Congestion Estimation with Buffer Planning in Floorplan Design

Chiu-Wing Sham, Wai-Chiu Wong and Evangeline F. Y. Young
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
{cwsham, wcwong2, fyyoung}@cse.cuhk.edu.hk

Abstract

In this paper, we study and implement a routability-driven floorplanner with buffer block planning. It evaluates the routability of a floorplan by computing the probability that a net will pass through each particular location of a floorplan taken into account buffer locations and routing blockages. Experimental results show that our congestion model can optimize congestion and delay (by successful buffer insertions) of a circuits better with only a slight penalty in area.

1. Introduction

Floorplanning plays an important role in physical design of VLSI circuits. It plans the shapes and locations of the modules on a chip, and the result of which will greatly affect the performance of the final circuit. In the past, area was the major concern in floorplan design. Advances in the deep sub-micron technology have brought many changes and challenges to this. As technology continues to scale down, the sizes of the transistors and modules are getting smaller and a significant portion of the circuit delay is coming from interconnects. In some advanced systems today, as much as 80% of a clock cycle is consumed by interconnects [4]. Area minimization became less important while routability and delay has become the major concern in floorplanning and many other designing steps.

Traditional floorplanners did not pay enough attention to congestion optimization. This will result in a large expansion in area or even an unroutable design failing to achieve timing closure after detailed routing. There are several previous works addressing these interconnect issues in floorplan design. In the paper [3], a floorplan is divided into grids and congestion is estimated at each grid, assuming that each wire is routed in either L-shaped or Z-shaped. Cong et al. define in their paper [9] the term *feasible region* of a net, and buffers are clustered into blocks in these feasible regions along the channel areas. Sarkar et al. [12] add in the notion of independence to feasible regions so that these regions for different buffers of a net can be computed independently. Tang and Wong [14] propose an optimal algorithm to assign buffers to buffer blocks assuming that only one buffer is needed per net. Dragan et al. [7] use a multi-commodity flow-based approach to allocate buffers to some pre-existing buffer blocks. Alpert et al. [2] make use of tile graph and dynamic programming to perform buffer block planning. Finally, Lou et al. [10] apply probabilistic analysis to estimate congestion and routability, and they show that their estimations correlate well with post-route congestion. However their congestion model does not take into account buffer insertions

Buffer insertion is one of the most popular and effective techniques [5, 13] to achieve timing closure. In current practices, buffers are inserted after routing. However buffers also take up silicon resources and cannot be inserted wherever we want. A good planning of the module positions during the floorplanning stage so that buffers can be inserted wherever needed in the later routing stages will be useful. Besides, buffer itself contributes delay and area, and their locations should be carefully planned. In our method, we will compute the buffer usage of each net and estimate congestion taking into account buffer locations and net routability. We employ a simple yet accurate method to estimate congestion from a more global and realistic perspective. The nets are assumed to be routed in multi-bend shortest Manhattan distance and issues like buffer locations and blocked nets (a net that cannot be routed in the shortest Manhattan distance) will also be considered in the evaluation of the floorplan. In order to consider buffer issues in congestion estimation, we need to compute the buffer locations for each net. Dynamic programming [6] is used to compute the buffer locations of an interconnect tree in the paper [11]. The biggest disadvantage of this method is that it will take a long computational time. In a floorplanner using the simulated annealing technique, tens of thousands of iterations will be invoked for each floorplanning step, it will be too slow if this process is repeated in each iteration. In order to speed up the process, we employ a table look-up approach to store and retrieve the delay and buffer information of a net. This allows flexible handling of the buffer information efficiently.

This paper is organized as follows. We will give an overview of our approach in Section 2. The congestion model used in our floorplanner will be discussed in section 3. The method to compute buffer locations will be described in section 4. Implementation details will be explained in section 5, and the experimental results will be shown in section 6 before the conclusion is made in the last section.

2. Overview of Our Floorplanner

We use simulated annealing with sequence pair representation to describe a non-slicing floorplan solution. Unlike traditional floorplanners, we will consider buffer usage of each net and estimate the wiring congestion of each intermediate solution taken into account the buffer requirements. In each iteration of the annealing process, we will compute the buffer requirements of each net using an efficient table look-up method that will be discussed in details in Section 4. After computing the buffer requirements of each net, we will estimate the wiring congestion of the floorplan solution considering both the module positions and the possible buffer locations. We assume that a net will be routed in multi-bend shortest Manhattan distance and buffers can only be inserted in un-occupied spaces between the logic modules. A floorplan solution will be evaluated according to the total chip area, interconnect length, congestion and availability of buffers. The congestion model and the estimation procedure will be explained in the following section in details.

3. Congestion Model

We will divide a floorplan into a 2-dimensional array of fixed-size grids. The size of each grid is a trade-off between efficiency and accuracy. In our implementation, we will use an array of 20×20 grids to 30×30 grids. In each iteration of the annealing process, we will first compute the buffer requirements of each net using a table look-up approach (section 4). These buffer requirements will be expressed as a list of grid positions. For example, a list (5, 10) means that two buffers are required for this net: the first one should be at a position of 5 grids from the source and the second one should be at a position of 10 grids from the source. After these buffer requirements are computed for each net, we will estimate the congestion and routability of a floorplan solution as follows.

3.1. Construction of Grid Structure and Blocked Grids

In order to obtain the congestion information at every location of a floorplan, we divide the floorplan into a 2-dimensional array of fixed-size grids. In each iteration of the annealing process, we will compute the congestion information at each grid assuming that a net will be routed in a multi-bend shortest Manhattan distance. Since buffers, which also consume area and cannot be inserted wherever we want, may need to be inserted on a net in order to satisfy its delay bound, we should consider the possible buffer locations in the congestion estimation. In order to handle this and other routing constraints like VCC, GND and CLK, we define some grids as *blocked grids*. A grid is blocked for net k if it is located at a pre-defined reserved area, or at a position where buffer should be inserted for net k but is totally occupied by some logic modules.

A simple example is shown in figure 1. In this example, we assume that the buffer requirements of net k is (3, 6), i.e., two buffers are needed and they should be at distances of three and six grids from the source respectively. The shaded grids are the grids where a buffer should be inserted for net k if the wire passes through it. However two of them are totally occupied by logic modules and cannot accommodate any buffer. These two grids are thus blocked. Besides, the two grids of dark grey in color are also blocked because they are reserved for some other routing purposes.

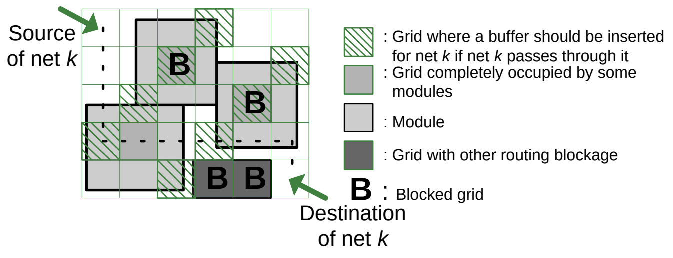
Source of net k
B
B
B B
Destination of net k
▨ : Grid where a buffer should be inserted for net k if net k passes through it
: Grid completely occupied by some modules
: Module
: Grid with other routing blockage
B : Blocked grid

Figure 1. Example of Blocked Grids

3.2. Counting the Number of Routes at a Grid

After dividing a floorplan into grids, we will count the number of routes of a net passing through each grid. In our floorplanner, we assume multi-bend routing with shortest Manhattan distance. An example is illustrated in figure 2.

In this example, we assume that the starting grid is on the upper-left side of the ending grid. At the beginning (figure $2(a)$), the number of possible routes at each grid lying vertically or horizontally with the starting grid is initialized to one. Then, the values of the other grids can be computed

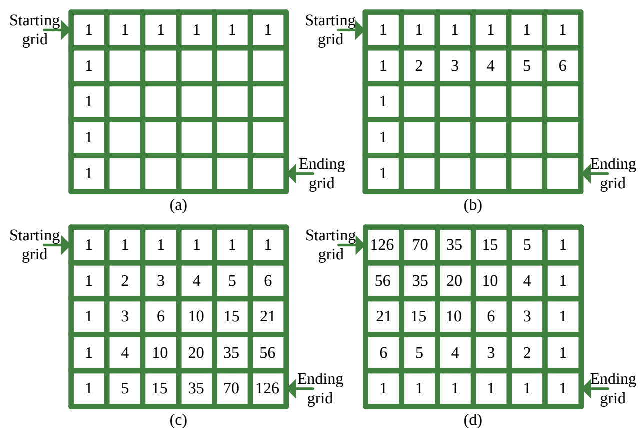

Figure 2. Example for counting the number of routes for a net

dynamically row by row by adding up the values on its top and on its left. In general, the value $g_1(x,y,k)$ computed at the grid (x,y) is the number of possible routes for net k that goes from the starting grid to the grid at position (x,y). We can then obtain the total number of possible routes from the starting grid to the ending grid by filling in the values at all the grids. In order to compute the number of routes passing through each grid, we need to repeat the same steps from the ending grid to the starting grid (figure 2(d)) and compute $g_2(x,y,k)$, that is the number of possible routes for net k going from the ending grid to the grid at position (x,y). The number of routes $g(x,y,k)$ for net k passing through the grids at (x,y) will then be computed as:

$$g(x,y,k) = g_1(x,y,k) \times g_2(x,y,k)$$

3.3. Counting routes with Blocked Grids

A blocked grid for net k does not allow the wire of net k to pass through, so the counting should be reset to zero at the blocked grids. An example is illustrated in figure 3.

By comparing figure 2 and figure 3, we can see that the total number of possible routes is reduced from 126 to 48 because of the lack of routing resources at the blocked grids. It may happen that the number of possible route is equal to zero. It means that the net cannot be routed from the starting grid to the ending grid with the shortest Manhattan distance, and we will call such a net a *blocked net*.

3.4. Delay of a Blocked Net

If a net is blocked, it cannot be routed in its shortest Manhattan distance without omitting some of its required

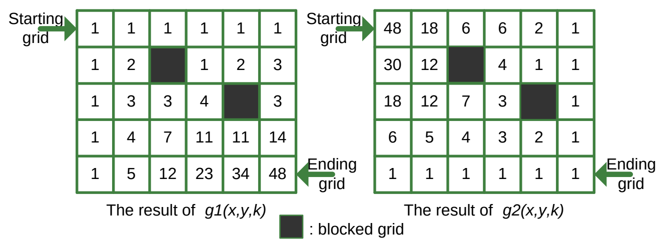

Figure 3. Results of g_1*(x,y,k)* and g_2*(x,y,k)* with blocked grids

buffers. In our modeling, it is reasonable to assume that buffers will still be inserted for a blocked net as much as possible to minimize its delay. There is a simple way to compute the maximum number of buffers that can be inserted for a blocked net using our computational scheme. A simple example is shown in figure 4.

In this example, net k is a blocked net that requires x=2 buffers. In order to compute the maximum number of buffers that can be inserted for net k, we will first label the starting grid with x, meaning that x buffers are required to be inserted in any route from that grid to the ending grid. Then we will compute the values at other grids by taking the minimum between the values at its top and on its left. A smaller value is taken because we want to satisfy the buffer requirements as much as possible. Whenever we reach an un-blocked grid where a buffer is required, we will take the minimum value as usual and minus one from it. This is because a buffer can be inserted at that position and the number of buffers required can be reduced by one. This computation is continued until the ending grid is reached. If the value y at the ending grid is zero, it means that all the buffers required can be inserted (but this will not happen for a blocked net). Otherwise, the value $(x - y)$ will be the maximum number of buffers that can be inserted for net k. This value $(x - y)$ will then be used to compute the optimized delay of net k.

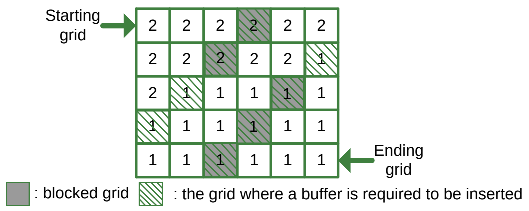

Figure 4. Example of computing the maximum number of buffers for a blocked net

3.5. Computing the Probability of Net Crossing

After counting the number of possible routes for a net, we can calculate the probability that the wire of a net will pass through any particular grid. The probability that the wire of a net k will pass through the grid at (x, y) is:

$$P(x, y, k) = \frac{g_1(x, y, k) \times g_2(x, y, k)}{g_1(x_0, y_0, k)}$$

where $g_1(x_0, y_0, k)$ is the total number of possible routes from the starting grid to the ending grid for net k. An example for figure 3 is shown below:

$$\begin{aligned} & P(x_0 + 4, y_0 - 3, k) \\ = & \frac{g_1(x0 + 4, y0 - 3, k) \times g_2(x_0 + 4, y_0 - 3, k)}{g_1(x_0, y_0, k)} \\ = & \frac{11 \times 2}{48} = 0.458333 \end{aligned}$$

By using the above method, we can compute the congestion at each grid due to each net one after another. Finally, the average number of wire passing through the grid at (x, y) will be computed, taking into account all the nets simultaneously, as:

$$weight(x, y) = \sum_{net\ k} P(x, y, k)$$

If the value of $weight(x, y)$ is large, it means that the grid at (x, y) is very congested.

3.6. Time Complexity

In the congestion estimation, we need to scan the grids from the source to the destination for each net. Therefore the time complexity is $O(m \times k)$ where k is number of grids scanned for each net and m is number of nets. However, the number of grids that required to be scanned for each net is bounded because we assume that the nets are routed in the shortest Manhattan distance. As a result, if the floorplan is divided into K grids, we need to scan at most K grids for each net and the time complexity of the congestion estimation is only $O(m)$.

4. Buffer Location Computation

Since the floorplan solution is divided into grid structures for congestion estimation, we will use the grid unit length as the length of a wire segment in the computation of buffer locations. Instead of computing the buffer locations by dynamic programming [6], we have developed a table lookup approach to speed up the computations of buffer locations.

4.1. Elmore Delay Computation

The Elmore Delay Model is commonly used for delay estimation. For a wire divided into n segments and with a load capacitance C_L and a load resistance R_L. If buffers are inserted in each segment, its delay can be computed by the following formula:

$$\begin{aligned} D = \sum_{k=1}^{n} [& R_{Bk-1}(c_0 l + c_f l + C_{Bk}) \\ & + r_0 l(\frac{c_0 l}{2} + \frac{c_f l}{2} + C_{Bk}) + d_{Bk}] \end{aligned}$$

where c_0, r_0 and c_f are unit wire capacitance, resistance and fringing capacitance respectively; $R_{B0} = R_L$, $C_{Bn} = C_L$ and $d_{Bn} = 0$; l is the length of a wire segment, R_B, C_B and d_B are resistance, capacitance and intrinsic delay of a buffer respectively.

4.2. Dynamic Programming Approach

A dynamic programming algorithm is used for interconnect tree optimization in the paper [11]. The algorithm adopts a bottom-up dynamic programming approach to minimize the maximum delay among all the paths from a source to the sinks. The interconnect tree is divided into short segments, and in each step of the bottom-up traversal of the tree, a set of solutions for each tree segment is computed based on the solution sets of its subtrees.

However, the biggest disadvantage of this approach is its long computation time. Its execution time will exponentially increase with the number of wire segments. If we estimate the delay of each wire in each iteration of the annealing process, the whole floorplanning process will take a very long time. In order to avoid this, we use the same approach, but will save the necessary information into a table before the annealing process. During the annealing process, all the buffer insertion information can then be "looked-up" from the table quickly. The computation time for buffer locations can thus be shortened significantly.

4.3. Table Lookup Approach

In order to speed up the computation time for buffer locations, we have developed a table lookup approach. Before the simulated annealing process, we compute the buffer locations (optimal delay) for each different wire length (in terms of grid unit) once. Since the wire length is estimated by its shortest Manhattan distance, there should be at most 60 grid unit length for each net (20-30 grid unit length in each dimension of the whole chip). The information stored in the table for each net length is the optimal delay of the wire, number of buffers required and their corresponding locations. The net length is used as an index for looking up the information in the table. The table structure is shown in figure 5.

index	*information*		
Wirelength (grid unit)	Optimized wire delay (fs)	Number of buffers requried	Buffer locations
1	55.6231	0	Null
2	78.4569	0	Null
⋮		⋮	
60	15646.002	4	12,24,36,48

Figure 5. An example of a lookup table for buffer location computation

5. Implementation Details

5.1. Handling Multi-pin Nets

The above calculations are used for 2-pin nets only. In order to extend it to handle multi-pin nets, there are several methods such as using minimum spanning tree (MST) or rectilinear steiner tree (RST). MST will run faster but it may over-estimate the congestion because of the overlapping net segments. However, this conservative estimation will not affect the resultant packing significantly because the total length of an MST can be reduced by 6% to 9% only by removing all the overlapping net segments to get a corresponding RST [8]. Since the run time of an RST algorithm is usually much longer than that of an MST algorithm, MST will be a better choice for estimation purposes in the early floorplanning stage. As a result, we apply MST to handle multi-pin nets in our floorplanner: breaking the multi-pin nets into a set of 2-pin nets.

5.2. Positioning of the I/O pins

During the floorplanning stage, the positions of the I/O pins are not fixed, so we need to locate the I/O pins before counting the number of possible routes. In order to distribute the I/O pins into the grids appropriately, intersection-to-intersection method is used. Consider a net connecting two modules A and B, we will first draw a line connecting the centers of two modules. The two intersecting points between this line and the boundaries of the modules will be found (figure 6) and the I/O pins will be placed into the grids of the intersection points.

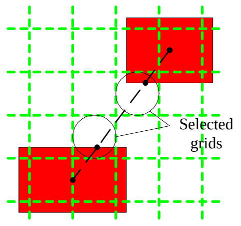

Figure 6. Intersection-to-intersection method to locate I/O pins

5.3. Cost function of the Floorplanner

In our floorplanner, we use the following cost function:

$$Cost = Area + \beta \times Wire + \lambda \times Max\_weight$$
$$+\delta \times Blocked\_wire$$

where $Area$ is the total area of the floorplan, $Wire$ is the total wirelength, Max_weight is the average weight of the top 10% most congested grids and $Blocked\_wire$ is the number of blocked wires, β, λ and δ are parameters. These parameters will be set at the beginning of the annealing process in such a way that the ratio of importance of the area term, the wirelength term, the congestion term and the blocked wire term will remain a constant. This can be done by performing a sequence of random walks at the beginning and sampling the average values of these penalty terms. The parameters β, λ and δ can then be computed accordingly.

6. Experimental Results

We tested our floorplanner using three MCNC building block benchmarks (ami33, ami49 and playout). The number of modules and nets are (33, 123), (49, 408) and (62, 1161) respectively. The areas of the modules are scaled by 10 times in order to demonstrate the effects of buffer insertions on delay. All the experiments were performed using an Intel 1.4GHz processor with 256 MB memory. The values of the parameters in the Elmore delay computation are based on the 0.18 μm technology (see Table 1) [1, 13].

We tested our floorplanner using the benchmarks with three cases: (a) without congestion nor buffer insertion, (b) with congestion optimization only, and (c) with both congestion and buffer insertion. The execution time, deadspace percentage, number of blocked nets, congestion and net delays are reported in Table 2. For congestion and net delays, we report the average of the top 10% most congested grids and the average of the top 10% longest delays. Every set of

Parameters	Value
wire resistance per unit length (Ω/μm)	0.075
wire capacitance per unit length (fF/μm)	0.118
wire fringing capacitance per unit length (fF/μm)	0.0641
intrinsic buffer delay (ps)	36.4
load capacitance/buffer capacitance (fF)	23.4
driver resistance/buffer resistance (Ω)	180

Table 1. Values of parameters used in Elmore delay computation

results displayed in the table is the average obtained by performing the experiment eight times. From Table 2, we can observe that there is about 10-25% decrease in net delay and up to 95% decrease in the number of blocked nets in case (c) (with both congestion and buffer insertion) in comparison with the other two cases. For congestion, we can obtain an improvement of about 20-35% in both case (b) and (c) in comparison with case (a) (without congestion nor buffer insertion). Case (a) optimizes the area the best, but it is the worst in terms of the number of blocked nets and congestion. In the other cases, there are significant improvements in reducing the number of blocked nets, congestion and net delay, with only a slight penalty in area (about 3% on average).

Circuit and case	Execution time(s)	Dead-Space (%)	Number of blocked nets	Congestion (number of nets per grid)	Net delay (fs)
ami33					
(a)	3.415	5.407	4.750	11.521	51.118
(b)	238.645	9.221	3.625	8.251	43.829
(c)	243.508	10.977	1.250	7.478	38.497
ami49					
(a)	8.117	5.242	44.875	65.196	485.424
(b)	221.371	6.446	42.875	51.389	502.183
(c)	254.110	6.396	8.750	59.470	450.725
playout					
(a)	19.881	6.282	107.625	153.849	59.603
(b)	1034.891	8.507	98.875	104.803	57.147
(c)	1074.447	8.543	4.750	109.713	51.922

Table 2. Experimental Results for MCNC benchmark

For the execution time, it can be seen that the time for handling buffer insertions using the table lookup approach is very efficient (increase of about 6% on average) by comparing the execution time of case (b) and (c).

7. Conclusion

In this paper, we presented a congestion model for estimating interconnect cost with buffer planning in a routability driven floorplanner. By using this probabilistic model, the number of blocked nets, delay and congestion can be estimated, taken into account buffer locations and routing blockages. Experimental results show that congestion and delay can be better optimized using this congestion model with little penalty in area.

References

[1] S. I. Assoication. *National Technology Roadmap for Semiconductors*. San Jose, CA, 1997.

[2] S. S. S. C. J. Alpert, J. Hu and P. G. Villarrubia. A practical methodology for early buffer and wire resource allocation. In *DAC*, 2001.

[3] H. M. Chen, H. Zhou, F. Y. Young, D. Wong, H. H. Yang, and N. Sherwani. Integrated floorplanning and interconnect planning. In *Int. Conf. on CAD*, pages 354–357, 1999.

[4] J. Cong. Challenges and opportunities for design innovations in nanometer technologies. In *SRC Design Sciences Concept Paper*, 1997.

[5] J. Cong, T. Kong, and D. Z. Pan. Buffer block planning for interconnect-driven floorplanning. In *Int. Conf. on Computer Aided Design*, pages 358–363, 1999.

[6] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts 02142, 1990.

[7] S. M. F. F. Dragan, A. B. Kahng and A. Zelikovsky. Provably good global buffering using an available buffer block plan. In *Proc. Int. Conf. On CAD*, 2000.

[8] J. M. Ho, G. Vijayan, and C. K. Wong. A new approach to the rectilinear steiner tree problem. In *26th ACM/IEEE DAC*, pages 161–166, 1989.

[9] T. K. J. Cong and D. Pan. Buffer block planning for interconnect driven floorplanning. In *CAD 1999. Digest of Technical Papers. 1999 IEEE/ACM Int. Conf.*, pages 358–363, 1999.

[10] S. K. J. Lou and H. S. Sheng. Estimating routing congestion using probabilistic analysis. In *Int. Sym. on Physical Design, ACM*, pages 112–117, April 2001.

[11] J. Lillis, C.-K. Cheng, and T.-T. Y. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE Journal of Solid-State Circuit*, 31(3):161–166, March 1989.

[12] C. K. K. P. Sarkar, V. Sundararaman. Routability-driven repeater block planning for interconnect-centric floorplanning. In *ISPD 2000*, 2000.

[13] P. Sarkar, V. Sundararaman, and C.-K. Koh. Routability-driven repeater block planning for interconnect-centric floorplanning. In *Int. Symp. Physical Design*, pages 186–191, 2000.

[14] X. P. Tang and D. Wong. Planning buffer locations by network flows. In *Intl. Symp. Physical Design*, pages 186–191, 2000.