Detailed Placement for IR Drop Mitigation by Power Staple Insertion in Sub-10nm VLSI

Sun ik Heo*, Andrew B. Kahng†‡, Minsoo Kim‡, Lutong Wang‡ and Chutong Yang†
†CSE and ‡ECE Departments, UC San Diego, La Jolla, CA, USA
*Samsung Electronics Co., Ltd., Hwaseong-si, Gyeonggi-do, South Korea
{abk, mik226, luw002, chy136}@ucsd.edu, sunik.heo@samsung.com

***Abstract*—Power Delivery Network (PDN) is one of the most challenging topics in modern VLSI design. Due to aggressive technology node scaling, resistance of back-end-of-line (BEOL) layers increases dramatically in sub-10nm VLSI, causing high supply voltage (IR) drop. To solve this problem, pre-placed or post-placed power staples are inserted in pin-access layers to connect adjacent power rails and reduce PDN resistance, at the cost of reduced routing flexibility, or reduced power staple insertion opportunity. In this work, we propose dynamic programming-based single-row and double-row detailed placement optimizations to maximize the power staple insertion in a post-placement flow. We further propose metaheuristics to improve the quality of result. Compared to the traditional post-placement flow, we achieve up to 13.2% (10mV) reduction in IR drop, with almost no WNS degradation.**

I. Introduction

Distribution of power (VDD) and ground (VSS) through the Power Delivery Network (PDN) is extremely challenging in modern VLSI design. Back-end-of-line (BEOL) resistance increases dramatically in sub-10nm VLSI [1] [10]; this increases delay and requires additional buffers to meet timing requirements. The resulting routing increases capacitive loads and power consumption, which causes greater supply voltage (IR) drop, and requires a denser PDN layout – which causes more congestion again. In sub-10nm technologies, due to this vicious cycle, adding power mesh is not always the best mitigation of IR drop.

Insertion of *power staples*. Traditional PDN structures have power mesh on higher metal layers and power rails on one or two lower metal layers, with stacked vias in between. Insertion of *power staples* is a new technique for improving PDN robustness in sub-10nm technologies. *Power staples* are short pieces of metal connecting two or more adjacent (i.e., consecutive) VDD or VSS rails, to mitigate the IR drop. Figure 1 illustrates power staple insertions. In this example, adjacent M2 rails (VDD-VDD, or VSS-VSS) are connected by power staples in M1. Since each power staple goes across at least two cell rows, vertical track availability in the context of standard cell pins and pre-routes is crucial to achieve sufficient power staple insertion.

Present-day limitations and our approach. There are two basic types of power staple insertion flows: *pre-placement* and *post-placement*. A *pre-placement* flow inserts power staples at fixed intervals before placement. This strategy can achieve good IR drop, but creates regular hard placement blockages, and thus affects the placement flexibility. Alternatively, a *post-placement* flow inserts power staples opportunistically after placement, wherever long empty vertical segments (spanning at least two cell rows) are available. However, this approach is inherently limited by the placement solution: especially for high-utilization designs where empty tracks are usually not aligned between adjacent rows, the post-placement flow will result in fewer power staples and worse IR drop than the pre-placement flow.[1]

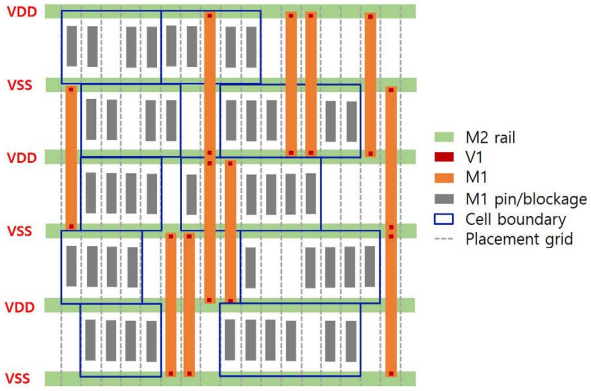

Fig. 1. Illustration of power staple insertions.

In this paper, we present new single-row and double-row *dynamic programming* (DP)-based detailed placement optimizations to improve power staple insertion in *post-placement* flows. Our contributions are summarized as follows.

- We propose a single-row dynamic programming-based approach to maximize a given objective function which comprehends the benefit from placing power staples. The proposed algorithm preserves the original cell ordering within each row.
- We propose a double- (multi)-row dynamic programming-based approach to maximize the objective function. The multi-row algorithm can more aggressively optimize the placement while keeping cells in their original rows and preserving row ordering.
- We propose a heuristic weighting function that balances the number of power staples for VDD and VSS.
- We perform extensive studies on the sensitivity and scalability of our algorithms, and show up to 13.2% IR drop reduction compared to a *post-placement* flow without our optimization.

[1]This is not an apples-to-apples comparison: the pre-placement flow inherently opts for PDN robustness over layout density, and cannot achieve very high utilizations.

978-3-9819263-2-3/DATE19/©2019 EDAA

II. Previous Work

Power staples and IR drop-aware placement. Power staple insertion is introduced in several works. [9] proposes a design-technology co-optimization (DTCO) framework with power staple insertion to mitigate IR drop. [11] introduces a "jumper" connection between vertically separated power rails to mitigate both dynamic IR drop and electromigration. [7] shows cell architectures using power staples, with both pre- and post-placement methodologies. Other works address IR drop during placement, e.g., [5] and [6] use analytic placement. [2] calculates current and voltage drop based on an admittance matrix, and proposes a cost function to guide placement perturbation.

Graph- and DP-based placement. Graph- and DP-based optimizations are widely used in detailed placement targeting various objectives. [4] proposes a series of methods in ordered single-row placement using a shortest-path algorithm to minimize perturbation or wirelength. [3] describes an optimal single-row and double-row DP optimization to address the *neighbor diffusion effect*. Their multi-row DP supports double-height cells, with cell relocation, reordering and flipping. [8] describes an ordered double-row placement with support of multi-height cells. A heuristic chain move algorithm is used to further improve the wirelength.

In summary, the works of [3] [4] [8] propose graph or dynamic programming models targeting various problems in detailed placement. They also support various cell movements and cell heights, which are not necessarily needed to optimize power staple insertion. Our work is distinguished in that (i) we formulate a single-row and double-row dynamic programming-based approach to maximize benefit functions that include the number and length of power staples; (ii) our double-row DP algorithm makes progress **site by site** instead of cell by cell as in [3], which allows precise calculation of power staple benefits (e.g., empty tracks available across multiple cell rows) per track; and (iii) we show the benefits of our optimization according to both #staples as well as IR drop in a *post-placement* power staple insertion flow.

III. Our Approach

A. Single-Row Optimization

Single-Row Optimization Problem: Given an initial legalized single-row placement and benefit table of power staples, perturb the placement to maximize power staple benefits to upper and lower rows while preserving the original cell ordering and placement legality.

For single-row dynamic programming, we place one cell at a time until all cells have been placed. Table I shows notations in our formulation. For each cell c_k, cell index k is that cell's position in the row, from left-to-right. Given a cell set C, the leftmost cell is c_1 and the rightmost is $c_{|C|}$.

For each cell, x_{c_k} is the location of cell c_k in the original placement. The *displacement range* $[-x_\Delta, x_\Delta]$ means that we cannot place a cell more than x_Δ sites away from the original location. Thus, cell c_k can only be placed in the interval $[x_{c_k} - x_\Delta, x_{c_k} + x_\Delta]$.

We use a 2D array $d[i][j]$ to represent the best-to-now solution (benefit) when we have placed cell c_i at location $x_i + j$. Thus, we obtain the best solution when we have placed all $|C|$ cells. The array is of size $(|C|+1) \times (2 * x_\Delta + 1)$ where we initialize all $d[0][j]$ to have an initial benefit of zero. Since cell ordering is preserved, the best result can be found by simply finding the most beneficial solution across all $d[|C|][j]$.

Algorithm 1 Single-Row Optimization

```
1: Initialize for all legal (i, j)
2:    d[i][j] ← −∞, d[0][j] ← 0
3: for i = 0 to |C| do
4:     for j = −x_Δ to x_Δ do
5:         for all (j') ∈ getNext(i, j) do
6:             i' = i + 1
7:             t ← d[i][j] + getBenefit(x_{c_i} + w_{c_i} + j, x_{c_{i'}} + w_{c'_i} + j' − 1) -
               α · disp(j')
8:             d[i'][j'] ← max(d[i'][j'], t)
9: finalBenefit ← max(d[|C|][j])
10: Return finalBenefit
11: Procedure getNext
12: Input: cell i and displacement j of cell c_i
13: Initialize nextList ← ∅
14: for j' = −x_Δ to x_Δ do
15:     i' = i + 1
16:     if inbound(x_{c_{i'}} + j') and x_{c_{i'}} + j' ≥ x_{c_i} + j + w_{c_i} then
17:         nextList ← j'
18: Return nextList
```

Algorithm 1 describes our dynamic programming optimization. Lines 1-2 initialize all DP array entries to have a benefit of negative infinity, except that all $d[0][j]$ are initialized to have an initial benefit of zero. Lines 3-8 describe the main DP algorithm. The algorithm places each cell one at a time, until all cells are placed. From each current placement of i cells with cell c_i at $x_i + j$, we try to place cell c_{i+1} at all legal sites j'. Function $getNext(i, j)$ gets results for all legal sites j' so that cell c_{i+1} does not overlap with any previous cells. The benefit is updated incrementally using the $getBenefit$ function. To encourage smaller displacement for each cell compared to its initial placement location, we subtract a displacement cost with a displacement factor α. $d[i'][j']$ is updated whenever the current solution is better in terms of t. Line 9 gets the best single-row placement solution across all $d[|C|][j]$. We also store a pointer for each entry of the DP array so that the algorithm can trace back from $d[i'][j']$ to $d[i][j]$, from which $d[i'][j']$ gets updated. In procedure $getNext$ of Algorithm 1, Line 13 initializes the legal location for cell $c_{i'}$ to be an empty list. Lines 14-17 add each available legal displacement j to the list.

The procedure $getBenefit$ takes the left and right coordinates of cell $c_{i'}$ and incrementally calculates the new power staples within this range. Since power staples should be at least two row heights, a simple benefit table could be as follows: a power staple length of one row height gets a benefit of zero; a power staple length between two and ten row heights gets a benefit of one (i.e., the power staple's length is extended by the two row-height stapling increment in the current cell row); and a power staple length larger than 10 row heights does not get any benefit (i.e., we do not encourage power staples longer than 10 row heights). Further discussion of the benefit table is given in Section IV-C.

B. Double-Row Optimization

We present our problem statement and double-row (extendible to multi-row) dynamic programming-based detailed placement.

TABLE I
NOTATIONS.

Notations	Meaning
C	Set of cells in a window of the initial placement.
c_k	k^{th} cell in the left-to-right ordered initial placement.
x_k	Initial x coordinate of the cell c_k. We define $x_0 = 0$ for convenience.
$[-x_\Delta, x_\Delta]$	Horizontal displacement range of the cells.
w_k	Width of the cell c_k. We define $w_0 = 0$ for convenience.
$[1, x_{max}]$	Valid sites for cell placement, starting from 1.
$d[i][j]$	The maximum value of the benefit function of the staples that can be placed when the cell c_i is at coordinate $x_{c_i} + j$. i ranges from 0 to n and j ranges from $-x_\Delta$ to x_Δ.
Additional notations for multi-row DP	
c_{rk}	The k^{th} cell, in left-to-right order, in row r.
w_{rk}	Width of the k^{th} cell in the left-to-right order in row r.
$D[i][s_{1i}][l_1][s_{2i}][l_2]$	The maximum value of the benefit function of the staples that can be placed from coordinate 1 to coordinate i, knowing that the nearest cell on the first row is $c_{s_1 i}$ at coordinate $i - l_1$ and the nearest cell on the second row is $c_{s_2 i}$ at coordinate $i - l_2$. Note that $s_r i = 0$ means that no cell in row r is placed.

Double-Row Optimization Problem: Given an initial legalized double-row placement and benefit table of power staples, perturb the placement to maximize power staple benefits to upper and lower rows without inter-row relocation while preserving the original cell ordering and placement legality.

Unlike our single-row dynamic programming algorithm, our double-row dynamic programming algorithm makes progress site by site instead of cell by cell. We use an array $D[i][s_1][l_1][s_2][l_2]$ to represent the solution and benefit up to site i. Similar to single-row optimization, where $d[i][j]$ represents a solution with $(i-1)$ cells placed on the left of $x_i + j$, and cell c_i placed at $x_i + j$, here, a valid solution up to site i may have cells placed on the left of site i, or crossing site i. s_1 (resp. s_2) indicates the last placed cell c_{s_1} (resp. c_{s_2}) in row 1 (resp. row 2), and l_1 (resp. l_2) indicates the distance between the left boundary of c_{s_1} (resp. c_{s_2}) and site i.

Figure 2 gives an example. The long vertical line indicates the current site i, and we have placed one cell in row 1, and two cells in row 2. From the figure, the left boundary of the last placed cell in row 1 has a distance of four to site i, and the left boundary of the second placed cell in row 2 has a distance of two to site i. Thus, such a solution is represented by DP array entry $D[i][1][4][2][2]$.

To progress from $D[i][s_1][l_1][s_2][l_2]$ to $D[i+1]$, we must: (i) choose whether to place new cells in the two rows, and thus increase l_1 and/or l_2; and (ii) update the benefit incrementally by adding the new power staples at site $i+1$ and subtracting any displacement cost of the new cells. Note that since we go over all sites, assuming we make progress from site i to $i+1$, we can limit the new cell placement (with its left boundary) to be exactly at site $i+1$ without sacrificing the solution space. A new cell placement at sites other than $i+1$, e.g., $i'+1$, will be handled when we make progress from i' to $i'+1$.

Algorithm 2 describes our double-row optimization. Lines 1-3 initialize the DP array entry to have a negative infinity benefit, except that all $D[0]$ entries have an initial benefit of zero. For the current DP entry $D[i][s_1][l_1][s_2][l_2]$ in Line 5, Lines 6-8 update the corresponding DP entry at site $i' = i+1$ for all legal $(i', s_1', l_1', s_2', l_2')$. The displacement function checks whether there are newly placed cells, and will only count the displacement for the newly placed cells. Then the final solution is obtained by finding the most beneficial solution when we go

Algorithm 2 Double-Row Optimization

```
1: Initialize for all (i, s_1, l_1, s_2, l_2)
2:     D[i][s_1][l_1][s_2][l_2] ← −∞
3:     D[0][s_1][l_1][s_2][l_2] ← 0
4: for i = 0 to x_max do
5:     for all (s_1, l_1, s_2, l_2) do
6:         for all (i', s_1', l_1', s_2', l_2') ∈ getNext(i, s_1, l_1, s_2, l_2) do
7:             t ← D[i][s_1][l_1][s_2][l_2] + getBenefit(i', s_1', l_1', s_2', l_2') − α ·
               disp(s_1, s_1', l_1', s_2, s_2', l_2')
8:             D[i'][s_1'][l_1'][s_2'][l_2'] ← (max(D[i'][s_1'][l_1'][s_2'][l_2'], t)
9: finalBenefit ← max(d[x_max][|c_1|][l_1][|c_2|][l_2])
10: Return finalBenefit
11: Procedure getNext
12: Input: i, s_1, l_1, s_2, l_2
13: Initialize nextList ← ∅
14: i' ← i + 1
15: Only place a cell on the first row
16: s_1' ← s_1 + 1, l_1' ← 0, s_2' ← s_2, l_2' ← l_2 + 1
17: if legalPlace(s_1, l_1, s_1') then
18:     nextList.add ← (i', s_1', l_1', s_2', l_2')
19: Only place a cell on the second row
20: s_1' ← s_1, l_1' ← l_1 + 1, s_2' ← s_2 + 1, l_2' ← 0
21: if legalPlace(s_2, l_2, s_2') then
22:     nextList.add ← (i', s_1', l_1', s_2', l_2')
23: Place cells on both rows
24: s_1' ← s_1 + 1, l_1' ← 0, s_2' ← s_2 + 1, l_2' ← 0
25: if legalPlace(s_1, l_1, s_1') and legalPlace(s_2, l_2, s_2') then
26:     nextList.add ← (i', s_1', l_1', s_2', l_2')
27: Place no new cells
28: s_1' ← s_1, l_1' ← l_1 + 1, s_2' ← s_2, l_2' ← l_2 + 1
29: nextList.add ← (i', s_1', l_1', s_2', l_2')
30: Return nextList
```

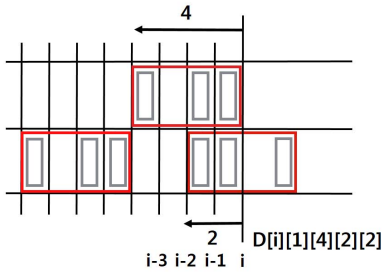

Fig. 2. The case represented by D[i][1][4][2][2].

over all sites, and all cells. In Algorithm 2, procedure $getNext$ generates all legal $(i', s_1', l_1', s_2', l_2')$. From the current DP entry, Lines 15-18 generate all legal $(i', s_1', l_1', s_2', l_2')$ when we place a cell only on the first row; similarly, Lines 19-22, 23-26, and 27-29 generate all legal $(i', s_1', l_1', s_2', l_2')$ when we place a cell only on the second row, on both rows, or on no rows, respectively.

For dynamic programming optimization of more than two rows, we can just add DP array dimensions $[s_k][l_k]$ for each added row. However, the added dimensions may grow the total runtime and memory footprint beyond practical usage. In this work, we use double-row dynamic programming for optimization.

The size of the DP array for n rows is $O(x_{max}(s_{max}(2*x_\Delta + w_{max}))^n)$, where w_{max} is the maximum cell width and s_{max} is the maximum value for dimension s_k. Experiments using our implementation reveal runtimes that are more obviously dependent on the number of *getNext()* function calls, as opposed to the size of the DP array. Our experiments also show that the number of *getNext()* function calls mainly depends on the number of cells in the optimization window and the displacement range x_Δ. This points to tuning of our DP implementation as a necessary future improvement.

C. Overall Flow

Figure 3 shows the overall optimization flow. Given a post-P&R database, we first perform our dynamic programming-based detailed placement optimization to maximize the potential number of power staples, and then perform incremental routing and Vt swapping to fix routing and timing degradation after our optimization. Next, we perform power staple insertion to complete the PDN and perform a vectorless dynamic IR drop analysis to show the results. We note that our optimization and power staple insertion could also be performed in both post-place and post-CTS stages. However, to compare the QoR with, and without, our optimization using a single database, in this work we only perform our optimization in the post-route stage, as shown in Figure 3.

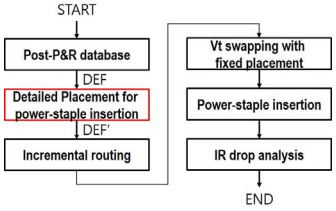

Fig. 3. Overall flow of our work. The red box indicates steps that we implement. Commercial tools [12] [13] are used for all other steps.

IV. Experiments

We implement our dynamic programming in C++ with OpenAccess 2.2.43 [15] to support LEF/DEF [14]. We perform experiments in a 7nm FinFET technology with multi-height triple-Vt libraries from a leading technology consortium.[2] We apply our optimization to Arm Cortex-M0 and three design blocks (AES, JPEG and MPEG) from OpenCores [16]. Design information is summarized in Table II.[3]

In this section, we first investigate sensitivity and scalability with respect to the displacement range. Second, we study effects of weighting factors, i.e., displacement factor and benefit table. Third, we propose a heuristic benefit table that is able to balance VDD and VSS staples. Fourth, we show experimental results across different design blocks and utilizations, with IR drop analysis. Last, we compare three examples of metaheuristics that can be implemented around our basic DP optimization. In the following experiments, we use our double-row optimization by default, and we show the comparison with single-row optimization in Section IV-D.

TABLE II
Design information.

design	#inst	clkp
M0	∼10K	500ps
AES	∼12K	500ps
MPEG	∼14K	500ps
JPEG	∼54K	500ps

[2]We use the process, voltage, and temperature conditions (TT, 0.65V, $25°C$) for both place-and-route and IR-drop analysis. We build power mesh on M7 to M9 with $0.5\mu m$ width, $15.5\mu m$ pitch and $7\mu m$ offset, and M2 power rails. We use M1 and V12 for power staples and M1 is not allowed for routing.

[3]We synthesize designs using *Synopsys Design Compiler L-2016.03-SP4* [18], and perform place-and-route using *Cadence Innovus Implementation System v16.2* [12]. We perform vectorless dynamic IR drop analysis using *Cadence Voltus IC power integrity solution* [13]. Our dynamic programming-based optimizations are performed in a single thread on a $2.6GHz$ Intel Xeon server.

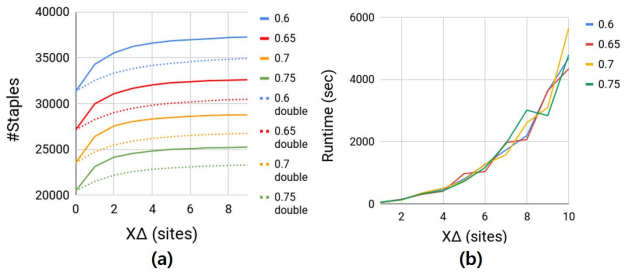

Fig. 4. (a) Sensitivity of #staples to displacement range x_Δ, and (b) sensitivity of runtime to displacement range (x_Δ).

A. Scalability/Sensitivity Study

Since the displacement range (x_Δ) determines the solution space, and also the size of the DP table data structure, we first study the sensitivity of #staples to the displacement range, and the scalability of our algorithm when we increase the displacement range. In this experiment, we sweep x_Δ from 0 to 10 with a step of 1. To show the stability of our optimization to block utilization, we use design block AES with four target utilizations: 0.60, 0.65, 0.70 and 0.75. In this experiment, since weighting factors do not affect the scalability, they have a default value of 0. The benefit table simply shows the total length of power staples that can be inserted.

Figure 4(a) shows the number of power staples vs. displacement range. The x-axis is the displacement range, and the y-axis is #staples. We can see that for values of displacement range > 5, there are diminishing returns in terms of #staples. Also, design blocks with a lower utilization tend to saturate with larger #staples, as one would expect. These data motivate future efforts to obtain a more detailed and accurate understanding of the relationship between target utilization and maximum IR drop, e.g., for use in early design exploration.[4] Note that the saturation of #staples versus displacement range depends on the library used. The dotted lines in Figure 4(a) show the (normalized) result when all widths are doubled (x-coordinates, and cell widths and pin locations in each library cell). With the wider cells, increasing the displacement range continues to give staple insertion benefits up to larger values of x_Δ.

Figure 4(b) shows the runtime vs. displacement range. The run with $x_\Delta = 9$ consumes $4\times$ runtime compared to the run with $x_\Delta = 5$, but only results in minor increase in #staples. Utilization generally does not affect the runtime, showing the robustness of our optimization. Thus, to balance the solution quality and runtime, we choose $x_\Delta = 5$ in all the following experiment.

Another study examines whether there is any benefit to breaking the optimization into multiple, more "gradual", phases. Table III shows how number of staples changes when we run two rounds of optimizations with total displacement

[4]As our experiments show, there is less available increment in the number of placed staples when the utilization is higher. Decreasing (resp. increasing) utilization reduces current density while increasing (resp. decreasing) potential staple insertion. I.e., there is a compounding of IR drop mitigation (resp., degradation) effects, which is further compounded by changes to gate sizing and voltage-induced timing margins. Deriving a principled model of this dynamic is an open challenge.

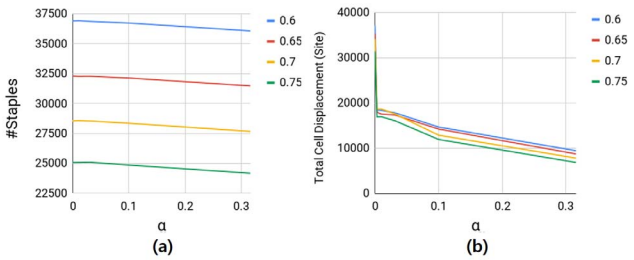

Fig. 5. Sensitivities of (a) #staples, and (b) total cell displacement, to the displacement factor (α).

TABLE III
TOTAL #STAPLES PLACED WHEN WE APPLY TWO ROUNDS OF OPTIMIZATION WITH TOTAL DISPLACEMENT RANGE (x_Δ) = 6. EACH ROW REPRESENTS DESIGN AES WITH UTILIZATION BETWEEN 0.60 AND 0.75.

Util	x_Δ					
	6	**1+5**	**2+4**	**3+3**	**4+2**	**5+1**
0.60	36974	36857	36634	36493	36611	36748
0.65	32389	32240	32079	31936	32054	32199
0.70	28624	28540	28362	28252	28287	28432
0.75	25090	25939	24867	24722	24780	24915

range (x_Δ) = 6. The outcome is worst when we perform two phases of optimization, each with displacement range (x_Δ) = 3. The apparent takeaway is that it is better to consume available displacement range "in one shot".

B. Study of Weighting Factors

Our next experiment studies the impact of the displacement factor (α). We sweep the displacement factor with value 0, and from $10^{-3.5}$ to $10^{-0.5}$, with a multiplier of $10^{0.5}$. We still use the same design block AES with four different utilizations as in Section IV-A. Figure 5(a) shows the #staples vs. displacement factor and Figure 5(b) shows the total cell displacement vs. displacement factor.

From Figure 5, with a non-zero weight, total displacement decreases sharply while #staples does not change much. This indicates that we can preserve most of #staples while reducing total cell displacement by using a non-zero α. A smaller total cell displacement is beneficial to minimize perturbation of the input layout and thus we use $\alpha = 0.01$ in all the following experiments. Note that this choice is based on our experiments, and that a different α may be preferred in a different technology node or design enablement.

C. Study of Benefit Table

We now discuss our investigations into the benefit table. First, we study the tradeoff between short and long staples, i.e., whether we should encourage fewer but longer staples, or more but shorter staples. Since our optimization changes the placement, to make a fair comparison and investigate the tradeoff, we use the AES design with pre-placed power staples at fixed intervals (20CPP) and perform IR drop analysis.[5] Each run uses a different staple length, from 2 to 10 row heights, with a step size of 2 row heights. We only use VDD power staples in this study.

[5]*CPP* = contacted poly pitch. In this work, we use a technology enablement where 1CPP = $42nm$.

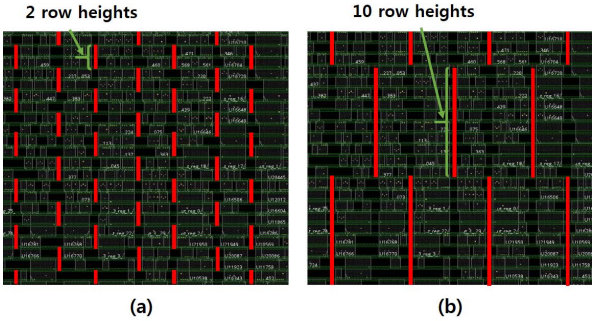

Fig. 6. Design examples with different lengths of power staples: (a) power staple length = 2 row heights, and (b) power staple length = 10 row heights.

As shown in Figure 6, the total length of all power staples is kept essentially constant across different configurations. We also ensure that the initial input placement is identical across all runs, with power staple tracks always free from pins, blockages and pre-routes. As shown in Table IV, we can see that the IR drop is not sensitive to the staple length.

TABLE IV
IR DROP VS. STAPLE LENGTH.

Staple length	2	4	6	8	10
Worst IR drop (mV)	67.9	67.8	67.7	67.7	67.6

Now we propose our heuristic benefit table setting to improve the balance between #VDD and VSS staples. Since the double-row dynamic programming is performed every two rows in a window, without overlapping (e.g., optimize 1^{st} and 2^{nd} rows, then 3^{rd} and 4^{th} rows), the optimization is always aligned with either VDD rails or with VSS rails, resulting in potentially biased power staples (e.g., staples of one rail are systematically more likely to occur than those of the other rail). To overcome this issue, we propose a reconfiguration of the benefit table so that a single-height staple gets a non-zero benefit (β), with all other staple length getting a benefit of one (up to ten row heights). Such single-height staples may get extended in the next optimization window so that there is a chance to re-balance VDD/VSS staples. We have experimentally swept β from 0 to 1.0, with a step size of 0.1, with results shown in Figure 7(a). As β increases, VDD/VSS staples become balanced without sacrificing the total #staples. We use $\beta = 0.7$ in all following experiments.

D. Metaheuristics

In this subsection, we compare three strategies: (i) single-row optimization (SR); (ii) double-row optimization with overlap (Meta-1); and (iii) two-pass double-row optimization without overlap (Meta-2). In (ii), we perform the double-row optimization every two rows with one row overlapped (e.g., we optimize row 1 and row 2, then row 2 and row 3, then row 3 and row 4, etc.). In (iii), the first pass starts at the odd rows, and then the second pass starts at the even rows (e.g., we optimize row 1 and row 2, row 3 and row 4, etc. in the first pass, and row 2 and row 3, etc. in the second pass).

Therefore, effectively each row is covered once in SR, and twice in Meta-1 and Meta-2. We show the experimental results in Figure 7(b). Compared to Baseline, where there is no detailed placement optimization, the methods SR, Meta-1 and Meta-2 respectively achieve 20.1%, 22.3% and 22.7% more inserted

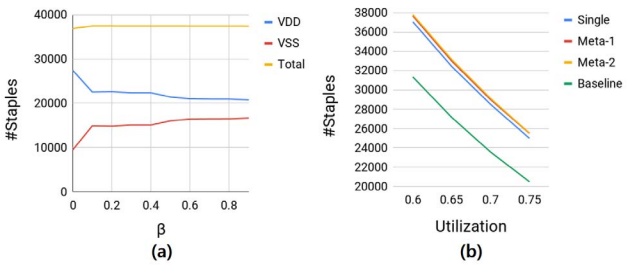

Fig. 7. (a) Number of staples (total, VDD and VSS) that can be placed, versus balance factor (β) (AES with utilization 0.60). (b) Number of staples that can be placed, versus utilization by different strategies (AES with utilization between 0.60 and 0.75).

staples, averaged over all four target utilizations. Thus, we use Meta-2 for our main experimental results (Subsection IV-E).[6]

E. Main Results

We execute our Meta-2 optimization using all design blocks with the aforementioned parameter settings, and report #staples, average and worst IR drop before and after optimization, along with Δ worst negative slack (WNS) and runtime, at four utilizations per design block. Table V shows both the initial and post-optimization values of metrics (IR drop values are in units of mV). Figure 8 shows vectorless dynamic IR drop heatmaps for AES with 0.60 utilization.

We observe that our optimization can increase #staples by anywhere from 6.2% to 24.6%, and reduce IR drop by up to 13.2%, compared to *post-placement* staple insertion without our optimization. Furthermore, ΔWNS is similar before and after optimization, showing the effectiveness and robustness of our optimization. On the downside, IR drop can increase after our optimization (e.g., the case of M0 with 0.65 utilization); this is because the design's current map changes with placement perturbation and timing recovery steps, even as we add more power staples. Hence, directions for improvement include better control of the commercial P&R tool, and/or implementing our own legalization and ECO capabilities. We also note that current runtimes are long, reflecting unoptimized implementation that pays heavily for *getNext()* function calls.

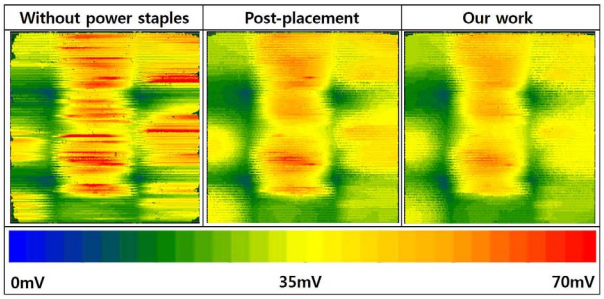

Fig. 8. Heatmaps of vectorless dynamic IR drop for AES.

[6]Note that there is generally no benefit from making multiple passes over the rows of the layout, the same solution. Our ongoing efforts seek richer combinations of metaheuristics to further improve results.

TABLE V
EXPERIMENTAL RESULTS USING DOUBLE-ROW OPTIMIZATION.

Design	Util	#Staples		IR drop (mV,Avg)		IR drop (mV,Worst)		ΔWNS	Runtime
		Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	(ns)	(sec)
M0	60%	29109	33423 (+14.8%)	29	28 (-3.4%)	76	66 (-13.2%)	-0.009	2666
	65%	25295	29254 (+15.7%)	28	28 (0.0%)	58	60 (+3.4%)	-0.016	2249.5
	70%	22233	25628 (+15.3%)	29	29 (0.0%)	77	76 (-1.3%)	-0.008	2450
	75%	19700	22889 (+16.2%)	29	29 (0.0%)	80	80 (0.0%)	-0.017	2058
AES	60%	31355	37813 (+20.6%)	23	23 (0.0%)	51	47 (-5.6%)	-0.001	2542.5
	65%	27165	33164 (+22.1%)	22	22 (0.0%)	54	51 (-5.6%)	-0.001	2127
	70%	23571	29129 (+23.6%)	22	22 (0.0%)	63	62 (-1.6%)	-0.001	3142
	75%	20496	25530 (+24.6%)	22	22 (0.0%)	71	68 (-4.2%)	-0.006	2520
MPEG	60%	75781	80951 (+6.8%)	24	24 (0.0%)	64	60 (-6.3%)	0.000	3700
	65%	67836	72685 (+7.1%)	24	24 (0.0%)	57	60 (+5.3%)	0.000	4088.5
	70%	61152	65362 (+6.9%)	25	25 (0.0%)	62	59 (-4.8%)	0.000	4410
	75%	55426	58928 (+6.3%)	30	30 (0.0%)	81	79 (-2.5%)	-0.002	3709
JPEG	60%	217467	244333 (+12.4%)	28	27 (-3.6%)	88	84 (-4.5%)	-0.001	16679
	65%	193089	218483 (+13.2%)	26	27 (+3.8%)	79	76 (-3.8%)	0.000	13869.5
	70%	171857	195781 (+13.9%)	28	28 (0.0%)	69	70 (+1.4%)	-0.007	15069
	75%	152805	174618 (+14.3%)	29	28 (-3.4%)	73	68 (-6.8%)	0.000	12563.5

V. CONCLUSIONS

We have presented novel DP-based single- and double-row detailed placement optimizations with metaheuristics to maximize power staple insertion in a post-placement flow. We perform extensive studies on the scalability and sensitivity to parameters, impact of the benefit table, as well as balance of VDD/VSS staples. Compared to the traditional post-placement flow, we achieve up to 13.2% ($10mV$) reduction in IR drop, with almost no WNS degradation compared to a pre-placement flow. Our ongoing efforts include (i) better exploration of the benefit table; (ii) understanding in greater detail the dynamics of the relationship between power staples and IR drop; (iii) further exploration of metaheuristics, (iv) runtime speedup techniques, and (v) exploration of power mesh structures with different width, pitch and offset.

ACKNOWLEDGMENT

Research at UCSD is supported by Qualcomm, Samsung, NXP Semiconductors, Mentor Graphics, DARPA (HR0011-18-2-0032), NSF (CCF-1564302) and the C-DEN center.

REFERENCES

[1] P. Besser, "BEOL Interconnect Innovations for Improving Performance", *NCCAVS Joint Users Group Technical Symposium*, 2017.
[2] J. M. Cohn, J. Venuto, I. L. Wemple and P. S. Zuchowski, "Method for Supply Voltage Drop Analysis During Placement Phase of Chip Design", *US Patent*, US6523154B2, 2000.
[3] C. Han, K. Han, A. B. Kahng, H. Lee, L. Wang and B. Xu, "Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI", *Proc. ICCAD*, 2017, pp. 667-674.
[4] A. B. Kahng, I. L. Markov and S. Reda, "On Legalization of Row-Based Placements", *Proc. GLSVLSI*, 2004, pp. 214-219.
[5] A. B. Kahng, B. Liu and Q. Wang, "Supply Voltage Degradation Aware Analytical Placement", *Proc. ICCD*, 2005, pp. 437-443.
[6] A. B. Kahng and Q. Wang, "A Faster Implementation of APlace", *Proc. ISPD*, 2006, pp. 218-220.
[7] L. Liebmann et al., "Exploiting Regularity: Breakthroughs in Sub-7nm Place-and-Route", *Proc. SPIE*, Design-Process-Technology Co-optimization for Manufacturability XI, 2017, pp. 1-10.
[8] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert and D. Z. Pan, "MrDP: Multiple-row Detailed Placement of Heterogeneous-sized Cells for Advanced Nodes", *Proc. ICCAD*, 2016, pp. 7:1-7:8.
[9] L. Mattii, D. M. Milojevic, P. Debacker, Y. Sherazi, M. Berekovic and P. Raghavan, "IR-Drop Aware Design & Technology Co-Optimization for N5 Node with Different Device and Cell Height Options", *Proc. ICCAD*, 2017, pp. 89-94.
[10] G. Yeric, "Circuit Application Requirements", *Proc. IEDM*, Short Course, 2014.
[11] C.-Y. Yu, Y.-T. Hou, C.-M. Fu, W.-H. Chen and W.-Y. Lo, "Apparatus and Method for Mitigating Dynamic IR Voltage Drop and Electromigration Affects", *US Patent*, US9768119B2, 2017.
[12] Cadence Innovus User Guide, http://www.cadence.com
[13] Cadence Voltus IC Power Integrity Solution User Guide, http://www.cadence.com
[14] LEF/DEF 5.8. http://www.si2.org/openeda.si2.org/projects/lefdef
[15] Si2 OpenAccess. http://www.si2.org/?page=69
[16] OpenCores: Open Source IP-Cores, http://www.opencores.org
[17] OpenMP Architecture Review Board, "OpenMP Application Program Interface, Version 4.0".
[18] Synopsys Design Compiler User Guide, http://www.synopsys.com