An Effective Congestion Driven Placement Framework

Ulrich Brenner
Research Institute for Discrete Mathematics
University of Bonn
Germany
brenner@or.uni-bonn.de

André Rohe
Research Institute for Discrete Mathematics
University of Bonn
Germany
rohe@or.uni-bonn.de

ABSTRACT

We present a fast but reliable way to detect routing criticalities in VLSI chips. In addition, we show how this congestion estimation can be incorporated into a partitioning based placement algorithm. Different to previous approaches, we do not rerun parts of the placement algorithm or apply a post-placement optimization, but we use our congestion estimator for a dynamic avoidance of routability problems in one single run of the placement algorithm. Computational experiments on chips with up to 1,300,000 cells are presented: The framework reduces the usage of the most critical routing edges by 9.0% on average, the running time increase for the placement is about 8.7%. However, due to the smaller congestion, the running time of routing tools can be decreased drastically, so the total time for placement and (global) routing is decreased by 47% on average.

1. INTRODUCTION

During the solution of placement problems, two main goals have to be considered: First, we have to insure the routability of the nets. Second, we want to minimize the cycle time of the chip, so sets of cells that are connected by timing critical nets have to be placed near to each other.

Most of the placement algorithms try to incorporate these different goals in one optimization function which estimates the total wire-length that is necessary to route all nets. In such an approach, timing can be reflected by introducing net-weights, i.e. the minimization function is the weighted sum of the (estimated) wire-lengths where timing critical nets get a high weight.

By the minimization of the weighted wire-length it is typically possible to get good results in terms of timing; short connections are in general also advantageous for routing. However, placements with short net-lengths will often be quite dense, which can cause local routing problems. If such congested regions are detected, a new placement of the cells in these (and maybe other) regions is necessary. This can lead to several iterations of the placement phase to get a routable placement. For small designs, such iterations may be acceptable, but with the growing complexity of chips (state-of-the-art chips have several millions of movable objects) routing problems cannot be handled with this iterative method in reasonable time. Therefore, it is desirable to detect and to reduce congested regions during the placement process.

In the last couple of years several new approaches have been developed in order to take congestion during the placement into consideration. Many algorithms use a very simple version of a probabilistic global router to estimate the congestion in a region: The chip area is divided into small tiles, for each tile border the expected number of wires routed through this border is compared to the number of free routing tracks that cross the border. For example, the approaches in [3], [5], [7], [8], and [23] follow this idea. The algorithms differ mainly in the way they handle multi-terminal nets and blockages and in their probabilistic distributions for the interconnections. A probabilistic routing estimator of this type will also be an important part of our congestion estimation. Other methods to detect congestion are described in [22], where Rent's Rule is used to estimate the peak congestion value and regional congestions on a chip, and in [18], where a normal distribution of the number of nets per tile is assumed.

Most authors describe not only ideas to detect congestion, but also propose ways to reduce it. Often congestion reduction is done in detailed placement or in a post-placement-optimization (see [16], [17], [18], [19], and [23]). Other authors incorporate the goal of congestion reduction into the global placement: In [10], it is shown how quadratic placement can be modified in order to avoid routing problems; in [9], the authors describe a partitioning approach that works similar to Min-Cut partitioning, but has minimization of congestion as a goal. The partitioning based algorithm in [5] increases the estimated area consumption after a global placement for cells in congested regions and then repeats the last levels of the partitioning while considering these modified cell sizes. There have also been experiments where congestion is used as an optimization function for simulated annealing (see [3], [16], and [19]), but in [16] and [19] it is shown that this approach does not lead to an improvement of routability. The best congestion results with simulated annealing are achieved by using the netlength as the optimization function.

In this paper, we will introduce a congestion driven placement framework that avoids routing congestion during placement. We present the first complete framework which includes congestion issues into placement on industry designs with more than one million circuits. We achieved this by using of a fast, but reliable way to estimate the routing criticality of a region. Since this estimation is quite accurrate even in early stages of the placement process, we do not have to use a (local) post-placement optimization and we do not have to repeat the placement run or parts of it to reduce congestion. Therefore, our approach is able to improve routability significantly without a large increase of running time.

The rest of the paper is organised as follows: In Section 2, we give an introduction to the placement problem. Section 3 contains a description of our method to detect congested areas. In Section 4, we describe how our congestion estimator can be incorporated into a placement tool and how we can avoid routing criticalities during the placement. Experimental tests with our algorithm are described in Section 5.

2. PRELIMINARIES

2.1 Notations and Description of the Problem

We will now give a short introduction into our terminology for placement and routing. A very good overview over the field of VLSI physical design is given in the books of Sait and Youssef [11],

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

Sarrafzadeh and Wong [12], and Sherwani [14].

A *chip* consists of a set C of *cells*, a set N of *nets* and a rectangular *chip area* A. Each cell $c \in C$ is given as a rectangular box of certain size $s(c) := x(c) \times y(c)$, and each net connects a subset of the cells. The connection points for nets to the cells are called *pins*. Some of the cells may be preplaced, and some parts of A may be blocked for the placement.

The task in placement is to find a location for each cell such that all cells are completely contained in A and no two cells intersect. Since the total size of cells in state-of-the-art application specific integrated circuits (ASICs) is typically less than 80% of the placement area, it is quite easy to find a valid placement solution. The main problem is to find a placement that is routable and meets the timing constraints. (timing critical nets have to be short).

In the routing, the pins of each net have to be connected in a three-dimensional routing grid graph. Since this graph is extremely large (it can contain several billions of nodes, see Section 5), most routers work in two steps. First, the nets are connected in a very coarse version of the routing grid. The result of this *global routing* gives a quite accurate estimation of the routability of the chip; it also serves a guideline for the *local router* which generates a detailed list of wires for each net.

Both the placement and the routing problem include further constraints in practice. For example, in placement, the cells have to be placed in cell rows (to match the power grid). In routing, minimum distance constraints for wires and vias have to be considered. Other rules exist, we will not go into further detail on them.

2.2 Basic Structure of Placement Algorithms

Many placement algorithms (e.g. [6], [15], [20]) work with a recursive partitioning approach: Given a rectangular area R and a set $C(R)$ of cells, the area R is divided by a horizontal and a vertical cutline into four parts $R_1, \ldots, R_4$ of (approximately) equal size, and $C(R)$ is divided into four parts $C(R_1), \ldots, C(R_4)$ such that the cells in $C(R_i)$ fit into the area R_i (for $i = 1, \ldots, 4$). The idea is to place the cells of the set $C(R_i)$ in the region A_i with respect to certain size constraints given by the circuits $c \in C(R)$ and the regions $R_1, \ldots, R_4$. For a region R let $s(R) = f(R) \cdot d(R)$, where $f(R)$ is the free area in R (i.e. the size of R minus the total size of all blockages and preplaced cells in R) and $d(R)$ is a factor (with $0 < d(R) \leq 1$) that specifies the maximum placement density we allow in region R (often $d(R)$ is identical for all regions of the chip). So, $s(R)$ is an upper bound for the total size of movable cells that can be placed within R. Based on this, the input and output of the placement partitioning problem are defined as follows:

Input: A set of cells $C(R)$ with a size $s(c)$ for each $c \in C(R)$ and a set of four numbers $s(R_1), \ldots, s(R_4)$ with $\sum_{i=1}^{4} s(R_i) \geq \sum_{c \in C(R)} s(c)$.

Output: A partition of $C(R)$ into disjoint sets $C(R_1), \ldots, C(R_4)$ such that $\sum_{c \in C(R_i)} s(c) \leq s(R_i)$ for each $i \in \{1, \ldots, 4\}$.

Note that a solution of this problem does not necessarily exist and that it is an NP-complete problem to decide if a solution exists. So, if necessary, we relax the problem by scaling the values $s(R_i)$ linearly by a factor $\rho > 1$. Here,

$$\rho := \left(\sum_{c \in C} s(c) + 3 \max_{c \in C(R)} s(c) \right) / \left(\sum_{i=1}^{4} s(R_j) \right)$$

is sufficient to guarantee that a solution exists and can be found in linear time.

The basic idea of our placement algorithm is to call the partitioning routine in a recursive way, i.e. to start with $R := A$ and $C(R) := C$ and call the routine again for the instances $(R_i, C(R_i))$ with $i = 1, \ldots, 4$. The recursion stops when the number of cells in a region is small enough. Other authors (e.g. [2]) follow a similar approach, but just bipartition the set of cells recursively. After this *global placement*, the cells are legalised in the so-called *detailed placement* phase. In this paper, we just consider the global placement step.

Solving this problem for all regions and all sets of cells is called a *level* of the placement algorithm. Some partitioning based placement algorithms use local optimization steps after each level that allow circuits to leave the region they are currently assigned to. We will describe an example of such a local optimization (the *repartitioning*) later.

2.3 Goals

For quite a long time, minimizing the total wire-length of a chip has been the most important goal for placement algorithms. Here, the wire-length is typically measured as the sum of the lengths of a Steiner tree approximation for each net. As mentioned above, placements that are optimized with this goal tend to be quite dense and therefore to have routing problems. It might also happen that the placement is routable, but some nets have to be routed with large detours, which may affect the timing. If possible, lowering the maximum allowed placement density for the whole chip area (i.e. spreading all cells) could reduce the routability problems, but will normally lead to worse timing results. One wants to lower the maximum density $d(R)$ for routing critical regions R only, but, of course, these regions are in general not known in advance.

The main goal of this work is to describe a fast framework for the dynamic location and removal of routing problems during placement. We will see that routing problems often have a very local structure and can be avoided by a loosening of the placement in congested regions.

The only requirement for the method we describe is a partitioning algorithm as described above, but it should be noted that the ideas of our framework can be applied to other placement ideas (e.g. force directed placement [4] or simulated annealing [13],[21]) as well. The running time of the underlying placement algorithm will hardly increase with our framework; the extra work we need is a (very fast) congestion estimation and a few additional repartitioning steps (as described later).

3. CONGESTION ANALYSIS

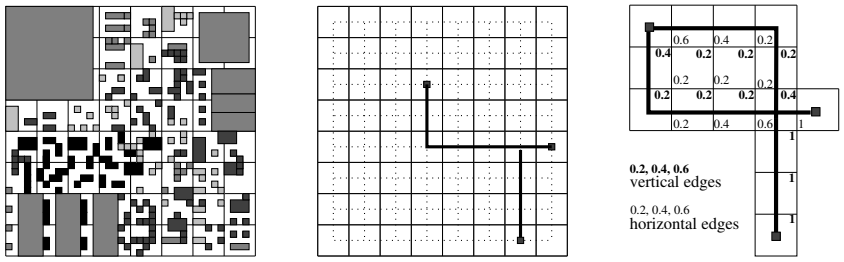
In this section we will describe a measurement for the congestion of a placement as it appears during the placement algorithm. Given a chip which is partitioned into $k \times k$ regions (forming a $(k+1) \times (k+1)$-placement-grid), we calculate a pin density for each region and a congestion estimation for each edge in the dual graph of the placement grid.

The core part of our algorithm is an estimation for the global routing of the current placement. Global routing tools divide the chip into a number of tiles (bins) and calculate a congestion estimation for the chip as well as a rough topology for each net. The main drawback of existing global routing tools is the running time. A global router using sophisticated methods based on multicommodity flows [1] might take 24h or more on a chip with 500,000 nets. Though just a congestion estimation with a global router does not take that long, it is in general too slow because we have to call such an estimator several times during the placement as a subroutine. Our simplified probabilistic global router tries to imitate a real router in the following way:

(a) The current status of the placement grid is used as the global routing partition. The global routing grid is defined as the dual of this partition (See, for example, Figure 1(a) that shows the placement grid, the dotted segments in Figure 1(b) show the routing grid).

(b) Multi-terminal nets are split into a set of two-terminal nets: We calculate a Steiner tree for all nets $n \in N$, each connection between two pins or Steiner points in this Steiner tree is treated as a separate two-point connection (see Figure 1(b)).

(c) For the two-point connections in the Steiner tree of net n, we calculate a probability $p_n(e)$ of each edge e in the dual graph of the placement grid. This method is similar to the algorithms proposed in [7] and [8]. We calculate the set P of all length-optimal paths with at most two bends (vias) between the points. In [7], also shortest connections with arbitrarily



(a) placement grid (b) Steiner tree and routing grid (c) probabilistic weights

Figure 1: A chip with the placement grid and its dual graph and the expectation values for the routing of a net.

many vias are examined (where larger numbers of vias get lower probabilities), but the restriction to two bends models quite well what a real router does. Based on this, $p_n(e)$ is set to the number of paths in P that use e divided by the cardinality of P. In Figure 1(c), three two-point connections are considered. The connections from the Steiner point to the right and the bottom terminal are uniquely optimal, we get $p_n(e) = 1$ for these edges. The connection of the left terminal with the Steiner point can be realised by five different paths with at most two vias. The usage probabilities for the vertical routing edges are shown in bold, the other probabilities with the normal font.

(d) For each edge $e \in E(G)$ in the dual of the placement grid, we calculate its expected usage $p(e) := \sum_{n \in N} p_n(e)$ and the capacity value $cap(e)$. The fraction $cong(e) := \frac{p(e)}{cap(e)}$ is our estimated congestion on e. Note that $cap(e)$ depends (via a user parameter) on the number of routing channels of the three-dimensional routing grid that cross e, but not on routing blockages due to preplaced cells. Such preplaced macro cells may block some routing layers, but in our experiments the areas covered by such macros were not routing critical and our estimation was accurrate enough without considering blockages. Of course, if it turns out to be useful, it would be possible to consider routing obstacles in our capacity estimation.

The measurement described above just considers the congestion on the edges of the (quite coarse) placement grid that is used as a global routing grid. Nets that are completely contained in one tile of the placement partition are not considered at all. In order to take also routability problems inside the tiles into account, we use the pin density $pin\text{-}dens(R)$ inside a region R as a second measurement. The number $pin\text{-}dens(R)$ is calculated straight-forward: Given the cells $C(R)$ inside the region, we divide the total number of pins in $C(R)$ by the number of routing grid nodes in R.

4. USAGE OF CONGESTION DATA

4.1 Calculation of Inflation Values

Once we have found a congested (i.e. routing critical) region during the placement process, we have to use this knowledge to remove or at least reduce the congestion in this region. We will show how our estimation can be used in the placement tool BonnPlace. BonnPlace is a partitioning based algorithm which, during a run, always stores cell positions that are computed by minimizing a quadratic function which models the wire-length. These current cell positions are also used to compute the solution in the partitioning steps. For a description of the BonnPlace algorithm, see [15]. In our approach, we follow the strategy of many chip designers and try to use a lowered density for groups of cells that tend to create routing problems. We handle such groups of cells by *inflating* them, i.e. increasing their estimated area usage in the partitioning step. This means that we do not use $s(c)$ $(= x(c) \cdot y(c))$ as the size of the cell c in this step, but a higher value $(1 + b(c)) \cdot s(c)$ (with $b(c) \geq 0$). When partitioning the set $C(R)$ (for a region R) into subsets $C(R_1), \ldots, C(R_4)$, we have to insure that the condition $\sum_{c \in C(R_i)} (1 + b(c)) \cdot s(c) \leq s(R_i)$ is fulfilled (for $i = 1, \ldots, 4$).

The numbers $b(c)$ depend on an input parameter $\tau \geq 0$ that specifies how much we want to allow the algorithm to increase the cell sizes. Before the placement starts, each cell c gets an initial value $b(c)$ that is proportional to the number of pins of c divided by $s(c)$. This is motivated by the observation that small cells with many pins often cause routing problems if they are placed densely. The initial numbers $b(c)$ are scaled with the parameter τ such that $\sum_{c \in C} b(c) \cdot s(c) = \frac{\tau}{4} \sum_{c \in C} s(c)$. This insures that the total size of the cells grows initially by a factor of $(1 + \frac{\tau}{4})$. During the placement run, the number $b(c)$ is updated according to the congestion estimation of the region R the cell c is currently placed in. Let $e_1, \ldots, e_4$ be the four edges of the global routing grid that are incident to R. If $cong(e_i) \geq 1$ we increase $b(c)$ by $\min\{1, 2(cong(e_i) - 1)\} \cdot \frac{\tau}{5}$ (for $i = 1, \ldots, 4$). This way, each of the edges can cause an increment of $b(c)$ by at most $\frac{\tau}{5}$. This maximum is attained for $cong(e_i) \geq 1.5$. The number $b(c)$ is also increased by a adding a number proportional to $pin\text{-}dens$ (but at most $\frac{\tau}{5}$), if $pin\text{-}dens$ is bigger than some fixed threshold value. Therefore, it is insured that τ is an upper bound for the total increment of $b(c)$ in each level, and that the increment due to congested global routing edges is the dominating factor. A typical value for τ is 0.2, all our experiments were made with this value.

Additionally, we also decrease the numbers $b(c)$ if *both* the congestion estimation on the routing grid edges and the pin density are far away from critical numbers. In this case, the number subtracted from $b(c)$ is also proportional to τ. This insures that unnecessarliy high values $b(c)$ (especially the initial values) can be corrected during the placement run.

4.2 Spreading inflated cells

Changing the estimated area usage for cells during the placement run would not be useful if we could not find a placement that respects these larger cell sizes. One way to find such a placement is to calculate (but not use) the inflating numbers during a first placement run and then use them in a post-placement optimization or in a new placement run (that may consist only of the last partitioning steps, see [5]). The new feature of our approach is that we do not have to repeat parts of the placement, but we use the *repartitioning* method to move cells out of regions that are too full (with respect to the sizes $(1 + b(c)) \cdot s(c)$). Repartitioning is a subroutine of BonnPlace that is called after a partitioning level and tries to find local improvements of the placement. It considers 2×2-windows, i.e. sets of four pairwise adjacent regions $R_1, \ldots, R_4$ that form a rectangle $R = R_1 \cup \cdots \cup R_4$. The subroutine computes a new partitioning for this region and new positions for the cells in the region. It accepts the new partitioning if the weighted netlength has decreased. This is done for all 2×2-windows. We repeatedly call this routine (with different orders of the 2×2-windows) as long as it yields a reasonable improvement of the wire-length. Repartitioning enables the cells to leave the region they are currently placed in. While this method was invented to improve the netlength of a placement, we use it in addition to reduce crowded regions. In our

Chip	Cells	Nets	Density	Grid-size	Rel.
ibm1	72496	73273	86.0%	$4091 \times 3563 \times 6$	2000
ibm2	72940	73822	30.9%	$6119 \times 6119 \times 5$	1998
ibm3	412505	426689	57.5%	$26792 \times 26792 \times 7$	2001
ibm4	681987	706499	57.7%	$14028 \times 13110 \times 6$	1999
ibm5	1336370	1390333	53.6%	$23912 \times 23912 \times 7$	2001

Table 1: The chips used for benchmarking.

repartitioning step, we consider regions that are too full, but have neighbour regions with some free capacity. The new partitioning is accepted if the balance of the regions is improved (i.e. the overcrowding is reduced), even if the netlength gets slightly worse. So the optimization function is a weighted sum of the maximum overcrowding of a region and the netlength. We repeat this until there is no crowded region with a neighbour region that contains a certain amount of free space.

Since we decrease the estimated area usage for cells in non-critical regions and we increase the maximal allowed density $d(R)$ for every region R in each placement level a little bit (about 1% per level), we normally have enough free capacity to move cells away from crowded regions.

5. COMPUTATIONAL RESULTS

The congestion framework we introduced in the previous sections was implemented using the placement program BonnPlace as a basis. In this section, we will evaluate the quality of the congestion estimation and the performance of the congestion driven BonnPlace. We will compare the results of standard and congestion driven BonnPlace on five different recent designs from IBM Microelectronics. The sizes of the designs are between 70,000 and 1,300,000 cells, an overview can be found in Table 1. The column "Density" shows the fraction of the total size of all cells divided by the size of the chip area. In the column "Rel." the year the chip was released is given.

In order to test the accuracy of our congestion estimation and to analyse the routability of the placements, we used the programs HDP and BonnGlobal. HDP is a tool developed by IBM to get a fast rough estimation of the routing criticality of a placement, while BonnGlobal is a global router based on a multicommodity flow algorithm [1]. All of our runs were performed on an IBM 680 with 600Mhz RS-IV processors.

5.1 Congestion analysis vs. Global Routing

As a first step, we will evaluate the quality of our congestion estimation routine on the ibm1 design by comparing the output of different global routers with our congestion estimator. In Figure 2, we show a placement produced by congestion driven BonnPlace (colored by the inflation values) and three different congestion estimations. It can be seen nicely that the very fast congestion estimation used by BonnPlace matches the output of the two global routers quite well: The main congestion problem is in the middle of the chip. All other chips of the test suite show a similar behaviour: The important congested spots are identified by our congestion estimator, sometimes the tool is a little bit too pessimistic. While the results are quite similar, the running times of the tools differ a lot: For our largest test-case ibm5, the BonnGlobal router runs nearly 30 hours, the HDP router three hours and our internal congestion estimator needs less than a minute.

5.2 Congestion Driven Placement

Finally, we will analyse the behaviour of congestion driven BonnPlace compared to standard BonnPlace.

In Table 2, we give an overview over the placement results we get on our five test-cases. Each chip was placed without and with the congestion driven framework. Despite the change to congestion driven BonnPlace, the parameters used for BonnPlace (and for the routers) were the same for the two placements on each chip. The first two columns of Table 2 show the running time (CPU) and the bounding box netlength (Length) of the standard BonnPlace runs.

Chip	Std. BonnPlace		Cong. Driven BonnPlace		
	CPU	Length	CPU	Length	Blow
ibm1	0:23 h	7.2 m	0:26 h	7.4 m	10.2%
ibm2	0:26 h	7.9 m	0:27 h	9.0 m	6.6%
ibm3	3:50 h	134 m	4:39 h	142 m	20.1%
ibm4	7:08 h	241 m	7:24 h	270 m	20.2%
ibm5	16:10 h	375 m	16:37 h	406 m	57.8%
Mean diff			+8.7%	+8.5%	

Table 2: Results of the different placement runs.

The next two columns give the same numbers for the congestion driven BonnPlace, and the last column (Blow) the increase of the cell size, i.e. the fraction of $\sum_{c \in C} b(c) \cdot s(c)$ and $\sum_{c \in C} s(c)$. We see that the value varies quite a bit on the different chips. The main reason for this is that the congestion parameters on all chips were the same although the routing criticalities of the chips differed a lot. The last row of the table contains the average differences between the standard and congestion driven runs (computed by the geometric mean): In the congestion driven version the CPU-time increases by about 8.7% and the bounding box netlength by about 8.5% on average.

Table 3 contains the results of the routing runs based on the placements of Table 2. The numbers in the columns "HDP" are the average congestions of the most congested 20% of the nets, computed by HDP. Experience shows that a value below 80 corresponds to a routable placement, values above 85 are seldomly routable. The other numbers we give are calculated with BonnGlobal. In the columns labeled with "Ov", we show the total number of overloads, "CPU" gives the running time of BonnGlobal and "Length" the sum of the Steiner trees for all nets that were calculated by BonnGlobal. Again, the last row of the table contains the average differences between the results for standard and congestion driven BonnPlace. The table shows the effect of the congestion framework:

(a) The HDP congestion value can be reduced significantly on each chip. This number is not only important for routability: With uncongested placements, it is a lot easier to achieve a routing without timing violations. It is also possible to spread the wires and increase the yield in production.

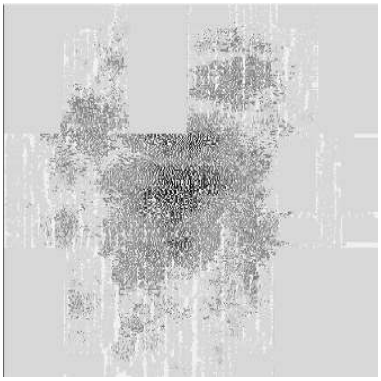
(b) All placements produce BonnGlobal overloads without the congestion framework. Figure 3 shows a comparison of the BonnGlobal congestion map on ibm1. We see a black, unroutable region in Figure 3(a) which is changed in the congestion driven run to a dark grey region in (b).

(c) The running time of BonnGlobal is reduced significantly. It should be noted that on highly congested designs, the running time of BonnGlobal explodes (see ibm3), but even without considering this test-case we see a huge CPU-time improvement in routing critical designs.

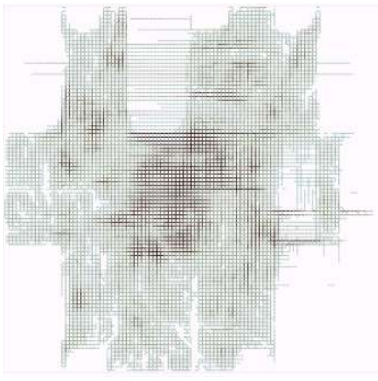
(d) Although the BonnPlace bounding box net-length increases using the congestion driven BonnPlace, the length of the Steiner trees decreases on average. This is due to the fact that in congested designs more detours are needed to reduce overloads.

To summarize, we can see that the routability of a placement can be significantly improved using the congestion driven framework. Note that, although the time for placement is increased by 8.7%, the total CPU-time for placement and global routing is decreased by about 47% on average. Additional experiments indicate that also the running time of local routing decreases using the congestion driven BonnPlace.

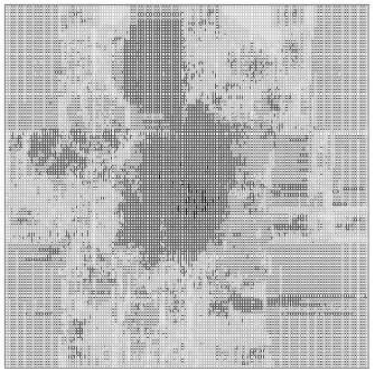
Finally, Table 4 shows that it is quite hard to achieve routable placements by just changing the parameter $d(R)$ (which is constant for all regions of the chip in our tests). On our two smallest test-cases (ibm1 and ibm2), we show the effect of a change of this parameter.



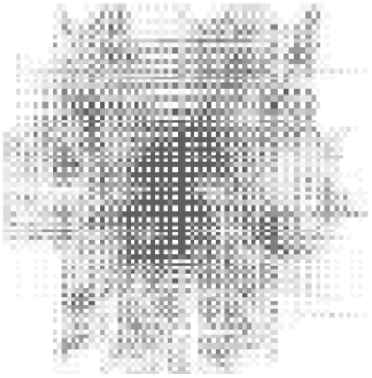
(a) placement with blow up values



(b) BonnPlace congestion estimation



(c) HDP global router



(d) BonnGlobal router

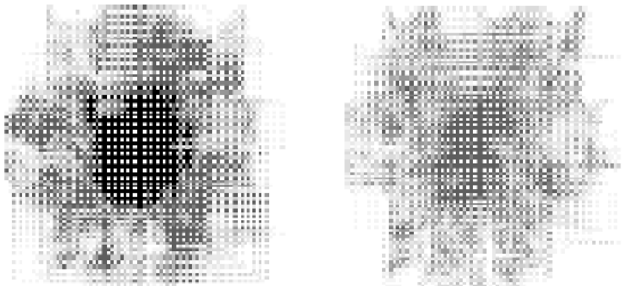
Figure 2: Comparison of global routings on a placement produced by congestion driven BonnPlace. (a) is the placement, where cells with high values of $b(c)$ are colored darker (black means $b(c) \geq 1$). (b) shows the estimation of BonnPlace, (c) and (d) the global routing calculated with HDP and BonnGlobal. Dark colors correspond here to bigger congestion. The differences between our congestion estimation and the HDP router in the upper part of the chip are due to a big preplaced cell. There, the HDP router is (because of a too conservative estimation of the routing capacities on this cell) too pessimistic, as the comparison to the more accurate BonnGlobal router shows.

Chip	Standard BonnPlace				Congestion Driven BonnPlace			
	HDP	BonnGlobal			HDP	BonnGlobal		
		Ov	CPU	Length		Ov	CPU	Length
ibm1	81.7	8374	0:15 h	9.0 m	75.4	0	0:05 h	7.5 m
ibm2	82.7	7000	0:19 h	11.5 m	75.4	0	0:05 h	10.1 m
ibm3	88.8	78111	47:36 h	162 m	77.3	0	4:51 h	164 m
ibm4	82.8	972	7:18 h	324 m	75.2	0	2:48 h	326 m
ibm5	89.9	14382	70:57 h	512 m	84.2	0	29:48 h	527 m
Mean diff					-9.0%		-73.0%	-5.2%

Table 3: Routing results for the different placements.

Chip	d(R)	Standard BonnPlace				Congestion Driven BonnPlace			
		HDP	BonnGlobal			HDP	BonnGlobal		
			Ov	CPU	Length		Ov	CPU	Length
ibm1	-10%	78.9	4463	0:19 h	8.4 m	75.5	0	0:05 h	7.5 m
	normal	81.7	8374	0:15 h	9.0 m	75.4	0	0:05 h	7.5 m
	+10%	84.1	17329	0:12 h	9.1 m	75.8	7	0:06 h	7.4 m
ibm2	-20%	75.5	0	0:05 h	10.3 m	72.6	0	0:05 h	11.8 m
	-10%	80.3	1235	0:13 h	10.9 m	73.8	0	0:04 h	10.9 m
	normal	82.7	7000	0:19 h	11.5 m	75.4	0	0:05 h	10.1 m
	+10%	87.4	19929	0:25 h	11.9 m	81.2	3358	0:15 h	11.5 m

Table 4: Effect of $d(R)$ on the chips ibm1 and ibm2.



(a) normal BonnPlace (b) congestion driven BonnPlace

Figure 3: Comparison of BonnGlobal congestion estimations on ibm1: the normal BonnPlace placement (a) is not routable (the black regions are unroutable), the congestion driven BonnPlace creates a routable placement.

The chip ibm1 has a very high density, it is impossible to achieve a routable design by just changing $d(R)$. On the other hand, the congestion driven BonnPlace either produces a routable or an almost routable placement (seven overloads remain if the $d(R)$ is increased by 10%). On ibm2, the behaviour is slightly different: Again, all congestion driven versions show a better routability than the corresponding normal BonnPlace versions. The density of ibm2 is very low, so it is possible to reduce $d(R)$ by 20%. With a 20% reduction of $d(R)$, we are able to achieve a routable design with the standard BonnPlace, but the netlength increases.

In general, one can see that with congestion driven BonnPlace it is much easier to produce a routable placement. It is hardly necessary to try different values of $d(R)$. This helps to avoid unnecessary placement runs in the early design stage.

The authors would like to thank Jürgen Koehl and Jens Vygen for their helpful comments during the development of this algorithm.

6. REFERENCES

[1] C. Albrecht. Global routing by new approximation algorithms for multicommodity flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20:622–632, 2001.

[2] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection alone produce routable placements? In *DAC-00*, pages 477–482, 2000.

[3] C.-L. E. Cheng. RISA: Accurate and efficient placement routability modeling. In *ICCAD-94*, pages 690–697. IEEE Computer Society Press, 1994.

[4] H. Eisenmann and F. Johannes. Generic global placement and floorplanning. In *DAC-98*, pages 269–274, 1998.

[5] 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 *Proceedings on the 2001 conference on Asia and South Pacific design automation*, pages 605–608. ACMPress, 2001.

[6] D. Huang and A. Kahng. Partitioning-based standard-cell global placement with an exact objective, 1997.

[7] P. Hung and M. J. Flynn. Stochastic congestion model for VLSI systems. Technical Report CSL-TR-97-737, Stanford University.

[8] J. Lou, S. Krishnamoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. In *ISPD-01*, pages 112–117. ACMPress, 2001.

[9] S. Mayrhofer and U. Lauther. Congestion-driven placement using a new multi-partitioning heuristic. In *ICCAD-90*, pages 332–335. IEEE Computer Society Press, 1990.

[10] Phiroze N. Parakh, Richard B. Brown, and Karem A. Sakallah. Congestion driven quadratic placement. In *DAC-98*, pages 275–278. ACM/IEEE, 1998.

[11] S.M. Sait and H. Youssef. *VLSI Physical Design Automation*. World Scientific, 1999.

[12] M. Sarrafzadeh and C.K. Wong. *An Introduction to VLSI Physical Design*. McGraw-Hill, 1996.

[13] C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer, 1988.

[14] N. Sherwani. *Algorithms for VLSI Physical Design Automation - 3rd Edition*. Kluwer Academic Publishers, 1998.

[15] J. Vygen. Algorithms for large-scale flat placement. In *DAC-97*, pages 746–751. ACM Press, 1997.

[16] M. Wang and M. Sarrafzadeh. On the behavior of congestion minimization during placement. In *ISPD-99*, pages 145–150. ACM Press, 1999.

[17] M. Wang and M. Sarrafzadeh. Modeling and minimization of routing congestion. In *Proceedings on the 2000 conference on Asia and South Pacific design automation*, pages 185–190. ACMPress, 2000.

[18] M. Wang, X. Yang, K. Eguro, and M. Sarrafzadeh. Multi-center congestion estimation and minimization during placement. In *ISPD-00*, pages 147–152. ACM Press, 2000.

[19] M. Wang, X. Yang, and M. Sarrafzadeh. Congestion minimization during placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19, 2000.

[20] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry designs, 2000.

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

[22] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion estimation during top-down placement. In *ISPD-01*, pages 164–170. ACMPress, 2001.

[23] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion reduction during placement with provably good approximation bound. In *ICCAD-01*, 2001.