Congestion Minimization During Placement Without Estimation

Bo Hu and Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
{hb, mms}@ece.ucsb.edu

Abstract

This paper presents a new congestion minimization technique for standard cell global placement. The most distinct feature of this approach is that it does not follow the traditional “estimate-then-eliminate” strategy. Instead, it avoids the excessive usage of routing resources by the “local” nets so that more routing resources are available for the uncertain “global” nets. The experimental results show that our new technique, SPARSE, achieves better routability than the traditional total wire length (Bounding Box) guided placers, which had been shown to deliver the best routability results among the placers optimizing different cost functions [2]. Another feature of SPARSE is the capability of allocating white space implicitly. SPARSE exploits the well known empirical Rent’s rule and is able to improve the routability even more in the presence of white space. Compared to the most recent academic routability-driven placer Dragon[8], SPARSE is able to produce solutions with equal or better routability.

1. Introduction

As one of the traditional placement objectives, congestion minimization has been researched for more than a decade. A placement with good routability not only saves time spent on routing, it also correlates the timing estimation at the placement stage with the end result after routing, causing routing to avoid unnecessary detours. The importance of good routability is more evident in deep sub-micron technology where interconnects dominate the performance, and possibly the area, of a VLSI circuit. Traditional logic synthesis captures the gate contribution to the chip area but does not account for the area of interconnects. Since interconnects tend not to scale with the gates, a design with minimum area achieved by logic synthesis might not be routable due to wiring congestion. As a result, the chip size may need some expansion to provide more routing resources to complete the design.

To minimize the congestion, numerous approaches have been proposed in the past years. Most of them [1], [2], [3], [5], [6], [7], [8], [9]share the same “estimate-then-eliminate” strategy. For estimation, they apply the probabilistic routing similar to [4] or a simplified global routing. Performing a detailed routing step is too CPU-intensive. In the probabilistic approach, each net has several candidate routes, and the probability of going through a particular routing edge can be calculated. To eliminate the congestion, [1] includes congestion data into its simulated annealing based formulation. In [6], the traditional quadratic placement formulation is modified to minimize congestion. In [5], a new multi-partitioning heuristic is proposed to take into account wire congestion.

In the presence of white space, routability can be effectively improved as shown in [3], [8], [9]. The idea is to spread the cells in the congested regions so as to reduce the routing resource demands.

It is generally believed that at the placement level, total-wire-length cost function correlates well with total routed wire length. Furthermore, it was shown in [2] that the total-wire-length optimizing placers achieve the best routability results when compared to placers optimizing various other cost functions. To eliminate the congestion further, [2] proposed a post processing technique.

Although these previously proposed approaches are effective in eliminating the congestion, they have not provided an insight into the question of what netlist structure produces predictable interconnects. Thus, they cannot directly guide the logic synthesis to construct better routable net structures at the logic level.

Another strand of previously proposed approaches for minimizing congestion have been developed in the FPGA context. In [12], Rent’s rule is employed to match the routing resource demands with the real estate supply in order to avoid routing overflows in local spots. In [13], Rent’s-rule-based clustering and placement techniques are proposed to efficiently reduce the maximum number of tracks required to route the whole design.

In this paper, we present a new congestion minimization technique for the fixed-die standard cell global placement. The most distinct feature of this technique is that it does not follow the traditional “estimate-then-eliminate” strategy. Instead, it avoids excessive usage of routing resources by the “local” nets so that more routing resources are available for the uncertain “global” nets. Our experimental results show that the new placer, SPARSE, achieves better routability than do the total-wire-length minimizing techniques. In addition, SPARSE is capable of allocating white space implicitly. SPARSE exploits a well known empirical Rent’s Rule and produces placement solutions with equal or better routability than the most recent academic routability-driven placer Dragon[8].

The rest of this paper is organized as follows. In section 2, we define terms and formulate the congestion-driven global placement problem in the fixed-die context. In section 3 we conduct a thorough analysis of congestion. We propose an approach to the estimation-free congestion minimization. Section 4 discusses the Rent’s-rule-based implicit white-space allocation. Section 5 explains our global placer, SPARSE. Experimental results are given in section 6, followed by conclusions in section 7.

0-7803-7607-2/02/$17.00 ©2002 IEEE

2. Problem Formulation

In this section we first give definitions used throughout this work, and then formulate the fixed-die standard cell placement problem.

2.1 Definitions

A die has a core area which consists of a set of rows *R* for standard cell placement. Each row *r* in R defines a number of available sites *S(r)*. The total number of the available sites for standard cells, denoted by *TS,* is the summation of *S(r)* over all rows. This is analytically expressed in EQ1.

$$TS = \sum_r S(r) \qquad \text{(EQ1)}$$

The fixed-die context suggests that the sets *R* and *S(r)* are fixed during the placement. The core area is divided into a set of rectangular global bins *B*. We denote the total area of the cells in a bin *b* as *a(b)*, and the capacity of *b* as *c(b)*. For each bin *b*, we have the following capacity constraint:

$$a(b) \le c(b) \qquad \text{(EQ2)}$$

A placement is called *legal* when the capacity constraints of all the bins are satisfied. A *global routing graph GRG(N,E)* is constructed based on this bin structure. Each bin corresponds to a node *n* in *N* and each boundary between two bins defines an edge *e* in *E*. Each routing edge *e* has a capacity *ec(e),* which is the number of routing tracks assigned to *e*. We use *ea(e)* to denote the actual number of nets going through *e* after routing. In EQ3 we define the edge overflow *eo(e)*. The total overflow *TO* is the summation of *eo(e)* over all routing edges as stated in EQ4.

$$eo(e) = \begin{cases} 0 & ea(e) \le ec(e) \\ ea(e) - ec(e) & ea(e) > ec(e) \end{cases} \qquad \text{(EQ3)}$$

$$TO = \sum_e eo(e) \qquad \text{(EQ4)}$$

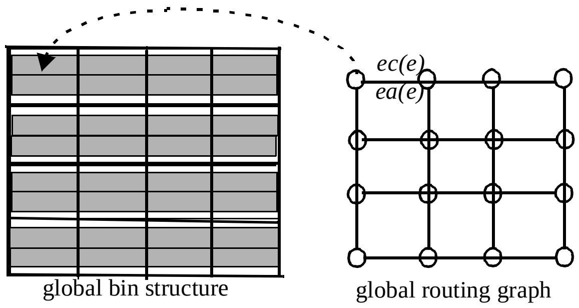

Fig. 1: Global Bin Structure and Corresponding Global Routing Graph

Figure 1 illustrates the global bin structure and its corresponding routing graph. The core area in the figure is partitioned into 4 x 4 bins. In this example, each global bin spans two rows. In the figure the mapping between a global bin and a node in GRG is indicated by the dashed arrow.

We denote the netlist to be placed by *NT(G,NE). G* and *NE* are the sets of gates and nets respectively. The degree of a net *i,* denoted by *d(i),* is the number of terminals *i* has. In bin-based placement, we also define a global degree of a net *i*, denoted by *D(i)*, as the number of global bins where *i* has at least one terminal.

To make placement feasible, the total available sites *TS* in the core area must be equal to or larger than the total cell area *TA*. If *TS* is larger than *TA*, we define *White Space(WS)* as *TS - TA*. Generally, the ratio of *White Space* in a VLSI design can range from 0.1% to 30%. We also define white space with respect to a bin *b* as *ws(b) = c(b) - a(b).*

2.2 Problem Formulation

We formulate the congestion-driven global placement problem as follows:

Given a global bin structure *B* and the corresponding routing graph *GRG(N,E),* place the netlist *NT(G, NE)* into *B* such that the *Total Overflow(TO)* is minimized while all the bin capacity constraints are satisfied. *TO* is computed after global routing has been performed on the routing graph *GRG(N,E).*

3. Congestion Analysis

Congestion occurs when routing demand exceeds the routing resource supply. A feasible placement is one in which the routing demand matches the routing supply across the chip. As has been shown in the previous works, minimizing the total wire length decreases the wire congestion as well. But it is often the case that local congestions cannot be completely eliminated. The reason is that the total wire length model captures only the total routing resource utilization, but has no provision for showing how local routing resources are being used. The total wire length minimization tends to cluster the strongly connected cells. As a result, local routing resources might be in excessive demand. In such a situation, either the region itself is not routable, or it cannot provide routing resources for nets attempting to pass through it. In either case, some nets will have to detour and end up with longer, unpredictable lengths. If too many nets detour, the final routed wire length differs considerably from the initial estimation at the placement phase.

To investigate the characteristics of congestion, we have performed the following local congestion analysis. We take MCNC and IBM placement benchmarks and carry out wire-length driven global placement and global routing on the global bin routing graph data structure. Before we analyze systematically the nature of congestion in the routing edges, we wish to distinguish local nets from global ones:

Definition 1: Local Certainty of a routing edge *e,* denoted by *LC(e),* is the number of nets which are going through *e* and have at least one terminal in either routing node incident to *e.* Each net *i* contributing to *LC(e)* is referred to as a Local Certainty Net, or simply a local net, with respect to the routing edge *e*.

Definition 2: Global Uncertainty of a routing edge *e*, denoted by *GU(e)*, is the number of nets which are passing through *e* and do not have any terminal in either routing node incident to *e.* Each net *i* contributing to *GU(e)* is referred to as a Global Uncertainty Net, or simply, a global net, with respect to the routing edge *e.*

We note that our concept of the local and global nets differs from the traditional one. It is possible that a short net is actually global with respect to some routing edge. At the same time, a long net must be local with respect to the edges which are incident to its terminals.

Based on the previous definitions, we introduce the concept of global uncertainty of a design.

Definition 3: Global Uncertainty of a placed and routed design, denoted by *GU*, is the summation of *GU(e)* over all routing edges *e*.

We also refer to the total number of routing tracks consumed by the Local Certainty Nets and Global Uncertainty Nets as the *Local Certainty Net Consumption* and *Global Uncertainty Net Consumption*, respectively.

After the placement and routing, we collect Local Certainty and Global Uncertainty information on each routing edge and list their average ratios in Table 1. In this investigation, each bin contains approximately 10~30 cells.

	Local Certainty (%)	Global Uncertainty(%)
biomed	79%	21%
industry2	79%	21%
ibm01	77%	23%
ibm02	67%	33%
Ave	**75.5%**	**24.5%**

TABLE 1. Local Certainty vs. Global Uncertainty

The data in Table 1 suggest that for each routing edge, on the average, Local Certainty Nets consume the majority of routing resources. We also observe that the Local Certainty Nets are more predictable than the Global Uncertainty Nets. For a given routing edge *e* and a net contributing to *LC(e)*, we instantly know that this net has at least one terminal in either one of two routing nodes incident to *e*. These observations motivate us to focus on the Local Certainty Nets for congestion minimization. The idea is to suppress the Local Certainty Net Consumption while keeping the Global Uncertainty Net Consumption approximately the same. By doing this, it is possible to achieve the following two objectives: (1) More Global Uncertainty Nets can be accommodated in a local region, since the Local Certainty Net Consumption in this region is reduced; (2) More freedom can be provided to accommodate the Global Uncertainty Nets. For instance, a global net could have more candidate routes, since overall the Local Certainty Net Consumption is reduced. It is worth mentioning that by restricting Local Certainty Net Consumption, we must not let the Global Uncertainty grow in an uncontrolled manner. If not, the growth of the Global Uncertainty might overwhelm the benefits of the Local Certainty reduction.

Based on this analysis, we propose to optimize the Local Certainty Net Consumption in order to reduce the congestion. However, applying the model developed so far, we would still need to predict the routes for all the nets to determine *LC(e)* and *GU(e)* for the routing edges. To eliminate completely the necessity of route prediction, we take the following bin-centric, instead of edge-centric, perspective.

Definition 4: A Local Certainty of a global bin *b*, denoted by *LC(b)*, is the number of nets which have at least one terminal in *b* and at least one terminal outside *b*. Similarly, each net contributing to *LC(b)* is referred to as a Local Certainty Net, or simply a local net with respect to bin *b*.

Since the Local Certainty Nets with respect to a bin have to pass through one of the four adjacent routing edges to reach other terminals, restricting the Local Certainty of a global bin is equivalent to restricting the total Local Certainty of its adjacent routing edges. We prefer *LC(b)* over the *LC(e)* because *LC(b)* is much easier to maintain and, more importantly, no probabilistic predictions are required for computing it.

4. Rent's-Rule-Based Implicit White Space Allocation

In the presence of white space, a simple total-wire-length-based placement tends to cluster cells as close as possible to minimize the cost function. As a result, this cost function is unable to effectively exploit the potential of available white space for congestion improvement. This also explains why previous works[8][9] handle white space allocation using a partitioning-based approach instead of total wire length as the optimization metric.

In the 1960s, several researchers independently discovered that an exponential relationship exists between a block of logic gates and the IO terminals it needs to communicate with the external gates. This relationship is called the Rent's rule and is stated in EQ5.

$$I = KB^{p} \qquad \text{(EQ5)}$$

In EQ5, K is the average number of terminals per gate, and B is the number of gates in a logic block. I gives the number of terminals required by the logic block for external communication, and this requirement is controlled by the exponent p, called the *Rent's Parameter*. It was shown that p is a measure of the design's interconnect complexity. In general, the larger p is, the more routing resources are required for the same amount of logic.

The Rent's rule says that a certain number of IO terminals are expected by a block of logic gates. For example, if we are given a design which is characterized by *Rent's Parameter* p and pick a logic block, the expected number of interconnects originating from this logic block is given by the Rent's Rule. In this case, we derive the number of IO terminals from the number of gates. On the other hand, we can also rewrite EQ5 into a form shown in EQ6, which expresses the number of gates, B, as a function of the number of IO terminals, I.

$$B = \left(\frac{I}{K}\right)^{\frac{1}{p}} \qquad \text{(EQ6)}$$

The equation above determines the number of gates, or the logic, from the given number of IO terminals I and the *Rent's Parameter* p. This mutual dependency between the number of logic gates and IO terminals suggests that in our bin-based placement, if we restrict (or reduce) the IO terminals of a bin *b*, *LC(b)*, we expect the number of logic gates in *b* to be reduced as well. This observation enables us to take a net-centric point of view in allocating the white space, which is different from the approaches in previous works which focus on logic gates. To allocate white space in a bin *b*, some logic gates assigned to *b* have to move outside of *b*. Instead of explicitly restricting the use of silicon in *b*, we exploit the intrinsic relationship between the logic and IO terminals, and try to restrict the use of IO terminals in order to lower the complexity of the internal logic.

5. SPARSE: A Congestion Driven Global Placer

Following the analysis in sections 3 and 4, we propose the congestion minimization technique which does not require probabilistic routing estimation. The idea is to restrict (evenly distribute) routing resources consumed by the Local Certainty nets so that the unpredictable Global Uncertainty nets can be accommodated better. In the presence of white space, our technique exploits the intrinsic relationship between the interconnect and logic utilization (as discussed in section 4), and allocates the white space implicitly. Our new placer, SPARSE, is based on a simulated annealing algorithm and a novel cost function which is derived from the traditional total-weighted wire-length. In section 5.1, we first give the cost function template with an unspecified parameter - *Sparse parameter.* Then in section 5.2 we show how the *Sparse parameter* can be determined.

5.1 The Cost Function Template

We begin with the traditional total-weighted wire-length-based cost function stated in EQ7.

$$CF = \sum_i w_i BB(i) \qquad \text{(EQ7)}$$

In the cost function above, *BB(i)* is the half-perimeter of the net's *i* bounding box, w_i is the adjustment factor for the multiple-pin nets. In case of timing optimization, w_i can also be used to favor timing critical nets.

For each global bin *b* we compute a parameter called the *Sparse Parameter* $P_s(b)$. Using $P_s(b)$, we modify EQ7, which becomes EQ8:

$$CF' = \sum_i w_i \left(\frac{\sum_{b \in i} P_s(b)}{D(i)} \right) BB(i) \qquad \text{(EQ8)}$$

In EQ8, *D(i)* is the global degree of a net *i*. The average value of $P_s(b)$ over all global bins where the net *i* has terminals, is used as an extra weight imposed on it. As a result, the modified version makes it more desirable that the terminals of a net are placed inside the global bins with low *Sparse Parameter.* If $P_s(b)$ is designed to characterize the Local Certainty *LC(b)*, the cost function in EQ8 favors a solution with more balanced *LC(b)* distribution. For example, for the net *i* in figure 2, the new cost-function-based optimization will prefer the first topology instead of the second one, although the two topologies have exactly the same wire length. In the first topology, the terminal *t* is placed in

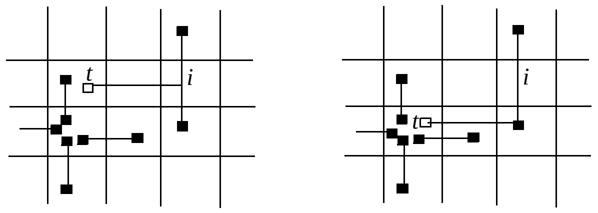

Fig. 2: An example for new cost function

a bin with smaller *LC(b)*. Supposing that each boundary between bins could accommodate only one routing track, it is clear that the second topology would make the design unroutable, while the first one would be routable.

5.2 Deriving *Sparse Parameter Ps(b)*

To derive $P_s(b)$, we start with the following requirements:

(1) $P_s(b)$ should reflect the Local Certainty *LC(b)*. Since our goal is to restrict the Local Certainty *LC(b)*, it is straightforward that $P_s(b)$ should be designed such that restricting $P_s(b)$ leads to the restriction of *LC(b).*

(2) $P_s(b)$ should be correlated with Global Uncertainty. A routing track, if occupied, is either consumed by a Local Certainty Net or a Global Uncertainty Net. The restriction of *LC(b)* leads to a more evenly distributed Local Certainty Net Consumption, and as a result, more routing tracks are available around the regions where it is reduced. These newly available tracks can be used to accommodate Global Uncertainty Nets, but in order to minimize congestion, we must make sure that the growth of Global Uncertainty is not going to overwhelm the newly available routing resources. Since Global Uncertainty is correlated with the total wire length, minimization of the new cost function in EQ8 should not disturb the total wire length in an uncontrolled manner. For example, suppose that a routing edge *e* with total 10 tracks(*ec(e)* = 10) has initially 8 Local Certainty nets and 3 Global Uncertainty nets. The overflow *eo(e)* = 8 + 3 - 10 = 1. Suppose that by restricting the Local Certainty consumption, we manage to reduce 8 Local Certainty nets to 6, while Global Certainty net count increases by 1. As a result, overflow is eliminated, since *eo(e)* = 6 + 4 - 10 = 0.

It is clear that *LC(b)* can be used as $P_s(b)$ because *LC(b)* trivially fulfills the first requirement. But a simple *LC(b)* based $P_s(b)$ does not satisfy the second requirement. The following example illustrates the reason:

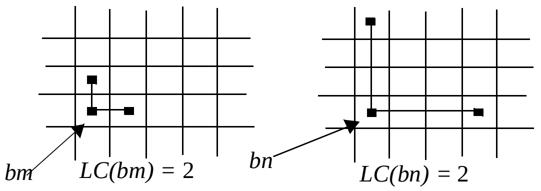

Fig. 3: Same *LC(b)* with different GU contribution

In figure 3, *LC(b)* based metric gives the same $P_s(b)$ value for *bm* and *bn*, since *LC(bm) = LC(bn)*. But the two cases have different effects on Global Uncertainty. The first case is more desirable because it does not increase the Global Uncertainty whereas the second case does. This example also demonstrates the correlation between wire length and Global Uncertainty. Increased wire lengths lead to increased Global Uncertainty.

This observation motivates us to take into account not only *LC(b)*, but also the length of each net *i* contributing to *LC(b)*. As a result, we choose a weighted sum of wire length as a base value to derive $P_s(b)$.

$$WS(b) = \sum_{i \in LC(b)} \frac{w_i BB(i)}{d(i)} \qquad \text{(EQ9)}$$

In EQ9, the wire length is weighted by its own degree d(i). Essentially, it means that small-fanout nets are prioritized to reduce the wire length. This weighting scheme is also consistent with the fanout distribution in a typical design, since small-fanout nets constitute a majority of total nets. In the meantime, for the same net, the smaller wire length suggests smaller WS(b) value. In addition, WS(b) correlates with LS(b) in that the fewer the number of nets in LS(b), the fewer the number of nets counted in EQ9. These observations suggest that WS(b) is a possible candidate to serve as $P_s(b)$.

It should be noted that in our current implementation, WS(b) captures only the nets contributing to LC(b). It ignores the nets totally absorbed inside global bins. The reason is that our global router counts only the overflow on the routing edges. But it is not hard to consider those "hidden" nets. For example, we could use pin density measure proposed in [9] to adjust WS(b) such that the more "hidden" nets there are, the larger will WS(b) and thus $P_s(b)$ become.

It is also worth mentioning that our new cost function is taking an interconnect-centric point of view. Consequently, the timing optimization can also be integrated into our scheme. For example, w_i in EQ8 and EQ9 can be adjusted to reflect timing criticality.

We choose as a final $P_s(b)$, a function of the form $SP(x) = ax^p + b$ which maps *WS(b)* to $P_s(b)$. This function, called the Sparse Function, is shown in Figure 4.

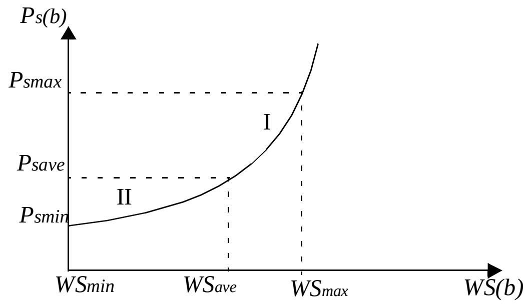

Fig. 4: Sparse Function: *WS(b)* to $P_s(b)$

In figure 4, WS_{min}, WS_{ave} , and WS_{max} represent the minimum, average and maximum *WS(b)* values respectively, and these values are obtained from an initial global placement. The decision to start from an initial placement is based on the fact that the core algorithm in SPARSE is simulated annealing. For very large benchmarks, it takes SPARSE a long time to finish if we perform placement from scratch. Moreover, the global placement problem has been well researched and several fast approaches have been proved effective. These include min-cut based, like Capo[11], and quadratic programming based, like Gordian[10]. For this reason, we consider SPARSE to be more of an intermediate step between global and detailed placement.

P_{smin}, P_{save} , and P_{smax} are user-defined. These three points, (WS_{min}, P_{smin}), (WS_{min}, P_{smin}), and (WS_{min}, P_{smin}), jointly determine the parameters, *a, b* and *p* in Sparse Function. In SPARSE, we deliberately introduce two distinct regions in the curve. Region I has a sharper slope. It corresponds to our optimization goal that penalizes excessive routing-resource usage by *LC(b).* Region II becomes flat as expected because the final cost function should converge to total wire length if *LC(b)* is relatively evenly distributed.

Once the Sparse Curve is determined, we build a look-up table to retrieve $P_s(b)$ value for a given *WS(b).* In the process of annealing, *WS(b)* and $P_s(b)$ are incrementally updated.

6. Experimental Results

To experiment with SPARSE, we first set up a design flow as shown in Figure 5.

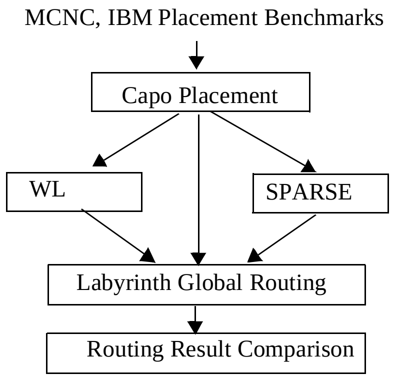

Fig. 5: SPARSE design flow

In the figure above, Capo[11] is called to deliver an initial global placement for SPARSE. WL is the traditional total-wire-length-based optimization using the cost function stated in EQ7. WL and SPARSE both take the initial global placement from Capo and perform low-temperature annealing optimization.

After placement optimization, the results from Capo, WL and SPARSE are fed into the Labyrinth global router downloaded from [14]. The routing results are noted for congestion analysis and comparison.

The benchmarks we use are MCNC Benchmarks and IBM Placement benchmarks available at [14]. The first three columns in Table 2 give the statistics for benchmarks used in the experiments. The fourth column shows the global bin structure for each benchmark. Each global bin contains roughly 10 to 20 standard cells. All the experiments are conducted on a 800Mhz Pentium III processor with 512M memory. CPU times are reported in minutes.

6.1 Placement Without White Space

The first experiment is used to show that SPARSE delivers a global placement with better routability if no extra space is available, that is, $WS = 0$. In the experiment, P_{smin}, P_{save}, and P_{smax} are set to 0.1, 0.5 and 2.5 respectively. The results are listed in Table 2.

Data in Table 2 suggest that WL and SPARSE achieve remarkable routability improvement as compared to Capo, since both WL and SPARSE take into account minimization of the Global Uncertainty. In addition, SPARSE is able to obtain even better results than WL. It is interesting that SPARSE actually achieves better total wire length after global routing than WL. For example, consider Table 2, and in particular, the data for benchmark *ibm04*. The pre-route estimation of the total wire length from WL is 141328 and from SPARSE is 145698, but after global routing, the total wire length of SPARSE is 92333 while that of WL is 98804, 7% larger than that of SPARSE. This indicates that SPARSE can achieve better routability with fewer nets detour-

	#gates	#nets	bin strut	total Half Bounding Box Wire length			#overflow			end wire length after routing			CPU(m)	
				Capo	WL	SP	Capo	WL	SP	Capo	WL	SP	WL	SP
bio	6417	5742	16x21	6033	4666	4720	244	165	40	8211	7458	6771	16	x2.5
ind2	12142	13419	20x32	24690	19178	19565	455	162	54	31821	25591	24419	41	x2.5
ind3	15059	21940	25x32	46683	37089	38324	632	35	28	55083	46020	45214	65	x2.4
avqs	21854	22124	28x41	20457	14879	14913	1250	60	29	24509	17983	17183	80	x2.5
ibm03	22207	21621	32x36	64845	56714	57645	796	131	23	88908	77941	71129	87	x2.8
ibm04	26633	26163	32x43	80561	70664	72849	1204	93	0	111643	98804	92333	121	x2.9
Total				**243269**	**203190**	**208016**	**4581**	**646**	**174**	**320175**	**273797**	**257049**		

TABLE 3. congestion minimization comparison between Capo, WL(total wire length based), and SPARSE (SP) with *WS* = 0

ing. Of course, since we have a much more complex cost function to update, CPU times are increased by 2 to 3 times.

6.2 Placement With White Space

The second experiment is conducted to show that SPARSE has a capability to implicitly allocate white space whereas WL does not.

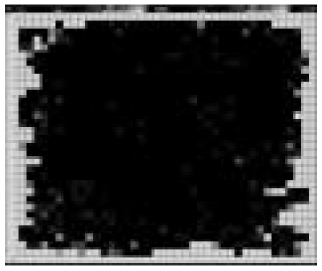

Fig. 6a:WL-based placement

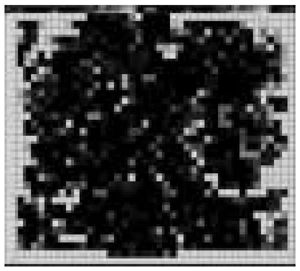

Fig. 6b: SPARSE placement

Figure 6a, 6b show the area utilization results from WL and SPARSE for IBM benchmark *ibm03*. In figure 6a we observe that WL packs blocks very densely to minimize the total wire length. In contrast, SPARSE controls implicitly the logic utilization and results in a more sparse placement as shown in figure 6b.

	total Half Bounding Box wire length		overflow		total wire length after routing	
	WL	SP	WL	SP	WL	SP
bio	5015	4960	13	0	7460	7007
ind2	20948	20991	15	0	27282	26072
ind3	40831	41033	0	0	48579	46759
avqs	15396	14763	6	0	17205	15947
ibm03	57770	56901	109	0	78623	71966
ibm04	91938	91826	56	0	100122	93246
Total	**231898**	**230474**	**199**	**0**	**279271**	**260997**

TABLE 2. SPARSE vs. WL with evenly distributed WS

To demonstrate the quality of the white space insertion, we conduct another experiment, which compares SPARSE with evenly distributed white space in WL. In other words, during the WL placement, we fix the same amount of white space into each global bin and prohibit cells to occupy it. The total white space for each benchmark is approximately 16% - 18% of the total core area.

Results in Table 3 suggest that SPARSE consistently produces placement solutions with better routability and less total wire length after routing. The white space allocation by SPARSE helps routability more than the evenly distributed white space helps WL. Examining the results in Table 2, we note that SPARSE eliminates all the overflows with the aid of the white space and produces routable designs.

We also compare our results with the recent academic placer Dragon[8]. In [8], white space is intelligently allocated based on the congestion estimates. We use the same benchmarks as in [8]. For each benchmark, the white space ratio is listed in the second column in Table 4. The experiment setup is shown in Figure 7.

We first compare SPARSE with Dragon. As in the previous experiments, we start SPARSE from the initial placement from Capo. In table 4, we show the comparison between Dragon (Dr) and Capo + SPARSE (Ca+SP). We set the routing edge capacity to a value such that the placement result by SPARSE is nearly routable, and report the routing result for the placement from Dragon. It can be seen that SP from Capo initial placement can achieve much better routability than Dragon under the same

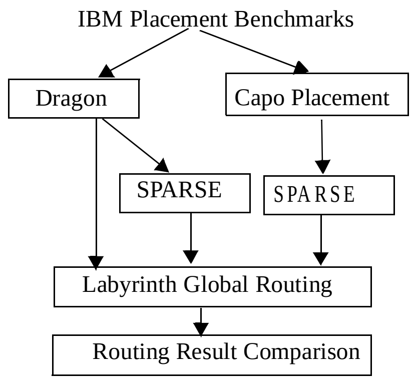

Fig. 7: SPARSE vs. Dragon

		total Bounding Box wire length			#overflow			end wire length after routing			placement CPU(m)	
	ws%	Dr	Ca+SP	Dr+SP	Dr	Ca+SP	Dr+SP	Dr	Ca+SP	Dr+SP	Dr	Ca+SP
ibm01-e	14.8%	23962	18752	18953	160	0	0	30930	22633	21076	16	x4.1
ibm01-h	12%	23560	17878	18375	275	19	0	30329	22122	20725	15	x4.2
ibm02-e	9.6%	67143	60491	62208	0	0	0	81266	72539	72188	40	x3.8
ibm02-h	4.7%	62337	60139	57471	0	45	0	77195	76973	68575	63	x2.5
ibm07-e	10.0%	95906	100777	90043	391	0	24	127847	131239	111718	42	x5.5
ibm07-h	4.7%	101377	98635	93603	55	48	0	131668	121772	104717	43	x5.4
Total		**374285**	**356672**	**340653**	**881**	**112**	**24**	**479235**	**447278**	**398999**		

TABLE 4. Congestion minimization comparison between SPARSE and Dragon[8], in the presence of white space

routing capacity constraint. The total overflow is reduced from 881 to 112. Moreover, the total wire length is reduced accordingly. For SPARSE, CPU times are increased by 2 to 5 times compared to Dragon. The run time can be improved by designing a more efficient updating of SPARSE cost function during annealing process.

As shown in figure 7, we also experimented with Dragon + SPARSE flow. Table 4 list the results from Dragon + SPARSE (Dr + SP). It can be seen that Dr + SP flow is better than Capo + SP in terms of both routability and total routed wire length. The results are consistent with those in [8] where Dragon is able to produce a placement with better routability and total wire length than Capo. When combining Dragon and SPARSE, the total overflow is reduced from 881 (by Dragon only) to 24 and the total wire length is from 479235 to 398999 (16%). These results suggests that SPARSE could be inserted into the existing place-and-route flows for both routability and wire length improvement.

7. Conclusion

In this paper, we have proposed a novel congestion minimization technique, which does not require probabilistic routing or any prior congestion estimation. Experiments show that our new technique achieves better routability and shorter total wire length than Capo[11], Dragon[8], and the traditional total-wire-length-based approach. Compared to the previous works, our technique is less layout-dependent and as a result, can provide more direct guidance for routability-driven logic optimization. Furthermore, because our new technique takes an interconnect-centric point of view, it can be smoothly integrated into a timing optimization flow.

8. Acknowledgments

The authors gratefully acknowledge support by MARCO/DARPA Giga Scale Research Center. The authors also thank Xiaojian Yang from UCLA for his help in the experiments.

References

[1] C.-L. E. Cheng. "RISA: Accurate and Efficient Placement Routability Modeling". Proc. *International Conference on Computer-Aided Design,* pp. 690-695, 1994.

[2] M. Wang, X. Yang, and M. Sarrafzadeh. "Congestion Minimization During Placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems",* vol. 19, no.10, pp.1140-1148, 2000.

[3] 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". Proc. *Asia and South Pacific Design Automation Conference,* pp. 605-608. 2001.

[4] J. Lou, S. Krishnamoorthy, and H. S. Sheng. "Estimating Routing Congestion Using Probabilistic Analysis", Proc. *International Symposium on Physical Design,* pp. 112-117, 2001.

[5] S. Mayrhofer and U. Lauther. "Congestion-driven Placement Using a New Multi-partitioning Heuristic", Proc. *International Conference on Computer-Aided Design,* pp. 332-335, 1990.

[6] P. N. Parakh, R. B. Brown, and K. A. Sakallah. "Congestion Driven Quadratic Placement", Proc. *Design Automation Conference,* pp. 275-278, 1998.

[7] X. Yang, R. Kastner, and M. Sarrafzadeh. "Congestion Reduction During Placement Based on Integer Programming", Proc. *International Conference on Computer-Aided Design,* pp.573-576, 2001.

[8] X. Yang, B-K. Choi, M. Sarrafzadeh. "Routability Driven White Space Allocation for Fixed-die Standard-cell Placement", Proc. *International Symposium on Physical Design,* pp. 42-47, 2002.

[9] A. Rohe, U. Brenner. "An Effective Congestion Driven Placement Framework", Proc. *International Symposium on Physical Design,* pp.6-11, 2002.

[10]J.M. Kleinhans, G.Sigl, F.M. Johannes and K.J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization", *IEEE Trans. CAD,* vol. 10, no.3, Mar 1991, pp.356-365.

[11]A.E. Caldwell, A.B. Kahng, I.L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?", Proc. *Design Automation Conference,* pp.477-482, 2000.

[12] G. Parthasarathy, M. Marek-Sadowska, A. Mukherjee, A. Singh, "Interconnect-Complexity Aware Placement for FPGA using Rent's Rule", *Proc. of 3rd System Level Interconnect Prediction WorkShop*, pp.23-30, April, 2001.

[13]A. Singh, G. Parthasarathy, M. Marek-Sadowska, "Interconnect-Resource Aware Placement for hierarchical FPGAs", Proc. *International Conference on Computer-Aided Design,* pp.132-136, 2001.

[14] "http://gigascale.org/bookshelf ".