimization [15] at the floorplanning stage. However, the options of speeding up the floorplanning process under these new constriants have never been fully explored.

Here, we discuss faster methods for *constrained floorplanning* with congestion and timing control. Constrained floorplanning is defined as floorplanning with fixed boundary shapes. Major work towards constrained floorplanning has been reported in [6][3][1]. We extend the definition of *constrained floorplanning* to constraints set on timing and congestion. In [12] a fast hierarchical floorplanning technique based on flexibility of modules has been presented In this work we propose modifications and enhancements to the Partitioning Based Floorplanner reported in [12] and compare the results with traditional Simulated Annealing. Our results show that , with suitable modifications, Partitioning Based Floorplanner performs as good as or better than the Simulated Annealing based approach in terms of timing and congestion optimization and at the same time achieving a speed-up of almost hundred.

The rest of the paper is organized as follows. In the next section, Section 2, we define our Constrained Floorplanning problem more precisely. In Section 3, we discuss how do we model different constraints and the associated properties. In Section 4, we show our modification of Hierarchical floorplanning approach to handle constrained floorplanning. The experimental setup and results are discussed in detail in Section 5 and Section 6 concludes our work by presenting further directions for research in this area.

2. PROBLEM DEFINITION

The input to the floorplanning algorithm is a circuit $C = (M, N)$ represented by a hypergraph, where M is the set of modules in the floorplanning system and N is the set of nets defining the connectivity among these modules. For the sake of simplicity we are assuming that the nets are connected to the center of the modules (this doesn't hamper a fair comparison between the two approaches). Each module m_i from the set of floorplanning modules $M = m_1, m_2, ...$ can be

- A rigid module: rectangular with fixed shape.
- A fully flexible module: can change its rectangular shape within a specified aspect-ratio range.
- A partially flexible module: can choose its rectangular shape from certain pre-defined fixed shapes.

There are some user defined constraints on the following objectives of the floorplan

- Shape constraint: Given a pre-specified maxWidth and maxHeight of the chip which define the shape and size of the final floorplan.
- Capacity constraint: The chip is divided into several bins by super-imposing a grid. Each bin boundary has a capacity (maximum number of nets that can cross it) associated with it. The objective is to minimize the capacity violation on these bins.
- Timing constraints: Based on the given clock speed the goal is to meet the critical delay for the longest paths (between two flip-flop boundaries). This delay may be modeled as the sum of the delays of the nets and gates on the critical path.

Given the input specification, the objective is to find a floorplan which best meets the given constraints.

3. MODELING THE OBJECTIVE FUNCTIONS AND CONSTRAINTS

3.1. Shape Constraints

For the constrained floorplanning the chip shape and size are pre-specified. Any violation from these boundaries in either direction is considered illegal. The shape cost is modeled as

$$ShapeCost = \mid w - W \mid + \mid h - H \mid; \qquad (1)$$

where W and H are the user-specified width and height for the final floorplan and w, h are width and height for the configuration under consideration. It is easy to see that the ideal final floorplan should have ShapeCost as zero.

3.2. Wirelength model

It has been fairly well researched that minimization of total wirelength helps towards other interconnect planning objectives like congestion and timing [16]. Our model for wirelength estimation is based on *center- to-center distance* between modules. In [11][14] Otten and Stockmeyer have presented a linear time algorithm for sizing sliceable floorplans. Their algorithm has been developed primarily for area-minimization during sizing. In our next lemma we prove that the set of non-redundant area implementations of combined module also contain non-redundant center-to-center length implementations.

Lemma : For any cut (vertical or horizontal) , in a sliceable floorplan, the set of non-redundant implementations of the combination contains all the minimum center-to-center wirelength implementations.

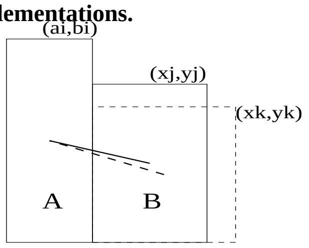

Figure 1. A non-redundant combination contains better wirelength

Proof: We consider vertical sizing (horizontal sizing follows a symmetric argument) as shown in Figure 1. Suppose the possible implementations of the modules A and B are (a_1, b_1), ..., (a_s, b_s) and (x_1, y_1), ..., (x_t, y_t), respectively. We can assume that the sets are sorted such that $a_i < a_{i+1}$ and $b_i > b_{i+1}$ for $i = 1, ..., s$ and similarly for x_js and y_js. Let us examine the implementations of the combination of A and B. Let (a_i, b_i) and (x_j, y_j) be the pairs selected to implement A and B respectively. The dimensions of the combination are given by $(a_i + x_j, max(b_i, y_j))$. Consider the case where b_i is the dominant height, that is, $max(b_i, y_j) = b_i$. Consider all pairs (x_k, y_k) with $k > j$. It is clear that all choices of (a_i, b_i) and (x_k, y_k) are inferior because (a_i, b_i) and (x_k, y_k) will not decrease the height but will increase the width. Thus, in the case when b_i is the dominant height of(a_i, b_i) and (x_j, y_j), all the implementations of (a_i, b_i) and (x_k, y_k) can be ignored for $k > j$. This way we have generated the non-redundant width and height implementations of the combination.

Thus far we have not included wirelength of the combination of the pair (a_i, b_i) and (x_j, y_j) when choosing non-redundant implementations. It may so happen that a redundant width and height implementation, pair (a_i, b_i) and (x_k, y_k) with $k > j$, has better center-to-center wirelength than the non-redundant pair (a_i, b_i) and (x_j, y_j). In the following discussion we will prove that such a thing can not happen. That is, the combination of (a_i, b_i) and (x_j, y_j) has better center-to-center wirelength than the redundant pair (a_i, b_i) and (x_k, y_k). For our wirelength model (center-to-center Euclidean distance) the distance between the centers of the non-redundant pair (a_i, b_i) and (x_j, y_j) is $(\frac{b_i - y_j}{2})^2 + (\frac{a_i + x_j}{2})^2$ and that between the redundant pair (a_i, b_i) and (x_k, y_k) is $(\frac{b_i - y_k}{2})^2 + (\frac{a_i + x_k}{2})^2$. Since $y_k < y_j$ and $x_k > x_j$ the center-to-center wirelength of the redundant combination is greater than that of the non-redundant combination. However, it is still possible that the redundant combination of (a_i, b_i) and (x_k, y_k) has better wirelength than some other non-redundant combination, say (a_l, b_l) and (x_m, y_m). Our claim is that even in such cases we can safely discard the combination of (a_i, b_i) and (x_k, y_k). The argument is that even if the redundant pair (a_i, b_i) and (x_k, y_k) has better wirelength than the non-redundant pair (a_l, b_l) and (x_m, y_m), we have a non-redundant combination (a_i, b_i) and (x_j, y_j) which has better wirelength and width/height than the redundant pair (a_i, b_i) and (x_k, y_k). Hence we can discard the pair (a_i, b_i) and (x_k, y_k) from our list of implementations of the combination of modules A and B. $\square$

Hence we need to retain only non-redundant area implementations as given by the sizing theorem. Such a list of non-redundant implementations contains all the minimum wirelengths for the combination. The proof can easily be extended to Bounding Box based wire models.

Note that this proof is valid only when we are combining two modules. That is, this wirelength minimization scheme by retaining non-redundant implementations is at best a local one. For global wirelength minimization we may still need to retain all the s x t implementations of the modules A and B. However, it is easy to see that such a choice will result in exponential growth in the number of implementations of the final floorplan. For our work we are retaining the non-redundant implementations at all levels of sizing. That is, even **when combining two sub-floorplans (each of these may itself be a combination of several modules) we retain only the non-redundant implementations**.

3.3. Congestion Model

We super-impose a grid on the floorplan under consideration and measure congestion in terms of capacity violation on the bin boundaries.

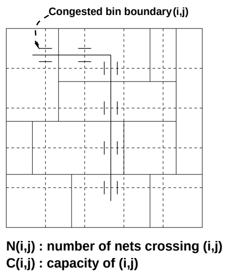
Congested bin boundary(i,j)
N(i,j) : number of nets crossing (i,j)
C(i,j) : capacity of (i,j)

Figure 2. Our congestion model

The model for our congestion estimation is shown in Figure 2. With each bin boundary (i, j) are associated the number of nets crossing it $N(i, j)$ and the capacity of the boundary $C(i, j)$. The bin boundaries marked as congested are violating the capacity constraint. To calculate the congestion (capacity violation), we do a simple bounding-box based routing to estimate the number of nets crossing a particular bin boundary and the overall congestion is the summation of all such capacity violations(overflows). Overflow on the bin boundary (i, j) is

$N(i, j) - C(i, j)$ if $N(i, j) > C(i, j)$ and zero otherwise.

The total congestion cost is:

$$CongestionCost = \sum_{i,j} [N(i, j) - C(i, j)] \tag{2}$$

3.4. Timing model

We model path delay as the sum of net delays and gate delays on this path. To calculate the net delays we use the timing model proposed in [13]: For the given coordinates of the terminals of the net (centers of modules in our case), the star point is calculated as the center of gravity of all the terminals on the net. The star point is connected to the driver terminal and all the driven terminals of the net. For a general module distribution the star model is shown in Figure 3.

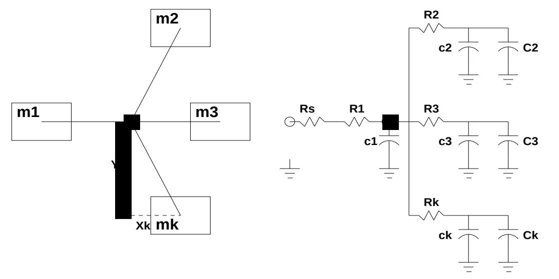

Figure 3. The star model and equivalent RC network

The number of terminals on the net is denoted by k, m_1 is the driver terminal and rest of the m_is are the driven terminals. Let X_i and Y_i denote the Manhattan distance of terminal i from the star node in x and y directions respectively. Based on this net model, the equivalent RC-model is calculated. In this model each edge has an output resistance R_s, a series resistance R_i and a parallel capacitance c_i and each driven terminal has a load capacitance of C_i. Based on the Manhattan distance from the star node, these lumped resistances and capacitances are

$$R_i = r_x.X_i + r_y.Y_i; \quad c_i = c_x.X_i + c_y.Y_i. \tag{3}$$

where r_x, r_y, c_x and c_y are the resistances and capacitances (per unit length) in x and y directions respectively.

The Elmore path delay from driver terminal d to the driven terminal i will be estimated as

$$(R_s + R_d)[c_d + \sum_{j \neq d}(c_j + C_j)] + R_i(c_i + C_i) \tag{4}$$

Though we are primarily relying on the star-model for modeling our delays, **our algorithm can work well with a variety of other timing models**. The timing cost for each floorplan configuration is then calculated as

$$TimingCost = \sum_{i=1}^{k} max(d[i] - D[i], 0); \tag{5}$$

where $d[i]$ is the Elmore delay of the ith critical path based on the Star model discussed earlier and $D[i]$ is the maximum allowed delay for that path.

4. OUR APPROACH

4.1. Modifying Traditional Hierarchical Floorplanner

Our Hierarchical Floorplanner is based on top-down partitioning and full implementation details are reported in [12]. In that work they handle area and wirelength as part of the optimization objectives. Authors have reported various heuristics to make partitioning based floorplanning effective. We will present the heuristics for partitioning based floorplanner as reported in [12] and our enhancements to these to handle congestion and timing:

4.1.1. Balance with respect to flexibility and area

The modeling of flexibility can be found in [12]. **Flexibility** is a measure of how flexible a module is, in terms of shape and number of representations. Flexibility in the representation of modules comes from the fact that they are at gate level and can be implemented in many shapes. While doing top-down partitioning the partitions are balanced with respect to area and flexibility. This helps in shape management by pairing rigid modules with flexible ones. The partitioner we use is hMetis which has been shown to be the best partitioner till date for VLSI circuits.

4.1.2. Combine modules to produce an aspect-ratio closer to one

Deciding on the cut (horizontal/vertical) for combining two modules is a hard problem. We propose to retain the cut which gives aspect-ratio of the combination closer to one. One motivation for doing that is to save on wirelength too. For that we also define the *equivalent aspect-ratio* of a flexible module which is simply the product of aspect-ratios of all its representations. For a fully flexible module the equivalent aspect-ratio is one.

4.1.3. Swapping sub-floorplans to improve wirelength

The min-cut partitioning may produce floorplans in which highly connected modules end up in diagonally opposite chip regions. To overcome such an undesirable situation, we propose to swap the subfloorplans and retain the configuration which improves the total wirelength.

4.1.4. Handling congestion and timing constraints

We need to handle additional constraints like **congestion** and **timing** in our hierarchical floorplanner. The **congestion** objective is inherently linked to a large extent to minimizing partitioning cut. Lesser the number of wires crossing the two partitions, less congested that boundary of partitions is going to be.

For taking **timing** into consideration, we note that a critical path is composed of several critical nets. Meeting the delay requirement of a critical path means smaller delay values for all these nets. Since delay is inherently related to the length of the path, by restricting such lengths at the partitioning stage we can reduce the overall delay of the critical path. For this we use a simple strategy of assigning large weights on the critical nets (nets on the critical paths). The partitioner *hMetis* will ensure that such high-weight nets are respected while partitioning. Though such a strategy may not always guarantee improvement in delay of individual critical paths, we observed that overall delay improvement (reduction in number of critical paths and total sum of all the delay violations) was quite satisfactory.

4.2. Use Low Temperature Annealing for shape management

One major limitation of hierarchical floorplanning has been its inability to optimize on area. Though various modifications of hierarchical floorplanning have been suggested, none of them has been able to match the shape-management achieved by full-blown Simulated Annealing floorplanner **(FullSA)**. In this work our main focus has been the optimization of objectives like wirelength, congestion and timing when the shape of the final floorplan is fixed. Even with a lot of flexibility in the representation of modules, the hierarchical floorplanner alone doesn't prove to be effective for constrained-shape requirement. For this purpose we employ Low Temperature Simulated Annealing on the floorplan obtained by Partitioning-Based floorplanner to obtain the desired shape (we call this approach of hierarchical partitioning followed by Low Temp SA as **HierPlusSA**). At the same time we need to make sure that we do not interfere too much with other objectives like wirelength, congestion and timing. The Simulated Annealing floorplanner (FullSA) that we will discuss in the next section can be suitably employed to meet the shape-constraint with little or no degradation in quality.

5. EXPERIMENTAL SETUP AND RESULTS

For our work we have chosen four circuits (ind1, ind2, hway2, fract) for modeling the floorplans. The modules in our floorplan circuits are the cluster of cells from the circuits used in placement. We used TimberWolf1.4.0 to place these cluster of cells with varying aspect-ratios to generate shapes for the modules. The flexible modules in our floorplan circuits have aspect-ratios ranging from 3 to $1/3$. Each floorplan circuit has four different percentages of rigid modules in the system (From 10% to 50%. ind1.10 implies that circuit ind1 has 10% rigid modules in it). The percentage of white space (as a percentage of total area of modules) is also given. The critical paths and associated maximum delays are assumed to be provided with the circuits. We model critical paths after observing the delays of few large delay nets in the unoptimized floorplan. The details about these circuits are given in Table 1. All the tests have been carried on Sun Ultra-10 workstations.

Table 1. Test Circuits

circuit	# of modules	# of nets	% of white space
ind1	20	19	10
ind2	40	40	10
hway2	73	67	10
fract	149	147	10

We compare our approach (HierPlusSA) with full-blown Simulated Annealing floorplanner (FullSA). For this we need to add new constraints:shape, congestion, timing into Simulated Annealing cost-function.

5.1. Modifying Simulated Annealing to include additional constraints

Our Simulated Annealing floorplanner is based on the Wong-Liu floorplanner discussed in [19]. Since for constrained floorplanning there are several other optimization objectives, we need to include them in the cost too. Various components of cost-function:*ShapeCost, WireLength, CongestionCost and TimingCost* have already been discussed in Section 3..

The over all-cost function is $\psi = \alpha.ShapeCost + \beta.WireLength + \gamma.CongestionCost + \eta.TimingCost$. The only near-continuous term in the cost function ψ is WireLength. Hence for setting the relative importance of these terms we choose Wirelength as the normalization factor. The values of α, γ and η are normalized at the beginning of annealing with respect to WireLength or at the time these costs are switched on in the annealing process. Just like Wong-Liu floorplanner the temperature schedule is of the form $T_k = r.T_{k-1}$, $k = 1, 2, ...$. The initial temperature T_0 is determined by performing a sequence of random moves as described in [19]. These initial random moves also help in setting the values for α, γ and η which remain unchanged throughout the annealing process (if any cost function is switched on at an intermediate stage then the wirelength at that step is used to normalize it). To account for the sudden jumps in the cost function we use the strategy proposed for Temperature Adjustment in [2]. However, instead of adjusting the temperature when turning on a new objective function we update the Boltzmann constant k.

Table 2. Comparing FullSA with TimberWolf

circuit	FullSA		TimberWolf1.4.0	
	area	cpu(s)	area	cpu(s)
ind1.10	10400	66	11858	47
ind2.10	26376	600	28322	551

We also compare our Simulated Annealing floorplanner (FullSA) with TimberWolf1.4.0 floorplanner to make sure that our implementation is fast and of good quality when compared to other Simulated Annealing implementations. Since the wirelength models of the two are different, we compare area and run-times. It turns out that our Simulated Annealing floorplanner is as fast as TimberWolf Simulated Annealing floorplanner and the area results (TimberWolf produces a larger area because it leaves some space between modules for routing channels or pad spacing) are comparable, as given in Table 2. As our major emphasis for comparison has been run-times, we are reporting results for only two of the test circuits (remaining circuits show similar trend).

In the following sub-sections we show the results for floorplanning under congestion and timing control. Wirelength is the summation of center-to-center Euclidean distances between connected modules. Congestion is reported as the summation of capacity violations on all the bin boundaries. Timing violations are reported in terms of : total violation on all the critical paths and number of critical paths violating the delay constraints. In our timing model we have chosen the values of r_x, r_y, c_x and c_y to be 1 (refer equation 3). The value of R_s in equation 4 is chosen to be 0.

5.2. Performance of FullSA and HierPlusSA under Congestion and Timing control

The results for FullSA and HierPlusSA under congestion and timing control are shown in figures 4 and 5 respectively. The data associated with these bar-charts can be found in Table 3. The results are categorized under four different columns: when both congestion and timing optimizations are off, when congestion optimization is on for 100% of the temperature steps and timing optimization is off, when congestion optimization is off and timing optimization is on for 100% of the steps and when both of these are on for 100% of the steps. *For HierPlusSA if we want timing optimization off, we do not weight the nets on the critical path (at the min-cut partitioning state) and turn off timing optimization in Low Temperature Simulated Annealing.*

It is easy to observe that the trends of the results for various optimization scenarios is similar for FullSA and HierPlusSA. As is clear from the results, turning congestion optimization on for 100% of the steps does lead to a significant decrease in the congestion of the circuit (though it leads to an increase in the run time by about 50%, refer Table 3). When timing optimization is turned on for 100% of the steps, the total timing violation and number of violating paths reduce drastically. The case where both congestion and timing optimizations are on for 100% of the steps shows a definite improvement over the case when both of these are off. To sum up the results,

- Full congestion optimization leads to a drastic decrease in congestion but the timing violation remains unchanged.
- Full timing optimization reduces the timing violation (in terms of total violation and number of violating critical paths) but the overall congestion of the circuit shows no change in general.
- Selective optimization of congestion or timing produces better results for congestion and timing respectively. However, the congestion and timing results with both the optimizations on are as good as the selective optimization results.

5.3. Comparing FullSA and HierPlusSA

The results comparing FullSA with HierPlusSA are given in Table 3. Once again we are reporting comparative results for the four optimization scenarios: when both congestion and timing optimizations are off, when congestion is on and timing is off, when timing is on and congestion is off and when both congestion and timing optimizations are on. The results can be summed up as follows,

- Even when both congestion and timing optimizations are off, HierPlusSA produces better congestion and timing results than that of FullSA.
- HierPlusSA produces overall better congestion results than FullSA when only congestion optimization is turned on.
- HierPlusSA achieves significant reduction in timing violations when compared to FullSA for the case when only timing optimization is on.
- For the case when both congestion and timing optimizations are on, HierPlusSA produces better congestion results and much better timing violations.

The results indicate that HierPlusSA outperforms FullSA in all the floorplan metrics (wirelength, congestion, timing) and across all the optimization scenarios (bold-faced data shows which approach performed better). In all the cases the improvement in

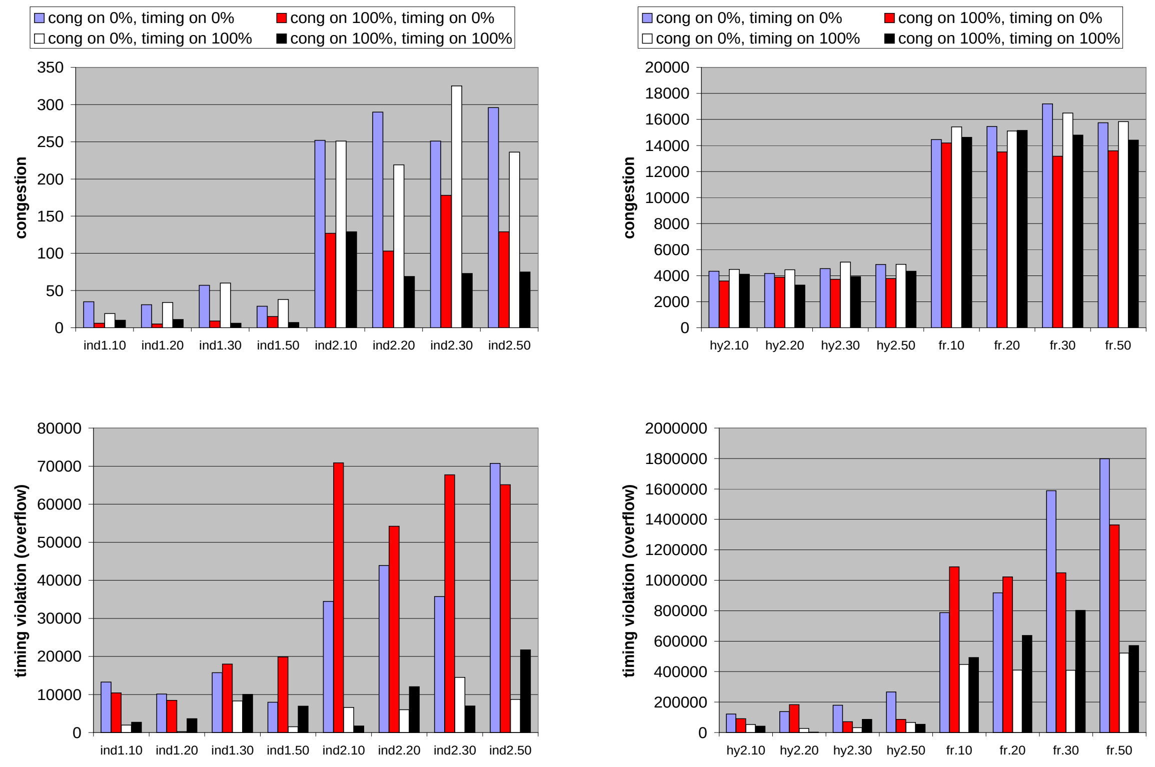

Figure 4. Performance of FullSA under congestion and timing control

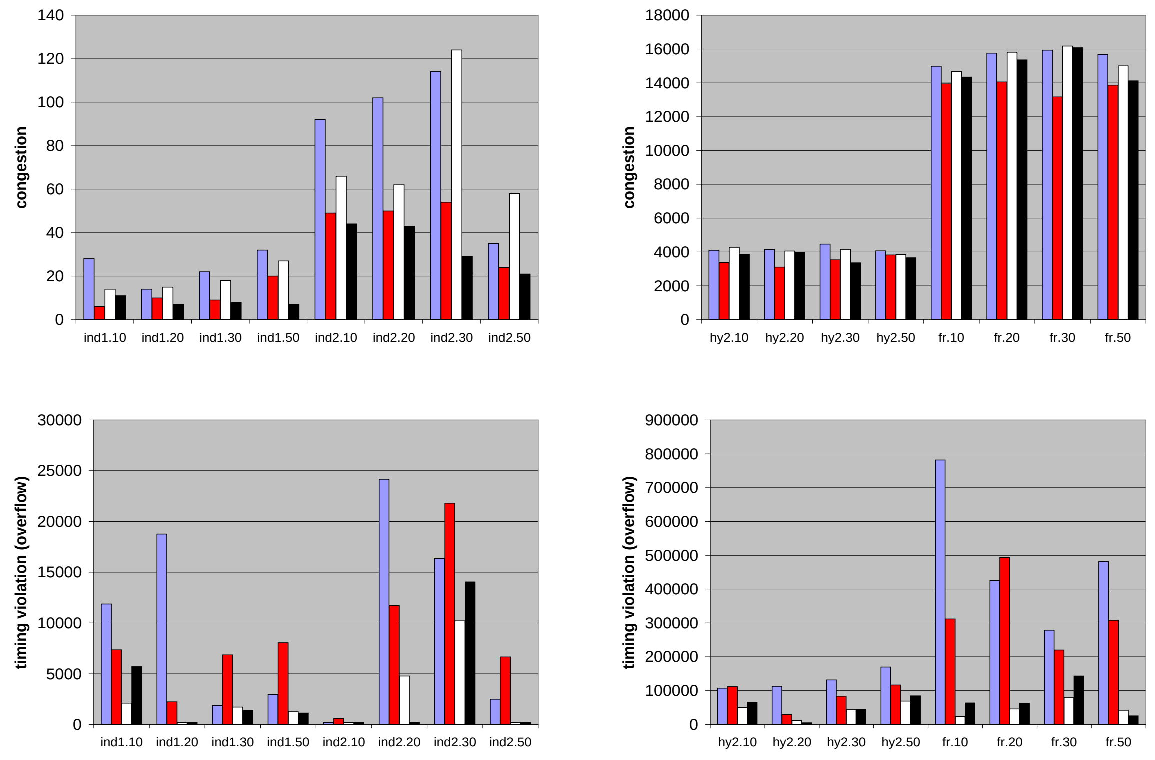

Figure 5. Performance of HierPlusSA under congestion and timing control

Table 3. Comparing HierPlusSA with FullSA

congestion optimization on for 0%, timing optimization on for 0%										
	FullSA				HierPlusSA					
ckt	WL	cong	tim vio	cpu(s)	WL	cong	tim vio	cpu(s)	% imp WL	speed up
ind1.10	779	35	**13255, 2**	80	691	**28**	**11887, 3**	1.2	11.3	67
ind1.20	674	31	**10136, 3**	115	686	**14**	18751, 3	1.2	-1.8	96
ind1.30	866	57	15749, 3	73	629	**22**	**1856, 2**	0.7	27.4	104
ind1.50	728	**29**	7956, 3	55	802	32	**2957, 2**	0.7	-10.2	79
ind2.10	2325	252	34433, 4	602	1753	**92**	**0, 0**	6.2	24.6	97
ind2.20	2536	290	43862, 4	468	2183	**102**	**24166, 2**	4.7	13.9	100
ind2.30	2477	251	35710, 4	551	1796	**114**	**16365, 2**	6.4	27.5	86
ind2.50	2779	296	70724, 4	395	1948	**35**	**2488, 2**	4.6	30.0	86
hway2.10	6530	4344	121604, 7	3045	5576	**4100**	**107195, 4**	29.4	14.6	104
hway2.20	6520	4166	137323, 6	3314	5636	**4149**	**112547, 4**	27.3	13.6	121
hway2.30	6283	4539	178973, 6	2529	5869	**4465**	**131371, 5**	29.8	6.6	85
hway2.50	6662	4855	266412, 8	1877	5640	**4073**	**169745, 4**	22.5	15.3	83
fract.10	21094	**14450**	787387, 14	14830	16141	14983	**781428, 12**	128	23.5	116
fract.20	22794	**15459**	918123, 15	14974	14372	15757	**425217, 11**	124	37.0	121
fract.30	22324	17196	1588353, 15	11818	16810	**15936**	**278809, 11**	117	24.7	101
fract.50	24117	15746	1797969, 15	8815	16202	**15681**	**481383, 12**	85	32.8	104
congestion optimization on for 100%, timing optimization on for 0%										
	FullSA				HierPlusSA					
ckt	WL	cong	tim vio	cpu(s)	WL	cong	tim vio	cpu(s)	% imp WL	speed up
ind1.10	754	**6**	10412, 3	136	732	**6**	7360, 3	1.5	2.9	91
ind1.20	783	**5**	8446, 3	166	718	10	2234, 2	1.6	8.3	104
ind1.30	834	**9**	17996, 3	110	658	**9**	6848, 2	1.1	21.1	100
ind1.50	932	**15**	19835, 3	111	724	20	8056, 2	1.1	22.3	101
ind2.10	2579	127	70846, 4	848	1929	**49**	191, 1	8.2	25.2	103
ind2.20	2696	103	54212, 4	807	1990	**50**	11721, 2	6.0	26.2	135
ind2.30	2797	178	67718, 4	833	1809	**54**	21808, 2	7.1	35.3	117
ind2.50	3127	129	65101, 4	607	1634	**24**	6655, 1	5.2	47.7	117
hway2.10	6178	3592	90855, 4	4631	5067	**3368**	111924, 2	40.5	18.0	114
hway2.20	6625	3884	182656, 8	4659	4931	**3112**	29353, 3	37.4	25.6	125
hway2.30	6383	3720	71111, 5	3894	5169	**3535**	83388, 3	42.2	19.0	92
hway2.50	6853	**3781**	86410, 5	2910	5416	3827	116504, 3	31.6	21.0	92
fract.10	23320	14202	1088003, 15	21227	14613	**13942**	311522, 12	172	37.3	123
fract.20	22299	**13504**	1021613, 15	22535	15830	14058	493159, 12	182	29.0	124
fract.30	21958	13182	1049159, 15	22341	14745	**13168**	219781, 9	181	32.8	123
fract.50	23551	**13588**	1364156, 15	18213	15067	13860	307818, 11	141	36.0	129
congestion optimization on for 0%, timing optimization on for 100%										
	FullSA				HierPlusSA					
ckt	WL	cong	tim vio	cpu(s)	WL	cong	tim vio	cpu(s)	% imp WL	speed up
ind1.10	701	19	**1960, 1**	132	634	14	2098, 1	1.2	9.6	110
ind1.20	637	34	246, 1	140	676	15	**0 , 0**	1.4	-6.1	100
ind1.30	988	60	8323, 3	71	697	18	**1707, 1**	1.0	29.4	71
ind1.50	996	38	1516, 1	54	614	27	**1249, 1**	0.7	38.4	77
ind2.10	2487	251	6563, 2	734	1795	66	**0, 0**	7.0	27.8	105
ind2.20	2602	219	5972, 2	584	1817	62	**4764, 1**	5.3	30.2	110
ind2.30	2704	325	14485, 2	779	1959	124	**10218, 1**	6.2	27.6	126
ind2.50	2639	236	8716, 2	477	1789	58	**0, 0**	3.8	32.2	126
hway2.10	6543	4483	52933, 2	4513	5677	4278	**50525, 2**	31.1	13.2	145
hway2.20	6526	4449	27414, 3	4275	5344	4060	**11880, 2**	28.8	18.1	148
hway2.30	7159	5041	**31700, 3**	3414	5810	4161	**43623, 2**	31.6	18.8	108
hway2.50	6892	4872	**66701, 4**	2628	5995	3853	68894, 4	19.6	13.0	134
fract.10	23175	15427	446781, 12	14738	13876	14656	**23361, 3**	111	40.1	133
fract.20	22724	15109	409753, 12	15575	16345	15814	**45628, 2**	120	30.4	130
fract.30	23530	16499	409679, 12	14196	14691	16171	**78941, 4**	115	37.6	123
fract.50	23863	15832	522467, 12	9275	15071	15002	**42137, 4**	86	36.8	108
congestion optimization on for 100%, timing optimization on for 100%										
	FullSA				HierPlusSA					
ckt	WL	cong	tim vio	cpu(s)	WL	cong	tim vio	cpu(s)	% imp WL	speed up
ind1.10	710	**10**	**2692, 1**	168	697	11	5695, 2	1.7	1.8	99
ind1.20	711	11	3616, 1	164	644	**7**	**0, 0**	1.6	9.4	103
ind1.30	793	**6**	10049, 3	116	710	8	**1408, 1**	1.3	10.5	89
ind1.50	735	**7**	6921, 2	104	614	**7**	**1132, 1**	0.9	16.5	116
ind2.10	2340	129	1698, 1	885	1992	**44**	**0, 0**	7.8	14.9	113
ind2.20	2641	69	12059, 2	723	1993	**43**	**0, 0**	6.0	24.5	121
ind2.30	2366	73	**6965, 2**	784	2060	**29**	14038, 2	7.5	12.9	105
ind2.50	2934	75	21742, 3	609	1559	**21**	**0, 0**	5.1	46.9	119
hway2.10	6622	4111	**41553, 2**	4755	6183	**3871**	65540, 3	41.3	6.6	115
hway2.20	6639	**3270**	**2826, 2**	5612	5789	3989	**5215, 1**	38.9	12.8	144
hway2.30	6667	3919	86486, 4	3921	5522	**3366**	**44779, 2**	40.3	17.2	97
hway2.50	7223	4343	**54235, 5**	3118	5464	**3661**	**84856, 4**	32.9	24.4	95
fract.10	22957	**14624**	492345, 13	24266	13943	14336	**63400, 4**	183	39.3	133
fract.20	24537	**15157**	637092, 13	22646	15356	15358	**62684, 5**	187	37.4	121
fract.30	25085	**14799**	802763, 13	22707	15772	16080	**143119, 5**	186	37.1	122
fract.50	23782	14412	571839, 13	17699	15420	**14115**	**25513, 4**	159	35.2	111

wirelength is 20% on average. The major gain is evident when we compare the speed of two approaches. Our algorithm (HierPlusSA) is, on average, **100 times faster** than traditional Simulated Annealing based floorplanner (FullSA).

6. CONCLUSION AND FUTURE WORK

In this work we have proposed faster methods for constrained floorplanning. We have shown that a careful modification of hierarchical floorplanner can perform as good as or better than Simulated Annealing based floorplanner. We have incorporated congestion and timing constraints into both Simulated Annealing and Hierarchical floorplanners. The comparative results show that we can produce floorplans with 20% better wirelength on average, better in congestion and timing optimization and at the same time achieving a speedup of 100 across a variety of test circuits. As the future work in this direction, we propose to integrate better timing and congestion optimization techniques with our floorplanning approach. Also we intend to study the effect of such early optimizations on final layout, a detailed placed and routed chip.

REFERENCES

[1] Tsu chang Lee. "A Bounded 2D Contour Searching Algorithm For Floorplan Design With Arbitrarily Shaped Rectilinear And Soft Modules". In *Design Automation Conference*, pages 525–530. IEEE/ACM, 1993.

[2] H. Chen, H. Zhou, F. Young, D. Wong, H. Yang, and N. Sherwani. "Integrated Floorplanning and Interconnect Planning". In *International Conference on Computer-Aided Design*, pages 354–357, 1999.

[3] S. Choi and C. Kyung. "A Floorplanning Algorithm Using Rectangular Voronoi Diagram and Force-Directed Block Shaping". In *International Conference on Computer-Aided Design*, pages 56–59, 1991.

[4] J. Cong, T. Kong, and D. Pan. "Buffer Block Planning for Interconnect-Driven Floorplanning". In *International Conference on Computer-Aided Design*, pages 358–361, 1999.

[5] W. M. Dai and E. S. Kuh. "Simultaneous Floorplanning and Global Routing for Hierarchical Building-block Layout". *IEEE Transactions on Computer Aided Design*, 6(5):828–837, 1987.

[6] S. Dong, J. Cong, and C. Liu. "Constrained Floorplan Design for Flexible Blocks". In *International Conference on Computer-Aided Design*, pages 488–491, 1989.

[7] P. Guo J. Xu and C. Cheng. "Rectilinear Block Placement Using Sequence-Pair". In *International Symposium on Physical Design*, pages 173–178. IEEE/ACM, 1998.

[8] M. Kang and W. Dai. "Topology Constrained Rectilinear Block Packing For Layout Reuse". In *International Symposium on Physical Design*, pages 179–186. IEEE/ACM, 1998.

[9] U. P. Lauther. "A Min-cut Placement Algorithm For General Cell Assemblies Based On A Graph Representation". In *Design Automation Conference*, pages 1–10. IEEE/ACM, 1979.

[10] W. K. Luk. "Multi-Terrain Partitioning and Floorplanning for Data-Path Chip (Microprocessor) Layout". In *International Conference on Computer-Aided Design*, pages 492–495, 1989.

[11] R. H. J. M. Otten. "Efficient Floorplan Optimization". In *International Conference on Computer Design*, pages 499–503. IEEE/ACM, 1983.

[12] A. Ranjan, K. Bazargan, and M. Sarrafzadeh. "Floorplanner 1000 Times Faster: A Good Predictor and Constructor". In *System Level Interconnect Prediction*, pages 115–120, 1999.

[13] G. Stenz, B. Riess, B. Rohfleisch, and F. Johannes. "Timing Driven Placement In Interaction With Netlist Transformations". In *International Symposium on Physical Design*, pages 36–41. IEEE/ACM, 1997.

[14] L. Stockmeyer. "Optimal Orientation of Cells in Slicing Floorplan Designs". *Information and Control*, 57(2):91–101, 1983.

[15] H. Su, C. Allen, and Y. Lin. "A Timing-Driven Soft-Macro Resynthesis Method in Interaction with Chip FloorplanninG". In *Design Automation Conference*, pages 262–267. IEEE/ACM, 1999.

[16] M. Wang and M. Sarrafzadeh. "On the Behavior of Congestion Minimization During Placement". In *International Symposium on Physical Design*, pages 145–150. IEEE/ACM, 1999.

[17] T. C. Wang and D. F. Wong. "An Optimal Algorithm for Floorplan Area Optimization". In *Design Automation Conference*, pages 180–186. IEEE/ACM, 1990.

[18] D. F. Wong, H. W. Leong, and C. L. Liu. *"Simulated Annealing for VLSI Design"*. Kluwer Academic, 1988.

[19] D. F. Wong and C. L. Liu. "A New Algorithm For Floorplan Design". In *Design Automation Conference*, pages 101–107. IEEE/ACM, 1986.

[20] D. F. Wong and P. S. Sakhamuri. "Efficient Floorplan Area Optimization". In *Design Automation Conference*, pages 586–589. IEEE/ACM, 1989.