Timing Driven Placement using Physical Net Constraints

Bill Halpin
Design Technology, Intel
2200 Mission College
Santa Clara, CA 95054
1 408 765 9867
william.halpin@intel.com

C.Y. Roger Chen
Syracuse University
Department of EE&CS
Syracuse, NY 13244
1 315 443 4179
crchen@syr.edu

Naresh Sehgal
EPD, Intel
2200 Mission College
Santa Clara, CA 95054
1 408 765 4179
naresh.k.sehgal@intel.com

ABSTRACT

This paper presents a new timing driven placement algorithm that explicitly meets physical net lengths constraints. It is the first recursive bi-section placement (RBP) algorithm that meets precise half perimeter bounding box constraints on critical nets. At each level of the recursive bi-section, we use linear programming to ensure that all net constraints are met. Our method can easily be incorporated with existing RBP methods. We use the net constraint based placer to improve timing results by setting and meeting constraints on timing critical nets. We report significantly better timing results on each of the MCNC benchmarks and achieve an average optimization exploitation of 69% versus previously reported 53%.

1. INTRODUCTION

The placement field has been the subject of much research[1] [3][5][8][10][11], due to its importance in the design of VLSI circuits. There are 3 main goals in the automated placement problem: minimizing chip area, achieving routable designs, and maximizing circuit performance. Maximizing circuit performance has been the focus of continued attention in placement as semiconductor process advances have scaled cell delays more rapidly than interconnect delay[2].

Previous recursive bi-section timing driven placers have utilized net weights to reduce the length of timing critical nets. This is problematic since the improvement resulting from an increased weight is unpredictable as the various increasing and decreasing weights in the system interact. This is a serious drawback as the unpredictability can lead to oscillations in the net criticality and weight, limiting the extent of timing improvement.

2. MOTIVATION

In this paper we present a new placement method that explicitly meets net length constraints using linear programming[13][14]. This work was motivated by the need in high performance microprocessor design for accurate control of maximum net lengths and a frustration with the lack of precise net length control provided by net weighting approaches[15][16]. As mentioned earlier, net weights suffer from a number of drawbacks. Foremost of these, it is not possible to predict the net length obtained in response to a net weight.

The natural choice for controlling net delays in timing driven placement is a limit constraining the net's maximum bounding box. These net constraints give precise control to the circuit designer or to an automated timing driven placement algorithm.

Our approach can be easily incorporated with many RBP methods including analytical methods[6][5][10] and min-cut[11] approaches. Moreover, our placer immediately reports if it is not able to meet a net constraint at any level of partitioning. This allows the designer to take some other action such as circuit changes (e.g. sizing, buffering or splitting).

3. PROPOSED APPROACH

In this paper we focus on fixed die placements that are achieved through recursive bi-section placement (RBP)[5][6][7][10][11]. RBP is widely used in both academia and industry because it is fast, scales well to large problem sizes, and produces excellent wirelength results. RBP determines the locations of the cells through a series of recursive bi-partitioning steps of the circuit netlist and placement areas. The algorithm starts with a single "parent" region containing all of the cells and covering the entire placement area. In each successive step, the placement area of each "parent" region is divided to form 2 child regions. Similarly, the cells in each parent region are partitioned into 2 groups that are assigned to the 2 physical regions. Once a cell is assigned to a physical region, it will remain in the region for the remainder of the placement algorithm. These partitioning steps continue until each region contains less then some threshold of cells.

We introduce the following terms to explain RBP. A netlist hypergraph $H(V,E)$ has n cells $V=\{v_1,v_2,...,v_n\}$; a net $e \in E$ is defined to be a subset of V with size greater than one, i.e. $|e|>1$. A cell, v_i, has area, v_i^a, and a physical location given by its x and y coordinates: v_i^x and v_i^y.

A placement region, $r(r^a,r^c)$, is defined as a rectangular physical area, r^a and a set of cells r^c which are placed somewhere in r^a. The number of cells in the region is defined as $|r^c|$. Given a region r, a partitioning, or bi-section, $r \rightarrow \{r_l,r_r\}$ creates 2 children regions r_l and r_r referred to as the left and right children of r. We refer to r as the parent region of r_l and r_r. The partitioning divides the parent region's area r^a into to halves, r_l^a and r_r^a. The partitioning also

Design Automation Conference '01, June 18-22, 2001, Las Vegas, Nevada.

divides the cells in the parent region, r^c, into 2 groups, r_l^c and r_r^c. The following properties hold over the partitioning operation:
$r_l^a \cap r_r^a = \emptyset$, $r_l^a \cup r_r^a = r^a$, $r_l^c \cap r_r^c = \emptyset$ and $r_l^c \cup r_r^c = r^c$.

A placement, P, is a set of placement regions, R, in which every cell in V is in exactly one region and no region overlaps another region. A legal placement is one which every placement region contains one cell.

The critical step in RBP is the creation of the region partitioning: $r_i \rightarrow \{r_{il}, r_{ir}\}$. Several techniques have been used for this partitioning step including mincut[11] and numerical[1][5][10] methods. Our proposed method is not restricted to any of these techniques and can be incorporated with any RBP routine. In our work we use a quadratic programming based partitioner similar to GordianL[6].

The main contribution of this work is a technique to ensure that every cell is assigned to a region such that there is some location for every cell within its region for which all net constraints will be met.

3.1 Net Constraint Representation

A net constraint represents a physical bound on the half perimeter of a net and hence the placement of the cells connected by the net. Formally, a net constraint, $l = \langle l^c, l^b \rangle$, where $l^c \subset V$, $|l^c| > 1$ and l^b is the half perimeter of the bounding box constraining the locations of l^c.

For a placement, $P(H)$, a net constraint, l is feasible, $f(l) == 1$, if for every $v \in l^c$ there is some location of v within its region such that bounding box enclosing the cell locations is less than or equal to the constraint bound box. The set of net constraints for a netlist is defined as $M = \{l_1, l_2, ... l_n\}$. An unconstrained net will have $l^b = \infty$. A placement at any stage of partitioning is said to be feasible if $\forall_{l \in M} f(l) == 1$.

3.2 Constraint Chain Representation

In general it is not possible to create feasible assignments by checking each constraint individually. This is illustrated by the example in Figure 1. Figure 1 shows such a case with 2 constraints: l_1 and l_2 where $l_1^c = \{v_1, v_2\}$ and $l_2^c = \{v_2, v_3\}$, $l_1^b = 22$ and $l_2^b = 23$. If we consider l_1 separately, we would conclude that the original assignment, $v_1 \in r_4^c$ and $v_2 \in r_7^c$ is feasible. With $v_1 \in r_4^c$ at (20,30), the area shaded with the horizontal hashes shows the feasible locations for v_2.

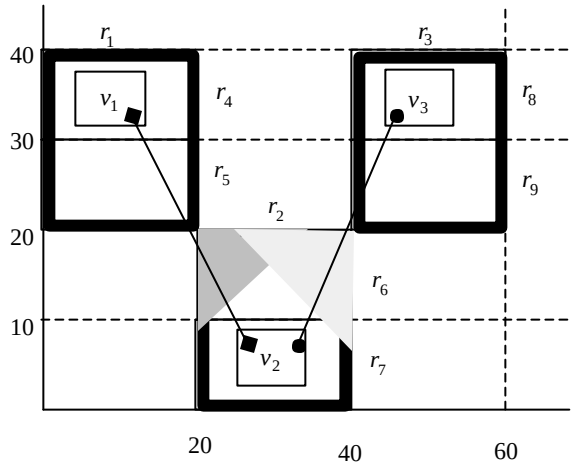


Figure 1. Chain Example

Similarly considering only l_2, we would conclude that the original assignment, $v_2 \in r_7^c$ and $v_3 \in r_8^c$ is feasible. With $v_3 \in r_8^c$ at (40, 30), the feasible locations for v_2 are shown by the area shaded with the vertical hashes. In fact, these assignments are not feasible as there is no location of $v_2 \in r_7^c$ for which both l_1 and l_2 are feasible. These two net constraints contain a common cell, v_2 and must be solved together which will result in the reassignment of v_2 from r_7^c to r_6^c. The feasible area for v_2 is the area in which the hashes overlap. We say that l_1 and l_2 are connected by v_2. The connected property is transitive. We define a chain as a set of net constraints in which every pair of net constraints is connected. A chain is feasible if all of the net constraints in the chain are feasible.

For a netlist, H, with net constraints, M, a maximal chain, MC, is a chain for which there is no constrained net in M connected to a net in MC that is not in MC. It is easy to show that if all MC are feasible for $P(H)$, then $P(H)$ is feasible. Furthermore each of the chain feasibility problems can be solved separately. We make use of these properties to efficiently solve the net constraint problem.

4. MODELING AS LINEAR PROGRAM

We have implemented our net constraint algorithm as a linear program. This section describes the modeling.

4.1 Modeling for Single Net Constraint

In this section we describe the linear programming model for a single net constraint. As mentioned earlier, a net constraint, l, has *m* cells $l^c = \{v_{p1}, v_{p2}, ..., v_{pm}\}$. These cells must be placed so that the sum of the height, l^h, and width, l^w, of their bounding box is less than the net constraint, $l^h + l^w \le l^b$. Unlike other RBP techniques[1][5][6][10][11] our problem requires that we solve for the x and y coordinates of each instance simultaneously as the constraint problem is not separable in x and y since $l^h + l^w \le l^b$ depends on both the x and y values.

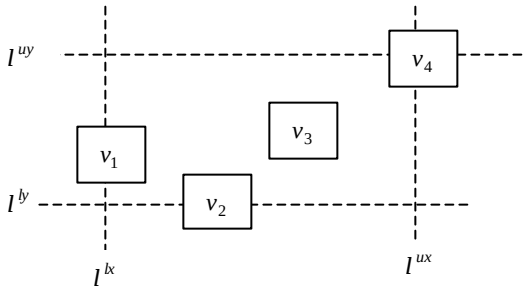
We introduce 2 variables, v_i^x and v_i^y, representing the location of each cell. We compute l^h and l^w through the introduction of 4 bounding box variables, l^{lx}, l^{ux}, l^{ly}, and l^{uy} which form the bounding rectangle of l^c. Figure 2 illustrates the use of the each of the four constraint variables. In our LP model we introduce a constraint for each bounding variable and cell in l^c. This will result in $4 * |l^c|$ inequality constraints in our LP for each net constraint.

As shown in Figure 2, we add a constraint for each cell and bounding line pair for a total of 16 constraints. The first 4 establish l^{lx} as the left boundary of $v \in l^c$, the next 4 establish l^{ux} as the right boundary of $v \in l$, and so on.

Once we have the bounding lines, it is easy to compute the height and width of the bounding box: $l^h = l^{uy} - l^{ly}$ and $l^w = l^{ux} - l^{lx}$. We constrain l^c to be within its bound l^b with the constraint: $(l^{ux} - l^{lx}) + (l^{uy} - l^{ly}) \le l^b$.

4.1 Modeling for Cell Assignments

We now describe our model of the cell assignments and region constraints. As mentioned earlier, the partitioner cuts each parent region into two children and assigns each cell in the parent region to



$$l^{lx} \le v_1^x,\ l^{lx} \le v_2^x,\ l^{lx} \le v_3^x,\ l^{lx} \le v_4^x$$
$$l^{ux} \ge v_1^x,\ l^{ux} \ge v_2^x,\ l^{ux} \ge v_3^x,\ l^{ux} \ge v_4^x$$
$$l^{ly} \le v_1^y,\ l^{ly} \le v_2^y,\ l^{ly} \le v_3^y,\ l^{ly} \le v_4^y$$
$$l^{uy} \ge v_1^y,\ l^{uy} \ge v_2^y,\ l^{uy} \ge v_3^y,\ l^{uy} \ge v_4^y$$

Figure 2. Modeling for Net Constraint

one of the children. The object of this assignment is to minimize the overall total wirelength.

We have 3 goals in our modeling of the physical locations of a cell. First we must ensure that the cell remains in one of the 2 child regions. This is a requirement of the RBP problem. Our second goal is to bias the assignment of the cell to the child suggested by the partitioner. We impose this bias to minimize the perturbation from the original partitioning assignment. We will allow cell reassignments only when there is no other way to meet the constraint. Our third goal is to not violate the capacity constraints of either child.

Referring to Figure 3, we explain the modeling for the cell movement and reassignment. We see a parent region bounded on the left by r_{lx}^a, on the right by r_{ux}^a, on the bottom by r_{ly}^a and the top by $r_{uy'}^a$. The parent was cut horizontally at r_{uy}^a resulting in upper and lower regions. The cell, v_1, was assigned to the lower region by the original partitioning. Since v_1 is not allowed to move out of the parent region, the location of v_1 is

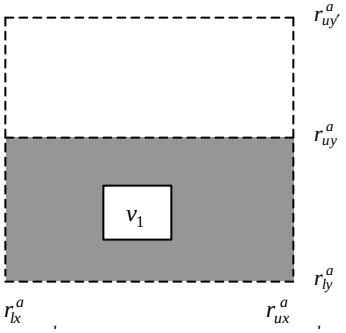


Figure 3. Modeling for Cell Assignments

bounded on the left, right and bottom resulting in the following 3 constraints: $r_{lx}^a \le v_1^x$, $r_{ux}^a \ge v_1^x$, and $r_{ly}^a \le v_1^y$ respectively.

The remaining constraints concern the child region assignment of v_1. If $v_1^y \le r_{uy}^a$ then no reassignment has occurred. If $r_{uy}^a < v_1^y \le r_{uy'}^a$ then the cell has been reassigned. We model this cell reassignment with an additional variable Δv_1, which we call the reassignment variable. If $\Delta v_1 = 0$ then the cell has been assigned to the same region suggested by the partitioner. If $\Delta v_1 > 0$ then the cell has been reassigned to the other child. We introduce 2 new constraints into the model to reflect the limitation of the movement of the cell within its two child regions:

$v_1^y \le r_{uy}^a + \Delta v_1 \left(r_{uy'}^a - r_{uy}^a \right)$, and $0 \le \Delta v_1 \le 1$.

These two constraints together represent the constraint on the maximum value of v_1^y. We will minimize Δv_1 in our objective function to bias against reassignments.

4.2 Objective Function

Section 4.1 discussed the cell modeling and introduces the cell reassignment variable Δv. In our net constraint placer we minimize the reassignments to preserve the original partitioning assignment unless necessary to meet the net constraints. The objective function minimizes the reassignment through the term $\min : \sum_{i \in l_j^c, j \in M} \Delta v_i$.

5. CONSTRAINT PLACEMENT

We now present the overall net constraint placement algorithm. We deploy the net constraint solver within each cutting stage of the recursive bi-section, immediately after the numerical or min-cut algorithm has done its initial assignment of cells to regions. As discussed in section 3.2 the problem is separable to a series of maximal chain placement problems. For each maximal chain, we form the linear program as described in Section 4. We solve this linear program using the public domain linear programming solver, lp_solve[17]. For any cell v_i for which $\Delta v_i > 0$ we re-assign this cell to the other child of its parent. We complete these operations for all maximal chains and then proceed to the next level of partitioning.

6. TIMING DRIVEN PLACEMENT

One of the motivations for this net constraint approach was its application in timing driven placement. In this section we describe how we utilize our net constraint placer to reduce the longest circuit path.

6.1 Determining the Net Constraints

Our net constraint placement algorithm requires that we identify the critical circuit nets and establish the appropriate constraints on these nets to reduce the longest circuit path. We employ an iterative approach where we place, recompute slacks, and then slowly tighten the constraints on critical nets. We then start the cycle again at the placement stage.

In each of the timing iterations, we tighten the constraint for critical nets. A net is critical if it is on a path with a delay in the longest 5%. The constraint for nets on the most critical path is reduced by 25% while the constraint on the least critical nets are reduced 5%. The remaining critical nets will vary linearly between 25% and 5% depending on the delay of the path it is on.

7. RESULTS

We have implemented our net constraint algorithm in C++ and on LINUX and ran all of our experiments on a IBM MPro system with dual Intel Pentium™ 800 MHz processors. We use lp_solve[17] to solve our linear program. We used the MCNC[18] benchmarks to compare our results to Eisenmann [8] and TimberWolf [3]. We take the results for both approaches from[8]. As with [8] we have assumed a capacitance per unit length of 242 pF/m and a resistance per unit length of 25.5 kO/m. Table 1 presents the absolute longest

delay path values for each routine in nanoseconds and the improvement in worst path delay over the other 2 methods.

Our method produces lower longest path timing results on every circuit with the improvements ranging from 2.7% to 11.8% over [8] and 80.5% to 90.5% over [3]. The relative timing and CPU run time numbers are presented in Table 2 along with the "unloaded" path delay for each circuit. The unloaded path delay is computed by setting all of the wire parasitic values to 0 and computing the longest path for circuit. In order to compare the effectiveness of the timing driven methods we calculate the exploitation potential for each circuit and placer. The exploitation potential is a measure of how much the density result can be improved relative to the unloaded path delay. We calculate the exploitation as the percent of the optimization potential that was achieved for the timing driven method over its density result.

Our net constraint based timing driven method is significantly better in exploiting the optimization potential than previous methods. Our exploitation exceeds that of [8] with improvement ranging from 21.5% to 110% and an average exploitation improvement of 57%. Comparing with [3] our exploitation improvement ranges from 6.6% to 160% and averages 97%.

8. CONCLUSION

We have presented the first recursive bi-section placement method that explicitly meets net constraints. At each bi-section step, we use linear programming to create an assignment of cells to regions that meets the net constraints. The linear program model is constructed to minimize the number of reassignments from the original partitioning. We present significant timing improvements on every MCNC benchmark in both longest path delay and optimization potential exploitation.

9. REFERENCES

[1] Michael A. B. Jackson, Arvind Srinivasan and E. S. Kuh, "A Fast Algorithm for Performance-Driven Placement," Digest of Technical Papers, ICCAD, pp. 328-331, November 1990.

[2] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron," pp. 203-211, ICCAD 1998.

[3] William Swartz and Carl Sechen, "Timing Driven Placement for Large Standard Cell Circuits," DAC, pp. 211-215, 1995.

[4] Wern-Jieh and Carl Sechen, "Efficient and Effective Placement for Very Large Circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 349-359, 1995.

[5] A. Srinivasan, A K. Chaudhary, E. S. Kuh, "RITUAL: Performance Driven Placement Algorithm for Small Cell ICs," ICCAD, pp. 48-51, Nov. 1991.

[6] Jurgen M. Kleinhans, Georg Sigl, Frank M. Johannes, and Kurt Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," IEEE Transactions on Computer Aided Design, Volume 10, No. 3 pp. 356-365, 1991.

[7] K. Doll, F. M. Johannes, and K.J. Antreich, "Iterative placement improvement by network flow methods," IEEE Transactions on CAD, vol. 13 pp.1190-1200, Oct 1994.

[8] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning," ACM/IEEE DAC, 1998.

[9] J. Cong, "Timing models for Interconnects and Devices," DAC, 1997.

[10] R.S. Tsay, "Timing-Driven Placement," DAC, 1997.

[11] Shih-Lian Ou and Massoud Pedram, "Timing-driven Placement Based on Partitioning with Dynamic Cut-net Control," DAC, 2000.

[12] S. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," ICCAD 2000.

[13] Jorge Nocedal and Stephen J. Wright, "Numerical Optimization," Springer-Verlag, 1999.

[14] H. Paul Williams, "Model Building in Mathematical Programming," John Wiley and Sons, 1999.

[15] Bill Halpin, C.Y. Roger Chen, and Naresh Sehgal, "A Sensitivity Based Placer for Standard Cells," GLS-VLSI, 1999.

[16] Ren-Song Tsay and Juergen Koehl, "An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement," DAC, pp620-624, 1991.

[17] ftp://ftp.es.ele.tue.nl/pub/lp_solve/. Information and Communication Systems group at the Electrical Engineering department of the Eindhoven University of Technology, 1998.

[18] "www.cbl.ncsu.edu/benchmarks/layoutsynth92/

Circuit	Eisenman'98		TimberWolf[7]		Our Approach		Improvement	Improvement
	without timing	with timing	without timing	with timing	without timing	with timing	vs Eisenmann	vs TimberWolf
fract	21.2	19.8	208	131	21.8	18.8	5.1%	85.6%
struct	92.5	90.1	907	449	93.1	87.7	2.7%	80.5%
biomed	48.6	35.7	-	-	49.2	31.5	11.8%	
avq.small	102	80	1106	798	104	75.6	5.5%	90.5%
avq.large	113	94	-	-	115	85.7	8.8%	

Table 1 Timing Results: Longest Path for Density and Timing Algorithms

Circuit	Eisenmann[13]			TimberWolf[7]			Our Approach			Improvement	Improvement
	lower bound	exploitation	rel. CPU	lower bound	exploitation	rel. CPU	lower bound	exploitation	rel. CPU	vs Eisenmann	vs TimberWolf
fract	18.5	52%	0.33	18.5	41%	55.56	18.5	91%	1	75.3%	123.7%
struct	84	28%	0.16	84	56%	4.57	84	59%	1	110.2%	6.6%
biomed	27	60%	0.73	-			27	80%	1	33.5%	
avq.small	69.9	69%	2.27	142	32%	25.97	69.9	83%	1	21.5%	160.7%
avq.large	79.9	57%	2.08	-			79.9	83%	1	45.4%	
avg		**53%**	**111%**		**43%**	**2870%**		**79%**	**1**	**57%**	**97%**

Table 2 Relative Timing Results: Exploitation of Optimization Potential and relative CPU requirements