Hippocrates: First-Do-No-Harm Detailed Placement

Haoxing Ren, David Z. Pan†, Charles J. Alpert, Gi-Joon Nam and Paul Villarrubia
IBM Corporation, 11400 Burnet Road, Austin TX 78758
{haoxing,alpert,gnam,pgvillar}@us.ibm.com
†Department of ECE, University of Texas at Austin, Austin TX 78712
dpan@ece.utexas.edu

ABSTRACT

Physical synthesis optimizations and engineering change orders typically change the locations of cells, resize cells or add more cells to the design after global placement. Unfortunately, those changes usually lead to wirelength increases; thus another pass of optimizations to further improve wirelength, timing and routing congestion characteristics is required. Simple wirelength-driven detailed placement techniques could be useful in this scenario. While such techniques can help to reduce wirelength, ones without careful timing constraint considerations might degrade the timing characteristics (worst negative slack, total negative slack, etc) and/or introduce more electrical violations (exceeding maximum output load constraints and maximum input slew constraints). In this paper, we propose a new detailed placement paradigm, which use a set of pin-based timing and electrical constraints in detailed placement to prevent it from degrading timing or violating electrical constraints while reducing wirelength, thus dubbed as Hippocrates: FIRST-DO-NO-HARM optimizations. Our experimental results show great promises. By honoring these constraints, our detailed placement technique not only reduces total wirelength (TWL), but also significantly improves timing, achieving 37% better total negative slack (TNS).

1. INTRODUCTION

"*Primum non nocere*" is a Latin phrase that means "First, do no harm". Originated from Hippocratic Corpus, it is one of the principle precepts all medical students are taught in schools. It reminds a physician that he or she must consider the possible harm that any intervention might do.

Global placement is one of the most critical process in modern physical synthesis. Its task is to determine the overall locations of objects in the design. Many global placement algorithms have been proposed to minimize total wirelength [1] [2] [3] [4] [5] [6] [7], just naming a few. However, physical synthesis transformations [8], such as buffering [9] and gate sizing [10], are applied after placement to further optimize timing. These transforms usually insert new cells or change the size of existing cells. Engineering change order (ECO) is another source of modifications for designs under optimization. It could also introduce new logics, change the physical sizes of objects, or change the locations of existing cells. All these changes might result in overlaps among cells. Therefore one needs additional legalization to remove those overlaps. Although many legalization algorithms have been proposed to minimize the disturbance to the original placement [11] [12] [13] [14] [15] [16], they usually result in wirelength degradation. This is because the relative order of newly inserted/sized cells are not fully optimized as much as it was done during global placement. The poor wirelength causes inferior timing and routing congestion problems. Therefore it is of great interest to further improve wirelength after physical synthesis or Engineering Change Order (ECO).

To the best of our knowledge, there is no previous work on incremental placement for both wirelength reduction and timing improvement. Several previous works [17] [18] and [19] addressed incremental placement for timing improvement issue. [19] proposed using differential timing method and Linear Programming (LP) to formulate the timing driven placement of critical cells. It moves a few critical cells, which helps improve timing significantly. However, our intention in this paper is to improve timing as well as total wirelength. Thus, we would have to include every gate in the LP formulation unlike [19], which only includes a portion of critical gates. This would result in severe overlapping situations and huge timing degradation during legalization. Similar to [19], the iterative method proposed by [17] and the net contraction method proposed by [18] both use analytical method to find better locations for a few critical cells. Thus for the same reason, they cannot directly solve wirelength optimization problem either. Therefore we resort to wirelength-driven detailed placement techniques to solve this problem.

Detailed placement techniques such as Simulated Annealing (SA) based swapping and moving [7], cell interleaving [12], branch-and-bound reordering [13], branch-and-price reordering [20], guided local search [21], global swap and local reordering [22], and net length constrained SA approach [23] can all reduce the total wirelength. However, reducing total wirelength does not necessarily result in timing improvement, particularly after physical synthesis. This is because detailed placement might increase the wirelength on critical paths while reducing the total wirelength.

As Hippocrates would remind us, we should "first, do no harm" to the timing characteristics of the design after physical synthesis or ECO because the design is already optimized. To achieve this FIRST-DO-NO-HARM paradigm, we need to model the delay impact of any placement change. Surely we can call a static timer after each placement change. But it is too costly to do so during detailed placement. The differential timing model [19] helps to reduce the computation need while still providing accurate enough timing estimation for placement. However, the path propagation technique of [19] is too costly to be directly used in any detailed placement framework. Therefore an efficient delay model is needed for detailed placement to avoid doing any harm to timing critical paths while improving total wirelength.

Our contributions in this work are:

- We develop a pin-based timing and electrical constraint model to prevent detailed placement from degrading timing or violating electrical constraints. These constraints need to be generated *only once* before detailed placement. They remain valid through the placement process no matter how many cells are moved.
- We implement these constraints in an industry strength detailed placer. The constrained detailed placer can significantly improve timing while reducing total wirelength.
- These constraints are simple to implement and can be easily integrated into a majority of detailed placement frameworks.

In the rest of the paper, we will introduce these constraints and provide the proof that honoring these pin-based timing constraints will DO NO HARM to the timing results during detailed placement. We will also demonstrate how we implement these constraints in detailed placer and demonstrate significant improvements on timing and wirelength.

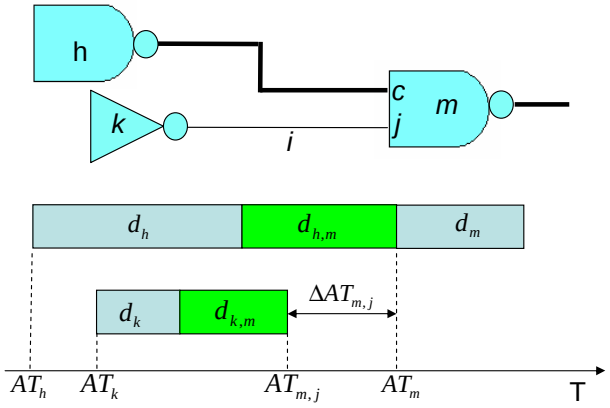

Figure 1: Critical Path and Delta Arrival Time

2. PIN-BASED TIMING AND ELECTRICAL CONSTRAINTS

In this section, we will model timing and electrical constraints for detailed placement. The purpose of adding these constraints is to prevent any degradation of timing results (worst negative slack, total negative slacks etc) or electrical violation such as maximum output load constraints and maximum input slew constraints.

Unlike [19] which imposes constraints on timing paths, our timing constraints are imposed on individual pins. Although these constraints that we model may be conservative, using pin-based constraint would greatly simplify the timing computation during placement because the expensive path propagation computation is not required.

2.1 *Delta Arrival Time* Constraints

As shown in Fig 1, we can build a timing diagram with gate and wire delay segments[1]. Let N be a set of gates, primary inputs (PIs) and primary outputs (POs) in a design; and for a gate $k, m \in N$, d_k be the delay of gate k, and $d_{k,m}$ be the delay of wire connection between k and m.

The arrival time of each pin of a gate is defined as the summation of delay segments from timing start points, i.e. PIs or the output of a sequential logic, to the pin itself. The arrival time of each gate is simply defined as the summation of delay segments on the most critical input pin.

Thus, let N_m be a set of gates or PIs connected to the input of gate m, and gate $k \in N_m$ connected to input pin j of gate m. The arrival time of pin j will be:

$$AT_{m,j} = AT_k + d_k + d_{k,m} \tag{1}$$

where $AT_{m,j}$ is the arrival time of pin j of gate m, AT_k is the arrival time of gate k (thus, the arrival time of the most critical input pins of gate k).

We define the *delta arrival time* of input pin as the differences between the arrival time of pin itself and the arrival time of the gate. As shown in Fig 1, the delta arrival time of the input j of gate m, $\Delta AT_{m,j}$, is defined as:

$$\Delta AT_{m,j} = AT_m - AT_{m,j} \tag{2}$$

For example, suppose $AT_h = 1ns$, $AT_k = 2ns$, $d_h = 3ns$, $d_k = 1ns$, $d_{h,m} = 2ns$ and $d_{k,m} = 1.5ns$; then arrival time at pin c of gate m is $AT_{m,c} = AT_h + d_h + d_{h,m} = 6ns$, while arrival time at pin j of gate m is $AT_{m,j} = AT_k + d_k + d_{k,m} = 4.5ns$. Therefore the most critical input pin of gate m is pin c and arrival time AT_m of gate m is equal to $AT_{m,c} = 6ns$. The delta arrival time at pin j is $\Delta AT_{m,j} = AT_m - AT_{m,j} = 1.5ns$. The delta arrival time at pin c is $\Delta AT_{m,c} = AT_m - AT_{m,c} = 0ns$. The delta arrival time for a primary output pin (PO) or a sequential logic input pin is always zero because there is only one pin to compare with.

Note that *delta arrival time* is absolutely **different** from *slack*. Slack is the difference between arrival time and required arrival time, while delta arrival time is the difference between the arrival time of an input pin to the most critical input pin. Pins with same slack can have different delta arrival time, and pins with same delta arrival time can have different slacks.

For combinational gates, the delta arrival time indicates how much arrival time can increase on a particular pin before it will make timing worse on the output pin. For example, if arrival time on pin j increases $0.5ns$, because $0.5ns < \Delta AT_{m,j} = 1.5ns$, pin c is still the most critical pin and the arrival time of gate m will not change. This observation leads to Lemma 1.

LEMMA 1. *Suppose the output pin of gate k is connected to the input pin j of gate m, i.e. $k \in N_m$, let new_AT_k be the new arrival time of gate k after placement modifications.*

If 1) the combined increment of gate delay k (Δd_k) and wire delay connecting k and m ($\Delta d_{k,m}$) is less than or equal to the delta arrival time of pin j, i.e. $\Delta d_k + \Delta d_{k,m} \leq \Delta AT_{m,j}$ and (2) the arrival times of gates in N_m do not increase,i.e. $new\_AT_k \leq AT_k$ for every $k \in N_m$, then the arrival time on gate m will not increase, i.e. $new\_AT_m \leq AT_m$.

PROOF. Substitute $AT_{m,j}$ with (2), (1) becomes

$$AT_m = AT_k + d_k + d_{m,k} + \Delta AT_{m,j} \tag{3}$$

If the placement changes, the gate delay d_m and wire delay $d_{m,k}$ and arrival time on all the gates will also change.

Thus, the new arrival time on gate m can be computed as

$$new\_AT_m = \max_{k \in N_m} [new\_AT_k + d_k + d_{k,m} + \Delta d_k + \Delta d_{k,m}] \tag{4}$$

Since $\Delta d_k + \Delta d_{k,m} \leq \Delta AT_{m,j}$ and $new\_AT_k \leq AT_k$ for every $k \in N_m$, we will always have

$$\begin{aligned} new\_AT_m &= \max_{k \in N_m} [new\_AT_k + d_k + d_{k,m} + \Delta d_k + \Delta d_{k,m}] \\ &\leq \max_{k \in N_m} [new\_AT_k + d_k + d_{k,m} + \Delta AT_{m,j}] \\ &\leq \max_{k \in N_m} [AT_k + d_k + d_{k,m} + \Delta AT_{m,j}] \\ &\leq AT_m \quad \square \end{aligned}$$

Therefore we can prove following theorem:

THEOREM 1. *The arrival time of any gate is nonincreasing, and slack on any timing end point (PO or sequential logic input pin) will not degrade if the change of combined gate and wire delay is always less than or equal to the delta arrival time on input pin (or PO) which it connects to, i.e.*

$$\Delta d_k + \Delta d_{k,m} \leq \Delta AT_{m,j} \tag{5}$$

PROOF. Suppose we traverse the netlist in a topological order from input to output and label gates in that order from 1 to n. At the beginning of traversal, we assume the arrival time on the timing start points are constants. Suppose gates 1 to k are the gates directly connected to timing beginning points. As Lemma 1 indicates, AT_1 to AT_k will not increase since the change of wire delay on any input pin is less than or equal to the delta arrival time on the input pin and the arrival times on PI and sequential logic output pin are constants. Now as we traverse to gate $k+1$, the arrival time of its input pins are determined by the arrival times of its previous gates and the delays between gate $k+1$ and its previous gates. Since we have proven that all the arrival time of gate 1 to k are nonincreasing, and the change of delay on any input pin is less than or equal to the delta arrival time on the input pin, according to Lemma 1, the arrival time of gate $k+1$ is also nonincreasing. In the same manner, we can approve

[1]We made a couple of simplifications to the static timing analysis model in order to simplify the explanation and notation. First, we assume the gate delays from each input pin to output pin are the same for ease of explaining the key idea of delta arrival time. For non-uniform input to output pin delay case (mostly on complex gates only), the difference can be incorporated into the input pin arrival time. Second, among all feasible signal transitions, i.e. rise-fall etc, we only consider the most critical transition, which is conservative as well.

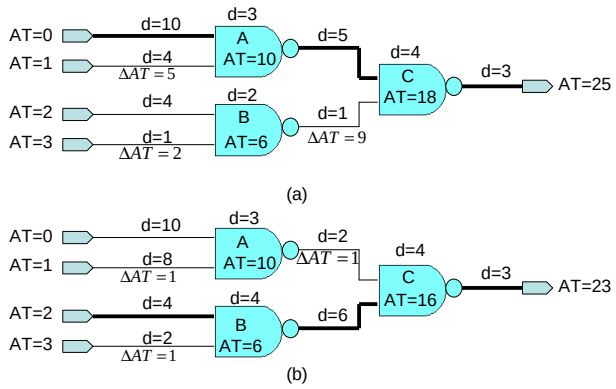

Figure 2: Delay and Arrival Time of a Simple Circuit

the arrival times of gates 1 to n are all nonincreasing. At the end of traverse where it hits timing end points, the arrival times of those gates or POs are also nonincreasing. Since the required arrival times (RAT) at timing end points are constant, the slacks on the timing end points will not degrade. □

Fig 2 shows a small network with 4 PI, 1 PO, 3 gates and 7 nets. The upper figure 2(a) shows the original gate and wire delays, arrival times on all three gates, PIs and PO, and delta arrival times on non-critical internal pins, while the lower figure 2(b) shows the changed gate/wire delays, and new arrival times. We can see that the increase on the merged gate delay and wire delay is less than the delta arrival time. For example, the combined gate and wire delay between gate B and C changes from 2+1=3 to 4+6=10. However, the ΔAT on this connection is 9 which is still greater than the mount of delay increase 10-3=7. Even the critical input pin of gate C changes, the arrival time on the input of C is still reduced from 18 to 16. We can see the arrival times on all the gates do not increase and so does the PO arrival time. Therefore the slack on PO is not degraded.

Theorem 1 asserts a delay constraint on each gate input pin or PO. It guarantees a non-deteriorating timing result if all the delay constraints are satisfied during any placement transformation [2] . Note that this analysis only needs to be done once, and it will be valid during the entire detailed placement process no matter how many cells change positions.

2.2 Electrical Constraints

In addition to delay constraints, detailed placement also needs to satisfy electrical constraints, such as maximum output capacitance and maximum input slew. The output capacitance constraint specifies the maximum load capacitance a gate can drive, including wire capacitances and sink pin capacitances. The slew constraint specifies the maximum slew on an input pin.

Suppose the maximum input pin slew is $s_{m,j}^{max}$ on pin j of gate m, and the maximum output pin load is c_k^{max} for gate k, we can formulate the output load and input slew constrains as follows.

$$\Delta s_{m,j} \leq s_{m,j}^{max} - old\_s_{m,j} \quad (6)$$

$$\Delta c_k \leq c_k^{max} - old\_c_k \quad (7)$$

where $\Delta s_{m,j}$ is the slew increment on pin j of gate m, and Δc_k is the load increment of gate k. $old\_s_{m,j}$ is the original slew on pin j, and old_c_k is the original load of gate k.

Both constraints are pin based constraints. The load constraint is asserted on each output pin and PI, while slew constraint is asserted on each input pin and PO.

3. CONSTRAINED DETAILED PLACEMENT

[2]We only consider the setup timing constraints but not hold timing constraints. Hold timing constraints can be fixed by simple buffering or gate sizing techniques following detailed placement.

Detailed placement can mean different things to different people. In this paper we mean the placement transforms that convert placement from one legal solution to another legal solution. These transforms take a legally placed netlist, change locations of cells while still maintaining the legality. Normally these transforms only check whether the movements reduce the total wirelength or not. Our "First-Do-No-Harm" constrained detailed placer will also check whether the timing and electrical constraints are met under a weighted total wirelength objective. In the rest of this section, we first derive these constraints, define the objective function and then present the overall detailed placement flow.

3.1 Constraint Formulation

During detailed placement, we can get an accurate estimation of the net half perimeter wirelength (HPWL) and Manhattan distance between source and sink. If we can roughly estimate delay or slew based on HPWL and Manhattan distance, then we can check whether the constraints given in section 2 are satisfied or not.

To do this we employ a differential gate delay and wire delay model similar to what proposed in [19], which estimates delay and slew increments of placement change. However, the main difference is the gate delay modeling. Whereas [19] models the gate delay and output slew with input pin slew and output pin load, we assume input pin slew as fixed. Therefore the gate delay and output slew is only determined by output load. The advantage of this is to avoid slew propagation, which is time consuming because one has to propagate the slew all the way down to the end and recompute the timing on many cells down the way. As long as we can keep the delta arrival time constraints, Theorem 1 guarantees that the arrival time on the most critical input pin of any gate will not increase, therefore it is reasonable to assume that the input slew on that pin will not increase as well. Thus using a constant input slew is a conservative estimation of gate delay. Using a conservative gate delay modeling actually makes the timing constraints more conservative, which helps safeguarding any timing degradation.

Gate delay and output slew can be represented as linear functions of input slew and output load as follows:

$$d_k = A_0 + A_1 c_k + A_2 s_{k,j} \quad (8)$$

$$s_k = B_0 + B_1 c_k + B_2 s_{k,j} \quad (9)$$

where d_k and s_k are the delay and output slew of gate k; j is the most critical input pin of gate k and $s_{k,j}$ is the input slew on pin j. A_0, A_1, A_2 and B_0, B_1 and B_2 are constants determined by the standard cell library characterization. Since we assume the critical input pin slew is constant, the differential gate delay and output slew can be computed by

$$\Delta d_k = A_k \Delta c_k \quad (10)$$

$$\Delta s_k = B_k \Delta c_k \quad (11)$$

where Δd_k and Δs_k is the gate delay and output slew increments for gate k, respectively. The $A_k = A_1$ and $B_k = B_1$ are the delay sensitivity and slew sensitivity to output load for gate k, respectively. Δc_k is the total output load increment, which can be computed by

$$\Delta c_k = c \Delta l_i \quad (12)$$

where c is the unit wire capacitance (here we only use one wiring layer for estimation); Δl_i is the total wirelength (HPWL) increment for net i which gate k drives. We do not consider effective capacitance because it is costly to estimate it during detailed placement.

Same as [19], we model the wire delay as if there is a separate wire connecting source and sink (star wire model). Although this model is not very accurate, it is efficient to embedded into placement engine and accurate enough for low fanout nets.

The delay $d_{k,m}$ and slew $s_{k,m}$ of the wire connecting gate k and m can be computed as

$$d_{k,m} = K_D \cdot r \cdot l_{k,m}(\frac{c \cdot l_{k,m}}{2} + cpin_k) \quad (13)$$

$$s_{k,m} = K_S \cdot r \cdot l_{k,m}(\frac{c \cdot l_{k,m}}{2} + cpin_k) \quad (14)$$

where $l_{k,m}$ is the Manhattan distance between gate k and m; $K_D = 0.69$, and $K_S = 2.2$ are constants based on transition of 10% to 90% VDD. Note that here $s_{k,m}$ is the additional slew introduced by the wire, the real slew on the input of gate m is the sum of gate slew s_k and wire slew $s_{k,m}$.

The differential wire delay and slew can be computed by:

$$\Delta d_{k,m} = K_D[r(c \cdot old\_l_{k,m} + cpin_k) \cdot \Delta l_{k,m} + \frac{rc(\Delta l_{k,m})^2}{2}] \quad (15)$$

$$\Delta s_{k,m} = K_S[r(c \cdot old\_l_{k,m} + cpin_k) \cdot \Delta l_{k,m} + \frac{rc(\Delta l_{k,m})^2}{2}] \quad (16)$$

where $\Delta d_{k,m}$ is the wire delay increment between gate k and m; $\Delta s_{k,m}$ is the wire slew increment; $\Delta l_{k,m}$ is the Manhattan distance increment between gate k and m, and $old\_l_{k,m}$ is the original Manhattan distance between k and m.

Add (10) with (15), and (11) with (16) and replace Δc_k with (12), we can compute the combined gate and wire delay and slew increments as:

$$\Delta d_k + \Delta d_{k,m} = \alpha \Delta l_i + \beta \Delta l_{k,m} + \gamma \Delta l_{k,m}^2 \quad (17)$$

$$\Delta s_k + \Delta s_{k,m} = \zeta \Delta l_i + \eta \Delta l_{k,m} + \theta \Delta l_{k,m}^2 \quad (18)$$

where

$$\begin{aligned} \alpha &= A_k c \\ \beta &= K_D r(c \cdot old\_l_{k,m} + cpin_k) \\ \gamma &= K_D \frac{rc}{2} \\ \zeta &= B_k c \\ \eta &= K_S r(c \cdot old\_l_{k,m} + cpin_k) \\ \theta &= K_S \frac{rc}{2} \end{aligned} \quad (19)$$

Using the differential timing equations (17) and (18) we can easily convert the delta arrival time constraints (5), maximum input slew constraints (6), and maximum output capacitance constraints (7), into constraints represented by the net HPWL and gate-to-gate wire Manhattan distance. These constrains are:

$$\begin{aligned} \alpha \Delta l_i + \beta \Delta l_{k,m} + \gamma \Delta l_{k,m}^2 &\leq \Delta AT_{m,j} \\ \zeta \Delta l_i + \eta \Delta l_{k,m} + \theta \Delta l_{k,m}^2 &\leq s_{m,j}^{max} - old\_s_j \\ c \Delta l_i &\leq c_k^{max} - old\_c_k \end{aligned} \quad (20)$$

where pin j is the input pin of gate m which is connected to gate k by net i.

3.2 Objective Function

The objective of our detailed placement is to reduce both TWL and TNS. Assuming we can guarantee that slacks do not degrade as Theorem 1 states, we can use weighted total wirelength as the optimization objective for detailed placement. Critical nets (nets with negative slacks) are given higher weights than other nets. The weighted wirelength objective function is given below.

$$WTWL = \sum w_i l_i \quad (21)$$

where WTWL is the weighted total wirelength. w_i is the net weight for net i, and l_i is the HPWL of net i.

We have tried different net weighting schemes [24] and found that the final results are not that different probably due to the constraints. Therefore we choose to use a simple slack based netweight assignment as shown below:

$$w_i = \begin{cases} \alpha - \beta slk_i & if \quad slk_i < 0 \\ \alpha & if \quad slk_i \geq 0 \end{cases} \quad (22)$$

where w_i is the netweight of net i, slk_i is the slack on net i; α and β are two positive constants.

Detailed placement normally moves a set of cells then evaluates the objective function. If the objective function decreases, it accepts those movements, otherwise it rejects those movements. Therefore we only need to check whether the WTWL increment $\Delta WTWL$ is negative or not. $\Delta WTWL$ can be computed as

$$\Delta WTWL = \sum_{i \in M} w_i \Delta l_i \quad (23)$$

where M is a set of net that are connected to cells moved.

3.3 Detailed Placement Transforms

Since detailed placement heuristics are well studied [7] [12] [13] [20] [21] [22] [23] and relative easy to understand, we only briefly introduce the techniques used in our placer, which is an industry strength placer. Again, the timing and electrical constraints can be directly used in any detailed placement framework which uses evaluate-and-execute approach and may be modified to be used in others which use model based approaches.

There are four transforms used in our placer: $SWAP$, $MOVE$, $COMPACT$ and $CENTER$. The move procedures for $SWAP$, $MOVE$, $COMPACT$ and $CENTER$ are shown in algorithm 1, 2, 3 and 4. $SWAP$ swaps two cells within a local window; $MOVE$ moves a cell within a local window; $COMPACT$ moves cells on the boundary of a net inside, which compacts the net bounding box; $CENTER$ moves a cell to the center of its connected nets.

Algorithm 1 $SWAP$ detailed placement transform

for all movable cell A **do**
  select a window W (5 circuit row high and wide) surrounding cell A
  for all movable cell $B \in W$ **do**
    swap A and B

Algorithm 2 $MOVE$ detailed placement transform

fill the empty space on the chip with minimum size pseudo cells
for all movable cell A **do**
  select a window W (5 circuit row high and wide) surrounding cell A
  for all cell or pseudo cell $B \in W$ **do**
    move A to the place of B

Algorithm 3 $COMPACT$ detailed placement transform

for all net A **do**
  for all movable cell B attached to net A's bounding box **do**
    move B toward the center of bounding box

The objective of these transforms is to reduce the weighted total wirelength WTWL. During each transform, it will recursively move one or multiple cells according to transform guidelines. The placement after these moves might not be legal, thus the transform will also legalize the placement by sliding cells along the circuit row. After legalization, the transform has produced a new legal placement.

Since the differential delay/slew modeling is not based on actual routing, it is possible that it becomes inaccurate when cells move a long distance especially for cells connected to high fanout nets. Therefore, to accurately estimate delay/slew on critical nets, we impose a maximum move limit on those cells connected to critical nets with high fanout (i.e. fanout > 4) to prevent them from moving too far away to make the differential delay/slew modeling invalid.

3.4 Algorithm

The complete description of Hippocrates detailed placement algorithm is give in algorithm 5. Given a legal placement after physical synthesis or ECO, we first run static timing analysis to get slack and delay for each timing point. Then we compute delay constraints coefficients and netweights and give them to detailed placer. The detailed placer performs transforms such as $SWAP$, $COMPACT$, $MOVE$, and $CENTER$ one by one. Each transform will make many moves. After each move and the legalization following the

Algorithm 4 $CENTER$ detailed placement transform

for all movable cell A **do**
find the set of cells connected to A
move A to the geometric center of that set of cells

move, one or multiple cells have moved, and the nets connecting those cells might have changed. The increments of HPWLs of those nets and Manhattan distances of any source-sink pairs on those nets can be computed easily with new cell coordinates. Based on these data, it then evaluates the timing and electrical constraints, weighted wirelength and movement distance. Moves that reduce WTWL while satisfying timing and electrical constraints and move limit will be accepted, others will be rejected and cell locations before those moves will be restored. Note that we do not call static timer during the detailed placement.

Algorithm 5 Hippocrates Detailed Placement

Inputs: legal placement after physical synthesis or ECO
perform static timing analysis
compute γ, θ for global wires, and α, ζ for each output pin and β, η for each input pin or PO based on (19).
compute net weight w_i based on (22) for each net i
assign move limit for a set of cells C that connect to critical nets with fanout > 4
for all detailed placement transforms $SWAP$, $MOVE$, $COMPACT$ and $CENTER$ **do**
while transform not finish **do**
move cells according to this transform
slide move cells to remove overlaps
identify a set of net M that is changed by these movements
compute HPWL change Δl_i for every net $i \in M$
compute Manhattan distance change $\Delta l_{k,m}$ between each source-sink gate pair k, m of net i, where k is the source gate and m is the sink gate
compute weighted total wirelength increment ($\Delta WTWL$) with (23)
if any of (20) is violated for any net $i \in M$
OR $\Delta WTWL \geq 0$
OR $movement > limit$ for cells in C **then**
Reject these movements, restore original cell locations
else
Accept these movements

4. EXPERIMENTAL RESULTS

We have implemented the constrained detailed placement algorithm in C on Linux machines. Instead of using MCNC or ISPD benchmarks, we select a few state-of-art industry circuits for better timing accuracy and latest technologies. The testcases use technologies ranging from 65nm to 130nm. The clock frequencies are multi GHz for 65nm and multi hundreds MHz for other technologies. The size and technology for each testcase is reported in Table 1. These testcases are the outputs of an industry strength physical synthesis tool which did timing driven placement, physical aware logic transforms, gate sizing and buffering, therefore the timing can be considered extensively optimized at this point. The total negative slack (TNS) and worst negative slack (WNS) of these testcases are also reported in Table 1.

To demonstrate that our Hippocrates style constrained detailed placer (Hipp) does really "do no harm", we compare it with two other detailed placers that use the same detailed placement transforms as (Hipp) uses. One is regular wirelength driven detailed placer (DP); the other is a naive timing driven detailed placer (TDP), which uses the exact same netweight as Hipp does but without constraints.

Table 2 shows the total negative slack (TNS) comparison of original placement (Base) and those after DP, TDP and Hipp. We highlight those cases where TNS did not degrade from baseline. We can

Table 1: Design size, technology and initial timing

designs	cells	tech(nm)	TNS(ns)	WNS(ps)
ckt1	3.8K	65	-4.048	-42
ckt2	3.9K	65	-0.575	-20
ckt3	4.4K	65	-2.447	-50
ckt4	6.0K	65	-14.011	-64
ckt5	7.5K	65	-0.753	-18
ckt6	64K	130	-452	-409
ckt7	295K	90	-83	-97
ckt8	445K	90	-631	-915

see that Hipp improves the TNS on all the testcases with an average of 37% improvement, while both DP and TDP degrade the TNS a lot. Although TDP can improve the TNS of several testcases, on average it still degrades the TNS a lot. The 37% TNS improvement from Hipp is a big improvement considering that the original placement is after extensive physical synthesis optimizations. The fact that TDP can not lower the TNS as Hipp does strongly demonstrates that it is the Hippocrates timing constraints that help prevent timing from degrading.

Table 2: TNS(ns) comparison of DP, TDP and Hipp

testcases	Base	DP	TDP	Hipp
ckt1	-4.048	-4.418	-5.229	**-3.822**
ckt2	-0.575	-1.337	-0.832	**-0.416**
ckt3	-2.447	-4.821	**-0.846**	**-1.76**
ckt4	-14.011	-14.46	**-13.05**	**-12.31**
ckt5	-0.753	-2.776	**-0.706**	**-0.229**
ckt6	-452	-681	**-415**	**-408**
ckt7	-83	-1486	-2805	**-33**
ckt8	-631	-3202	-3707	**-108**
Average		-332%	-469%	**37%**

Table 3 reports the worst negative slack (WNS) for both DP, TDP and Hipp. The testcases where WNS did not degrade from baseline are also highlighted. We can see that Hipp can keep or improve WNS on all testcases, while DP and TDP degrade it a lot. TDP is slightly better than DP, but both made timing worse. Table 3 also shows the longest path delay change after detailed placements. On average, Hipp reduces the longest path delay by 0.4%, while DP increase it by 33% and TDP by 26%. We also verified that Hipp does not introduce any additional electrical violation while DP and TDP both introduce significant amount of violations.

Table 3: WNS(ps) comparison of DP, TDP and Hipp

testcases	Base (ps)	DP	TDP	Hipp
ckt1	-42	-52	**-41**	**-41**
ckt2	-20	-73	-35	**-20**
ckt3	-50	-63	-51	**-49**
ckt4	-64	-80	-107	**-61**
ckt5	-18	-55	-67	**-18**
ckt6	-409	-620	-456	**-399**
ckt7	-97	-3372	-2828	**-79**
ckt8	-915	-2103	-1881	**-909**
Longest Path Change		33%	26%	**-0.4%**

Table 4 gives the total wirelength (TWL) comparison among DP, TDP and Hipp. Considering that the placement is already optimally placed by a global placer during physical synthesis, the improvements are significant. DP makes 6.86% improvement, TDP makes 5.89% improvement and Hipp makes 4.04% improvement. Although DP and TDP do make more TWL improvement than Hipp, the timing improvement of Hipp makes it a better candidate for post optimization applications. If we compare TDP and DP, we find that DP is doing slightly better than TDP on TWL because TDP puts higher weights on critical nets while DP has a uniform weight for every net. We also evaluated wiring congestion after Hipp and found the congestion is slightly better after Hipp than the original placement.

One can also run Hipp consecutively and get a slightly better result. Table 5 shows the average TNS and TWL improvement per-

Table 4: TWL (x10^6) comparison of DP, TDP and Hipp

testcases	Base	DP	TDP	Hipp
ckt1	0.88	0.83	0.84	0.86
ckt2	0.76	0.71	0.72	0.74
ckt3	0.53	0.49	0.50	0.51
ckt4	0.64	0.59	0.60	0.62
ckt5	1.15	1.08	1.09	1.11
ckt6	12.09	11.33	11.39	11.44
ckt7	85.41	81.70	82.57	83.49
ckt8	109.55	96.53	97.78	99.61
Average		6.86%	5.89%	4.04%

centages of the Hipp iterations over the original placement. We can see that more Hipp iterations indeed made both TWL and TNS better although the improvement rate is diminishing. WNS are the same for all the iterations, thus we do not show them. Although those improvements are small, the fact that we can further reduce wirelength and improve timing by additional iterations once again proves that Hipp does no harm to timing.

Table 5: TNS & TWL improvements for Hipp iterations

iterations	1	2	3
TNS	36.98 %	37.55 %	37.59%
TWL	4.04 %	4.27 %	4.32 %

The runtime comparison of DP, TDP and Hipp is reported in Table 6. Hipp only took half an hour on a half-million-gate design. Although Hipp is slower than DP and TDP due to additional constraints validation, it is still fast enough to be easily integrated in an industry back end design flow.

To speedup detailed placement process, we have also implemented a simplified Hipp algorithm (SHipp) to speedup the runtime. It ignores the incremental wire delay during constraints computation, which means γ, θ, β, and η (20) are zero. The reasoning for such simplification is: the wire RC delay/slew is relatively less than the gate dealy/slew (gate delay includes wire load) on critical paths after physical synthesis. Therefore, ignoring the incremental wire delay/slew has less impact on the accuracy of entire constraint computation provided that the constraints are already conservative. Our preliminary result on SHipp shows significant speed up than Hipp, almost comparable to regular DP. Same timing and wirelength performance on ckt6, ckt7, and ckt8 as Hipp, and a little worse result on ckt1-ckt5 because ckt1-ckt5 are 65nm design which has larger wire delays to gate delays ratio.

Table 6: Runtime (s) comparison of DP, TDP and Hipp

testcases	DP	TDP	Hipp
ckt1	4	4	10
ckt2	5	5	12
ckt3	5	5	13
ckt4	6	6	16
ckt5	7	7	20
ckt6	76	81	366
ckt7	330	350	1230
ckt8	792	870	1788

5. CONCLUSION

Following Hippocrates' "First, do no harm" principle, we devise a set of pin-based constraints for detailed placement to keep the original timing and honor electrical constraints while reducing wirelength. These constraints are essentially sufficient conditions for the path based constraints. We demonstrated that by using these constraints and weighted wirelength objective function, detailed placement can not only reduce wirelength, but also significantly improve timing. These constraints and objective function are simple to implement and can be applied to many detailed placement frameworks. Besides post-optimization detailed placement, we believe a rich set of placement transforms during physical synthesis can also use first-do-no-harm paradigm.

6. REFERENCES

[1] A. B. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 734–747, 2005.

[2] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. Int. Symp. on Physical Design*, pp. 185–192, 2005.

[3] B. Hu and M. Marek-Sadowska, "Far: Fixed-points and relaxation based placement," in *Proc. Int. Symp. on Physical Design*, pp. 161–166, 2002.

[4] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. Design Automation Conf.*, pp. 269–274, 1998.

[5] A. E. Caldwell, A. B. Kahng, and I. L.Markov, "Can recursive bisection alone produce routable, placements?," in *Proc. Design Automation Conf.*, pp. 477–482, 2000.

[6] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Proc. Int. Conf. on Computer Aided Design*, pp. 260–263, 2000.

[7] C. Sechen and A. Sangiovanni-Vincentelli, "The timberwolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. 20, pp. 510–522, 1985.

[8] W. Donath, P. Kudva, L. Stok, P. Villarrubia, L. Reddy, A. Sullivan, and K. Chakraborty, "Transformational placement and synthesis," in *Proc. Design, Automation and Test in Eurpoe*, Mar. 2000.

[9] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 865–868, May 1990.

[10] M. Berkelaar and J. Jess, "Gate sizing in MOS digital circuits with linear programming," in *Proc. European Design Automation Conf.*, pp. 217–221, June 1990.

[11] U. Brenner, A. Pauli, and J. Vygen, "Almost optimum placement legalization by minimum cost flow and dynamic programming," in *Proc. Int. Symp. on Physical Design*, pp. 2–9, 2004.

[12] S. W. Hur and J. Lilis, "Mongrel: hybrid techniques for standard cell placement," in *Proc. Int. Conf. on Computer Aided Design*, pp. 165–170, 2000.

[13] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden, "Fractional cut: improved recursive bisection placement," in *Proc. Int. Conf. on Computer Aided Design*, pp. 307–310, 2003.

[14] A. B. Kahng, I. L. Markov, and S. Reda, "On legalization of row-based placements," in *Proceedings 14th Great Lakes Symposium on VLSI*, pp. 214–219, 2004.

[15] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia, "Diffusion-based placement migration," in *Proc. Design Automation Conf.*, pp. 515–520, 2005.

[16] L. Tao, H. Ren, C. Alpert, and D. Z. Pan, "Computational geometry based placement migration," in *Proc. Int. Conf. on Computer Aided Design*, pp. 41–47, 2005.

[17] A. H. Ajami and M. Pedram, "Post-layout timing driven cell placement using an accurate net length model," in *Proc. Design Automation Conf.*, pp. 595–600, 2001.

[18] W. Choi and K. Bazargan, "Incremental placement for timing optimization," in *Proc. Int. Conf. on Computer Aided Design*, pp. 463–466, 2003.

[19] A. Chowdhary, K. Rajagopal, S. Venkatesan, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin, "How accurately can we model timing in a placement engine," in *Proc. Design Automation Conf.*, pp. 801–806, 2005.

[20] P. Ramachandaran, A. R. Agnihotri, S. Ono, P. Damodaran, K. Srihari, and P. H. Madden, "Optimal placement by branch-and-price," in *Proc. Asia and South Pacific Design Automation Conf.*, pp. 337–342, Jan. 2005.

[21] O. Faroe, D. Pisinger, and M. Zachariasen, "Local search for final placement in vlsi design," in *Proc. Int. Conf. on Computer Aided Design*, pp. 565–572, 2001.

[22] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *Proc. Int. Conf. on Computer Aided Design*, pp. 48–55, 2005.

[23] B. Halpin, N. Sehgal, and C. Y. R. Chen, "Detailed placement with net length constraints," in *Proc. of The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, pp. 22–27, 2003.

[24] H. Ren, D. Z. Pan, and D. Kung, "Sensitivity guided net weighting for placement driven synthesis," in *Proc. Int. Symp. on Physical Design*, pp. 10–17, 2004.