Fast Buffer Planning and Congestion Optimization in Interconnect-driven Floorplanning

Keith W.C. Wong and Evangeline F.Y. Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
E-mail: wcwong2,fyyoung@cse.cuhk.edu.hk

Abstract— **In this paper, we study and implement a routability-driven floorplanner with congestion estimation and buffer block planning. We assume that buffers should be inserted at flexible intervals from each other for long enough wires. Under this buffer insertion constraint, our floorplanner will estimate congestion by computing the best possible buffer locations for each net and perform probabilistic analysis based on the solution. Dynamic programming is used such that estimations can be done very effectively. Nets are topologically grouped to consider bus-based routing and to facilitate the estimation process. We compare our results with those in paper [16] which are the latest results for this problem, and show that our approach can perform better in both quality and runtime.**

I. 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 transistors and modules are getting smaller and a significant portion of circuit delay is coming from interconnects. In some advanced systems today, as much as 80% of the clock cycle is consumed by interconnects [6]. Area minimization has become less important while routability became 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 [5], a floorplan is divided into grids and congestion is estimated at each grid, assuming that each wire is routed in either L-shape or Z-shape. Cong et al. define in their paper [7] the term feasible region of a net, and buffers are clustered into blocks in these feasible regions along the channel areas. Sarkar et al. [15] 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 [17] propose an optimal algorithm to assign buffers to buffer blocks assuming that only one buffer is needed per net. Dragan et al. [9] 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. [11] 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 insertion.

Buffer insertion is one of the most popular and effective techniques [6] 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 floorplanner, we will compute the best possible buffer locations for each net and estimate congestion based on the solution. We adopted the *variable interval buffer insertion constraint* introduced in the paper [16], i.e., buffers are constrained to be inserted for long enough wires such that the distance between adjacent buffers is lying within a range $[L, U]$ given by the user. This constraint in buffer location provides flexibility for the later routing stage and allows users to specify their constraints accordingly. The probabilistic approach in paper [16] is not efficient enough to be useful practically. In our work, dynamic programming is used to compute the buffer locations of a net, then probabilistic analysis is performed to estimate congestion. To further improve the efficiency of our floorplanner and to consider bus-based routing, we employ a net grouping technique. Grouping of nets with related topology and circuit properties into buses can improve the efficiency of the design process and allow faster convergence of solution. We compare our results with those in paper [16] and can show that our approach in congestion estimation is more accurate and efficient.

The paper is organized as follows. We will formulate the problem in section II and give an overview of our approach in section III. The method for congestion estimation will be discussed in section IV. An approach to compute the bounds in the buffer insertion constraint will be suggested in section V. Our net grouping approach will be explained in section VI, with the floorplanner implementation details follow in section VII. Finally, experimental results will be shown in section VIII before a conclusion and some discussions are given in the last section.

II. Problem Formulation

We assume that wires are routed over-the-cell and buffers can only be inserted in un-occupied spaces between the logic modules. Given a lower and upper bound $[L,\ U]$ for the variable interval buffer insertion constraint, a set of m nets and a set of n modules where each module M_i has an area A_i and a lower and upper aspect ratio bound $[r_i, s_i]$, we want to obtain a non-overlap packing of these modules such that the area of the packing, the interconnect cost and the congestion cost are small, every net satisfies its buffer insertion constraint and every module satisfies its area and aspect ratio constraint.

III. Floorplanner Overview

We use the stochastic technique of simulated annealing with sequence pair representation [13] for our floorplanner. Unlike traditional floorplanners, we will consider the buffer requirements of each net and estimate the wiring congestion in the evaluation of each intermediate solution. In each iteration of the annealing process, we will select the best possible buffer locations for each net using dynamic programming and estimate the wiring congestion between adjacent pairs of selected buffers by probabilistic analysis. We assume that a net will be routed in its shortest Manhattan distance with multi-bends. A floorplan solution will be evaluated according to the total chip area, interconnect length, congestion and number of *blocked nets* (a net that cannot be routed in its shortest Manhattan distance while satisfying all the buffer requirements). The congestion model and estimation process will be explained in the following sections in details.

IV. Congestion Estimation

We divide a floorplan into a 2-dimensional array of fixed-size grids. By dividing the floorplan into grids, the congestion information at every location of the whole floorplan can be obtained. The congestion estimation process is performed net by net. For each net i, we will select the best possible buffer locations that satisfy the buffer insertion constraint. This buffer selection process can be done efficiently by dynamic programming. After computing the buffer locations for a net, we will estimate the congestion due to this net by considering all the source-buffer pair, buffer-buffer pairs and buffer-sink pair along the route. The congestion information at each grid will be updated, which will affect the buffer insertion process of the other nets. The whole process is repeated until all the nets are routed and analyzed. An example is shown in figure 1. (The coordinates shown in the grids are the positions of the previous buffers if a buffer is inserted in that grid.) Suppose that we are now routing a net from the source at (0, 6) to the sink at (8, 0). The dynamic programming procedure will be invoked to compute the best possible buffer locations. Suppose that the buffers at locations (5, 2) and (4, 5) are selected. We will compute the congestion caused by the connections from the source at (0, 6) to the first buffer at (4, 5), from the first buffer at (4, 5) to the second buffer at (5, 2) and from the second buffer at (5, 2) to the sink at (8, 0).

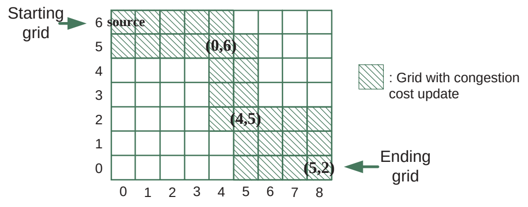

Fig. 1. Congestion estimation after buffer locations of a net are determined.

A. Computation of Buffer Locations

In our floorplanner, we assume that every net must satisfy the variable interval buffer insertion constraint, i.e., buffers are constrained to be inserted for long enough wires such that the distance between adjacent buffers is lying within a range $[L,\ U]$ where L and U are given by the user. The computation of buffer locations must satisfy these constraints. This can be achieved by dynamic programming by scanning the grids lying within the rectangle bounded by the source and the sink one by one from the source to the sink. At each grid location (x, y), we will check whether (x, y) is a feasible buffer position. If (x, y) is a feasible buffer location, we will compute the location of its best possible previous buffer assuming that a buffer will be inserted at (x, y). When we reach the sink, we will be able to backtrack the sequence of the best possible buffer locations from the sink to the source.

A.1 Cost of Grids for Buffer Insertion

In order to find the best possible buffer locations, we need to define the availability of a grid for buffer insertion. This is computed based on the routability (congestion) of that grid, the amount of empty space in it and the number of buffers already inserted there. For the grid at (x, y), it is computed as:

$$\begin{aligned} resource(x,y) &= p_1 \times (\text{congestion at } (x,y)) + \\ & \quad p_2 \times \frac{\text{no. of buffers inserted in } (x,y)}{\text{max. no. of buffers allowed in } (x,y)} \end{aligned} \tag{1}$$

where p_1 and p_2 are parameters for adjusting the importance of the buffer resources and the routing congestion in the cost function. Then we can define the cost of inserting a buffer at grid (x, y) as:

$$cost(x,y) = resource(x,y) + \min_{(a,b) \in R(x,y)} (cost(a,b)) \tag{2}$$

where $R(x, y)$ is the set of grids (a, b) such that (a, b) is at a distance d from (x, y) where $d\epsilon[L,\ U]$ and (a, b) is lying on a path of shortest Manhattan distance from the source to (x, y). We do not consider path congestion in the selection of buffer locations here because path congestion will be taken into account in the cost function of the annealing process. Notice that when a grid is totally covered by some logic modules or is reserved for some other routing purposes like VCC, GND and CLK, the space for buffers in that grid will be zero and will never be chosen as a buffer location. These grids are called *blocked grid* and an illustration is shown in figure 2.

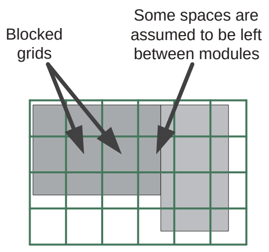

Fig. 2. Illustration of blocked grids.

A.2 Dynamic Programming

The buffer locations are computed by dynamic programming. Given the source and sink of a net, we will visit the grids in the bounding rectangle of the net one by one from the source to the sink. At each grid (x, y), we will first check whether the grid (x, y) can be a feasible buffer location for this net. If grid (x, y) is a feasible location for buffer insertion, we will compute its cost, $cost(x, y)$, according to equation 2. Notice that the cost of any grid (a, b) in $R(x, y)$ should have already been computed when we reach the grid at (x, y) and we can thus compute cost(x, y) immediately. By computing $cost(x, y)$, we are indeed finding out the location of the best possible previous buffer if a buffer is inserted at (x, y). This best previous buffer location will be computed and stored at (x, y) and will be used to backtrack the sequence of buffer locations at the end when the sink is reached. An example is illustrated in figure 3. In this example, we assume that the source (0,6) is on the upper-left side of the sink (8,0) and the bound in the variable interval buffer insertion constraint is [3, 5]. When we reach the grid at (5,2), the shaded area is the feasible region for the previous buffer location, i.e., the region $R(5, 2)$.

A.3 An Example

An example based on the case in figure 1 is shown in figure 4. In figure 4, suppose that the scanning process in the dynamic programming reaches the grid at (5,2). The numbers in the grids of the shaded region are the costs of the grids (INF means a very large cost value). Those grids with larger vacant area, fewer buffers inserted and less congested routing will have lower costs. We will first check whether buffer insertion is allowed at (5,2). Since the shortest Manhattan distance from the source to (5,2) is 10 grid units, buffer is allowed to be inserted (by table lookup in our implementation). Then, we

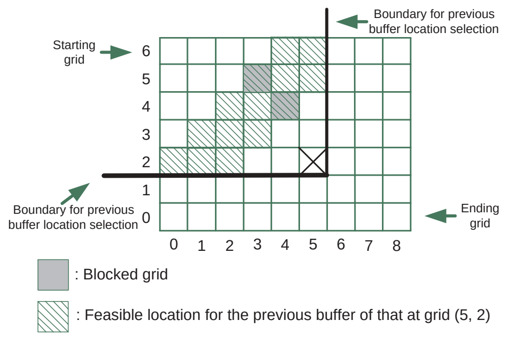
Starting grid
Boundary for previous buffer location selection
Boundary for previous buffer location selection
Ending grid
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8
: Blocked grid
: Feasible location for the previous buffer of that at grid (5, 2)

Fig. 3. Feasible locations for the previous buffer of that at grid (5,2) when $[L,\ U] = [3,\ 5]$.

will compute $cost(5, 2)$ and the optimal location of the previous buffer if a buffer is inserted at (5,2). The number shown in grid (5,2) is an initial value obtained by dividing the number of buffers already inserted there by the maximum number of buffers allowed. To compute $cost(5, 2)$, we can scan the grids in the shaded region $R(5, 2)$ one by one. We start the scanning process from the furthest grid. The first grid scanned is (4,6) and the cost of (5,2) is updated to 20+10=30. We will keep the lowest cost and the corresponding previous buffer location. When the scan reaches the grid at (2,4), the cost can be lower to 5+10=15 and thus (4,6) is replaced by (2,4) as the best previous buffer location. After all the grids are scanned, we can obtain (2,2) as the optimal location of the previous buffer and the cost of (5,2) is updated to 3+10 = 13.

Once we reach the sink, we can backtrack the whole sequence of buffer locations for this particular net. First, we can obtain the best previous buffer location (x_1, y_1) for the sink. Then we can obtain the best previous buffer location (x_2, y_2) for (x_1, y_1). In this way, we can find out all the buffer locations recursively until the source is reached. For those grids with buffers inserted, the maximum number of buffers allowed will be reduced by one, and the number of buffers already inserted will be incremented by one. Nets that cannot reach the source from the sink during the backtracking phase are counted as blocked nets.

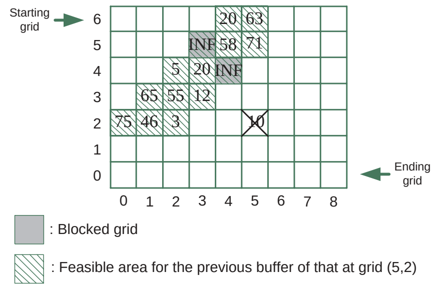

Fig. 4. Selection of the optimal previous buffer location for grid (5,2) when $[L,\ U] = [3,\ 5]$.

A.4 Congestion Estimation

After computing the buffer locations of a net, we will compute the congestion information by breaking the net into a set of sub-nets consisting of all the source-buffer pair, buffer-buffer pairs and the buffer-sink pair. For each sub-net, we assume that every multi-bend route of shortest Manhattan distance is feasible and the congestion at each grid $(x,\ y)$ due to a sub-net k is computed as:

$$\begin{aligned} & congestion(x,y,k) \\ & = \frac{\text{no. of possible routes for } k \text{ passing } (x,y)}{\text{total no. of possible routes for } k} \end{aligned} \quad (3)$$

We will then sum up the congestion due to each sub-net in the circuit to obtain the congestion information at grid (x,y):

$$congestion(x,y) = \sum_{\text{sub-net } k} congestion(x,y,k) \quad (4)$$

V. Buffer Location Bound

In our floorplanner, the bound $[L,\ U]$ in the variable interval buffer insertion constraint can be input by the user. In our current implementation, we compute these values based on the Elmore delay model. If a wire with a load capacitance C_L and a load resistance R_L is divided into n segments, and a buffer is inserted between each pair of adjacent segments, the delay of the wire can be expressed as:

$$\begin{aligned} D &= \sum_{k=1}^{n} [R_{B_{k-1}}(c_0 l_k + c_f l_k + C_{B_k}) + \\ & \quad r_0 l_k (\frac{c_0 l_k}{2} + \frac{c_f l_k}{2} + C_{B_k}) + d_B] \end{aligned} \quad (5)$$

where c_0, r_0 and c_f are unit wire capacitance, resistance and fringing capacitance respectively, l_k is the length of wire segment k, R_B, C_B and d_B are resistance, capacitance and intrinsic delay of a buffer respectively. The paper [3] suggested that for a two-pin net, the distance between any two adjacent buffers should be equal in order to optimize the wire delay. To simplify the computation, we assume that the driver resistance $R_D = R_B$ and the load capacitance $C_L = C_B$, and the optimal delay occurs when buffers are evenly distributed in a wire of uniform wire width. To obtain the upper bound value U, we need to compute the length of a wire segment such that the delay when no buffer is inserted (D_{nb}) will be larger than that when a buffer is inserted at the middle of the wire (D_{wb}).

Mathematically, we have

$$D_{nb} \quad > \quad D_{wb}$$

Rewrite into Elmore delay expressions, we have

$$l \quad > \quad \sqrt{\frac{4(R_B C_B + d_B)}{r_0(c_0 + c_f)}} \quad (6)$$

The above computation implies that it is worthwhile to insert a buffer if the wire segment length l is longer than $\sqrt{\frac{4(R_B C_B + d_B)}{r_0(c_0+c_f)}}$. Since the delay without buffer insertion is smaller if the wire length is less than U, we can set

$$U = \left\lfloor \sqrt{\frac{4(R_B C_B + d_B)}{r_0(c_0+c_f)}} / d \right\rfloor \text{ grid units.}$$

where d is the grid unit length. A buffer will be inserted in the middle if the wire segment is longer than U. Lower bound L is computed as

$$L = \left\lceil \frac{1}{2} \sqrt{\frac{4(R_B C_B + d_B)}{r_0(c_0+c_f)}} / d \right\rceil \text{ grid units.}$$

For the source and the sink, since we assume that $R_D = R_B$ and $C_B = C_L$, the wire segment from the source or to the sink can be treated in the same way as the other wire segments.

VI. Net Grouping

In most floorplanners, routability is measured with net-based models. The continuous decrease in feature size has allowed more and more transistors to be fabricated on one single chip. An enormous amount of communication between different components has resulted. Grouping of nets with related topology and circuit properties into buses can improve the efficiency of the design process and allow faster convergence of solution. In additions, it is required in some designs to have certain groups of nets to have the same electrical characteristics and bus-based design can handle these requirements more effectively [14].

In order to consider bus-based routing, we developed an efficient net grouping approach in our floorplanner. A bus is formed by bundling together the nets that start and end at the same grid. This approach has led to a significant reduction in the complexity of the floorplanning process since a much smaller number of net groups will be resulted. Our grouping method is divided into two levels. In the first level, we group the nets that connect the same set of modules together (figure 5). After the first stage grouping, multi-pin nets are then decomposed into sets of two-pin nets by the minimum spanning tree (MST) approach. The first level grouping can avoid repetitive decomposition of the same multi-pin net in the annealing process. Each two-pin net is then represented by a pair of coordinates of its source and sink. The second level grouping will group together all those two-pin nets having the same pair of source and sink coordinates.

In the computation of congestion information, we will route those grouped nets together in sub-groups. For example, if we have a group of 100 nets having the same pair of source and sink coordinates, we will route them in sub-groups of K, instead of routing all of them together, where K is a certain percentage of the total number of nets in that group. This percentage is called the *net grouping factor* and is input by the users.

VII. Implementation Details

A. 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

Net no.	Modules
1	1,2,3,4,5
2	1,2,3,4,5
3	1,3
4	2,5
5	2,5
6	2,5

(a)

Net no.	Modules	Number of same net
1	1,2,3,4,5	2
2	1,3	1
3	2,5	3

(b)

Fig. 5. (a) Input netlist (b) Grouped netlist after the first stage grouping.

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 these two modules. The two intersecting points between this line and the boundaries of the modules will be found and the I/O pins will be placed into the grids of the intersection points.

B. Multi-pin Nets Handling

In order to handle multi-pin nets, we need to decompose a multi-pin net into a set of two-pin nets. There are several methods to decompose a multi-pin net into two-pin nets such as using the minimum spanning tree (MST) method, or the rectilinear steiner tree (RST) method. MST runs 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 at most by 6% to 9% by removing all the overlapping net segments to obtain a corresponding RST [10]. Since the runtime of an RST algorithm is usually much longer than that of an MST algorithm, MST is 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.

C. Cost function of the floorplanner

We employ the following cost function in our floorplanner:

$$\begin{aligned} Cost &= Area + \beta \times Wire + \lambda \times Max\_weight + \\ & \delta \times Blocked\_net \end{aligned}$$

where *Area* is the area of the minimum bounding rectangle of the packing, *Wire* is the total wire length of the interconnections in grid unit, *Max_weight* is the average number of wires in the top 10% most congested grids, and *Blocked_net* is the number of blocked nets in the packing. The values of β, δ and λ are set before the annealing process to maintain a balanced weighing between the importances of all the terms.

VIII. Experimental Results

We tested our floorplanner using three MCNC building block benchmarks, ami33, ami49 and playout. The number of modules and nets in these benchmarks are (33,123), (49,408) and (62,1161) respectively. The areas of the modules are scaled up uniformly to demonstrate the effects of buffer insertions. All the experiments were performed using an Intel 1.4 GHz processor with 256 MB memory. The values used for the parameters described in the Elmore delay model are based on the $0.18\mu m$ technology (see table I) [1, 15]. A simple global router is used to route the output for evaluation. In the global router, multi-pin nets are decomposed into two-pin nets based on the MST method and the two-pin nets are routed one after another by dynamic programming. There are limitations on the number of wires in each grid (*wiring capacity*). If a net can be routed from its source to its sink in shortest Manhattan distance without violating the buffer insertion constraint and the wiring capacity constraint, the net is said to be *routable*, otherwise, it is called an *unroutable* net. For each net, we will try to minimize the maximum congestion along its route, minimize the number of buffers used and maximize the amount of remaining buffer resources.

TABLE I
Parameters used in the Elmore delay computation.

Parameters	Value
wire resistance (Ω/mm)	0.075
wire capacitance (fF/mm)	0.118
wire fringing capacitance (fF/mm)	0.0641
intrinsic repeater delay (ps)	36.4
load capacitance/buffer capacitance (fF)	23.4
driver resistance/buffer resistance (Ω)	180

We have used a net grouping factor of 50%, 40% and 30% for the benchmark ami33, ami49 and playout respectively. (These net grouping factors are obtained from experiments.) We will compare the performance based on the number of unroutable nets and the congestion of the floorplan output. Unroutable nets refer to the nets that go over places running out of routing resources or have unsuccessful buffer insertion. Congestion indicates the average number of wires passing through the top 10% most congested grids (of area $10^3\mu m^2$). Every set of results shown in the following tables is an average obtained by running the experiment eight times.

A. Comparisons with other floorplanners

Table II shows the results of three floorplanners: the floorplanner with area and wire length optimization only (traditional floorplanner, FP1), the floorplanner presented in paper [16] (FP2), and our floorplanner using dynamic programming and with net grouping.

From table II, we can see that both our floorplanner and the one in paper [16], with a slight penalty in area (about 1% increase in deadspace), have a better routability compared with the traditional floorplanner.

For the floorplanner in paper [16] (FP2), although it is slightly better in congestion compared with our floorplanner (less than 1% better on average), we have fewer unroutable routes (about 37% improvements on average), which is a more important factor in satisfying the timing requirements of a design. On the other hand, the run time of our floorplanner is much faster than that of FP2, from about 2.1 times (ami49) to 3.8 times faster (playout).

TABLE II
COMPARISON ON ROUTABILITY WITH DIFFERENT FLOORPLANNING METHOD.

	Run-time (s)	**Wire-length ($10^3 \mu m$)**	**Dead-space (%)**	**Unroutable Net No.**	**Congestion (10^3 μm^{-2})**
[1] **ami33**	33 modules, 123 nets, grouping = 50%				
FP1	146.00	20.640	11.56	14.75	2.27
FP2	678.45	20.587	11.80	9.63	2.21
Ours	290.69	23.212	12.39	3.88	2.13
ami49	49 modules, 408 nets, grouping = 40%				
FP1	170.74	399.75	10.13	13.13	0.128
FP2	789.46	379.80	10.80	10.0	0.124
Ours	369.18	384.55	11.24	6.75	0.127
[1] **playout**	62 modules, 1611 nets, grouping = 30%				
FP1	552.68	306.76	11.75	163.88	25.60
FP2	3498.23	290.56	10.38	115.88	24.44
Ours	912.21	274.74	11.74	94.5	24.94

[1] Data sets ami33 and playout are scaled up 10 times in area.

TABLE III
COMPARISON ON EXECUTION TIME OF THE FLOORPLANNER WITH AND WITHOUT NET GROUPING.

Circuit	**Grouping %**	**Runtime without net grouping (s)**	**Runtime with net grouping (s)**
ami33	50	351.24	290.69
ami49	40	439.62	369.18
playout	30	3074.07	912.20

B. Net Grouping

Table III shows the results on the run time of the floorplanner with and without net grouping. It can be observed that there are significant improvements on the run time by using the net grouping method. For ami33 and ami49, the run time improvements are both about 17%, and that of playout is about 70%. The net grouping method is effective in run time improvement.

IX. CONCLUSION AND DISCUSSIONS

A routability-driven floorplannner is presented. Buffer locations are determined by dynamic programming according to the buffer resources and wiring congestion of the floorplan. Probabilistic analysis is applied to measure the routability of a floorplan solution after buffer locations are picked. To further speed up this sophisticated method and to handle bus-based routing, we have developed a net grouping method. Experimental results show that our floorplanning method can reduce congestion and the number of unroutable nets efficiently with only a small penalty in area.

In this work, we assume that all the nets are routed in their shortest Manhattan distances and a net will be blocked if none of its shortest routes can satisfy the buffer requirements. However, we can extend our technique to consider the case that a slight detour with good buffering to achieve acceptable timing. This can be done by considering a larger rectangular region, instead of the bounding rectangle from the source to the sink, and by modifying the dynamic programming accordingly to select buffer locations. Besides, the dynamic programming approach can be extended by routing the nets in a random order to remove the effects of the net order dependency.

REFERENCES

[1] S. I. Association, *SRC Design Sciences Concept Paper*, 1997.

[2] C. J. Alpert and A. Devgan, “Wire Segmenting for Improved Buffer Insertion”, *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 588-593, 1997.

[3] C. J. Alpert and J. Hu and S. S. Sapatnekar and P. G. Villarrubia, “A Practical Methodology for Early Buffer and Wire Resource Allocation”, *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 189-194, 2001.

[4] H. M. Chen and D. F. Wong and W. K. Mak and H. H. Yang, “Faster and More Accurate Wiring Evaluation in Interconnect-centric Floorplanning”, *Great Lakes Symposium on VLSI*, pp. 62-67, 2001.

[5] H. M. Chen and H. Zhou and F. Y. Young and D. F. Wong and H. H. Yang and N. Sherwani, “Integrated Floorplanning and Interconnect Planning”, *Proceedings IEEE International Conference on Computer-Aided Design*, pp.354-357, 1999.

[6] J. Cong, “Challenges and Opportunities for Design Innovations in Nanometer Technologies”, *SRC Design Sciences Concept Paper*, 1997.

[7] J. Cong and T. Kong and D. Z. Pan, “Buffer Block Planning for Interconnect-driven Floorplanning”, *Proceedings IEEE International Conference on Computer-Aided Design*, pp. 358-363, 1999.

[8] T. Cormen and C. Leiserson and R. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[9] F. F. Dragan and A. B. Kahng and I. Mandoiu and S. Muddu and A. Zelikovsky, “Provably Good Global Buffering using an Available Buffer Block Plan”, *Proceedings IEEE International Conference on Computer-Aided Design*, pp. 104-109, 2000.

[10] J. M. Ho and G. Vijayan and C. K. Wong, “A New Approach to the Rectilinear Steiner Tree Problem”, *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 161-166, 1989.

[11] J. Lou and S. Krishnamoorthy and H. S. Sheng, “Estimating Routing Congestion using Probablistic Analysis”, *Proceedings of International Symposium on Physical Design*, pp. 112-117, 2001.

[12] J. Lillis and 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*, Vol. 31, pp. 161-166, March, 1989.

[13] H. Murata and K. Fujiyoushi and S. Nakatake and Y. Kajitani, “Rectangle-Packing-Based Module Placement”, *Proceedings IEEE International Conference on Computer-Aided Design*, pp. 472-479, 1995.

[14] F. Rafiq and N. Sherwani and M. Chrzanowska-Jeske and Hannah H. Yang, “Integrated Floorplanning with Buffer/Channel Insertion for Bus-Based Microprocessor Designs”, *Proceedings of International Symposium on Physical Design*, pp. 56-61, 2002.

[15] P. Sarkar and C. K. Koh, “Routability-driven Repeater Block Planning for Interconnect-centric Floorplanning”, *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 20, pp. 660-671, 2001.

[16] C. W. Sham and E. F. Y. Young, “Routability Driven Floorplanner with Buffer Block Planning”, *Proceedings of International Symposium on Physical Design*, pp. 50-55, 2002.

[17] X. P. Tang and D. F. Wong, “Planning Buffer Locations by Network Flows”, *Proceedings of International Symposium on Physical Design*, pp. 186-191, 2000.