Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes

Peter Debacker[+], Kwangsoo Han[†], Andrew B. Kahng[†‡],
Hyein Lee[†], Praveen Raghavan[+] and Lutong Wang[†]
[†]ECE and [‡]CSE Departments, UC San Diego, La Jolla, CA 92093, USA
[+]imec, Leuven 3001, Belgium
{peter.debacker, praveen.raghavan}@imec.be, {kwhan, abk, hyeinlee, luw002}@ucsd.edu

ABSTRACT

Aggressive pitch scaling in sub-10nm nodes has introduced complex design rules which make routing extremely challenging. Cell architectures have also been changed to meet the design rules. For example, metal layers below M1 are used to gain additional routing resources. New cell architectures wherein inter-row M1 routing is allowed force consideration of vertical alignment of cells. In this work, we propose a mixed-integer linear programming (MILP)-based, detailed placement optimization to maximize direct vertical M1 routing utilization for congestion and wirelength reduction.

1. INTRODUCTION

In tandem with aggressive pitch scaling in sub-10nm technology nodes, the detailed routing problem has become extremely challenging. Routing today must deal with large numbers of complex design rules that are driven by patterning technologies – notably, self-aligned multiple patterning and line-end cut on minimum-pitch metal layers, as well as contact- and via-layer patterning. The quest to scale "PPAC" (power, performance, area, cost) has led to a very delicate balancing act among power delivery, routing resource, and resistivity in middle-of-line (MOL) and local metal layers.

To address these challenges, the industry has seen rapid innovation in standard-cell architecture starting at the foundry 10nm ("N10") node, and accelerating into the N7/N5 enablement. As examples of cell architecture evolution, metal layers below M1 are used for internal routing within a standard cell, or horizontal M1 power/ground pins are removed to gain additional routing resources for inter-cell routing. These new cell architectures, wherein *inter-row M1 routing is allowed*, force new consideration of vertical alignment of cells.

1.1 New Cell Architectures in Sub-10nm

Figure 1 illustrates inverter (INV) layout in three types of cell architectures: (a) conventional 12-track, (b) *ClosedM1* 7.5-track, and (c) *OpenM1* 7.5-track. The conventional 12-track INV has power/ground (VDD/VSS) in M1, which prevents use of vertical M1 routing for pin access. In other words, with the conventional cell architecture, pin access is available only with M2 routing. However, in sub-10nm nodes, where metal layers below M1 are used for internal cell routing, the M1 layer can be used for pin access as well as for routing with both the *ClosedM1* and *OpenM1* cell architectures.

***ClosedM1* standard cell architecture.** A *ClosedM1* standard cell has 1D vertical M1 pins, including VDD/VSS pins, as shown in Figure 1(b). The M1 VDD/VSS pins at the left and right boundaries of the cell are connected to M2 VDD/VSS pins at the top and bottom boundaries by using via V12. In this way, VDD/VSS pins do not block inter-row M1 routing. Also, due to the design rules for self-aligned multiple patterning (SAMP), the M1 pins in *ClosedM1* have 1D shapes and are regularly placed with a fixed pitch. In particular, the *ClosedM1* cell library that we use in this work has M1 pitch equal to the width of a placement site. Therefore, if we vertically align pins of given net, these pins can be connected by a small M1 segment with negligible routing cost or overheads. Figure 2(a) illustrates an example of direct vertical M1 routing (dM1) between two INVs. Here we define a direct vertical M1 routing as a (sub)net routing using only one M1 routing segment. Importantly, even though the *ClosedM1* cell architecture enables inter-row M1 routing, the realized power/performance/area (PPA) benefit from M1 routing may not be significant unless a router can effectively exploit the availability of direct vertical M1 routing. This is because M1 routing tracks are blocked by M1 pins, and the inter-row M1 routing can be used only when two pins are sufficiently aligned. Thus, *both* the detailed placer and the router must comprehend vertical alignment in order to maximally exploit direct vertical M1 routing for *ClosedM1*-based designs.

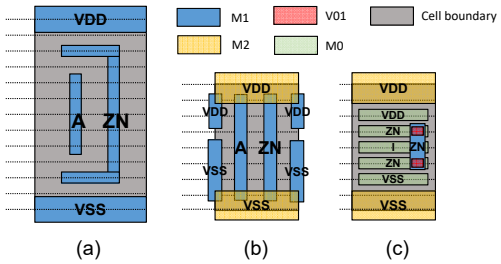

Figure 1: New cell architectures to gain additional routing resources. (a) Conventional 12-track INV; (b) *ClosedM1* 7.5-track INV; (c) *OpenM1* 7.5-track INV.

***OpenM1* standard cell architecture.** At sub-10nm nodes, the *OpenM1* standard cell architecture is introduced to enable more M1 routing resource than with the *ClosedM1* architecture. For *OpenM1* cells, M1 routing is "open" since most of the pins are on the M0 layer, which is a complementary layer below the M1 layer. As shown in Figure 1(c), the I, ZN, VDD, VSS pins have horizontal M0 segments, and an M1 segment connects two M0 segments for the ZN pin. We note that there is no connection between M0 and M2 segments for VDD/VSS pins. Thus, M1 routing for VDD/VSS pins must be accomplished with a special structure for the power distribution network.[1] In terms of signal routing, if two pins are overlapped horizontally (i.e., their projections onto the x-axis intersect), direct vertical M1 routing can be used to connect them. Figure 2(b) shows a direct vertical M1 routing between the ZN pin of the upper INV and the I pin of the lower INV. As long as the ZN and I pins are overlapped horizontally, the two pins can be connected using a single vertical M1 segment along with two V01 vias.

Compared to both the conventional and the *ClosedM1* cell architectures, *OpenM1* effectively enables an additional metal layer for routing, which can have considerable routability benefits. Furthermore, unlike with the sub-10nm *ClosedM1* architecture, conventional P&R tools can easily find benefits from *OpenM1* without any special optimization to maximize M1 routing. This being said, below we explore the question of whether there might still be room (beyond the current state of the art in commercial P&R tooling) to optimize for better pin accessibility in *OpenM1*-based designs, given that pins are horizontal. For instance, by maximizing "overlap" between pins in a net, we might induce a router to use more direct vertical M1 routing between pins, which would reduce usage (blockage) and detouring on upper layers (M2, M3, etc.). In Subsection 5.2, we report experimental results with and

[1]For example, vertical M1 segments must be inserted with a fixed pitch to staple M2 and M0 VDD/VSS pins.

DAC'17, June 18-22, 2017, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/3061639.3062338.

without a detailed placement optimization that maximizes pin overlaps for *OpenM1*.

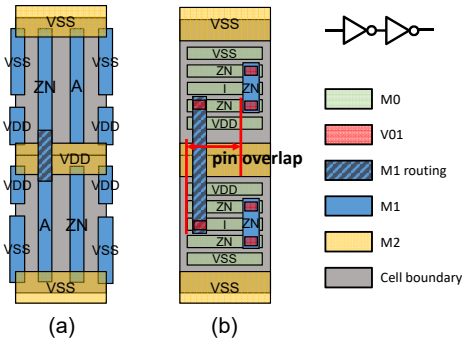

Figure 2: Direct vertical M1 routing examples: (a) *ClosedM1*, (b) *OpenM1*.

1.2 This Work

In this work, we propose a vertical M1 routing-aware detailed placement optimization based on mixed-integer linear programing (MILP) for two new sub-10nm cell architectures, i.e., *OpenM1* and *ClosedM1*. We note that the vertical M1 routing-aware detailed placement is a completely different problem from traditional wirelength-driven detailed placement, in the sense that the routing cost is non-monotonic due to vertical M1 routing, which is almost "free". Our MILP formulation enables exploration of the tradeoff between minimization of the traditional half-perimeter wirelength (HPWL) objective and maximization of the number of vertical pin alignments (= potential direct pin-pin routings using vertical M1) via a weighting factor (α). Below, we specifically study the impact of α on routed wirelength. The main contributions of our work are summarized as follows.[2]

- We propose an MILP-based detailed placement optimization for two cell architectures that are relevant in sub-10nm process nodes, to consider and exploit (direct vertical) inter-row M1 routing.
- We propose a distributable window-based optimization to overcome the runtime limitation of the MILP-based approach.
- We implement our proposed approach in C++ with OpenAccess 2.2.43 [19] and incorporate it into a commercial tool-based placement and routing (P&R) flow. The results from our approach are evaluated using a commercial tool flow.
- We explore various metaheuristic configurations (optimization degrees of freedom, window size, iteration strategy, etc.) and study impacts on runtime and solution quality.

The remainder of this paper is organized as follows. Section 2 reviews related previous works. In Section 3, we describe our MILP formulations for detailed placement optimization considering direct vertical M1 routing. In Section 4, we explain our overall optimization metaheuristic, centered around a distributable window-based optimization. Section 5 provides experimental results and analysis. We give conclusions and future research directions in Section 6.

2. PREVIOUS WORK

We classify relevant previous works on detailed placement and placement legalization into three categories: (i) dynamic programming-based approaches, (ii) graph model-based approaches, and (iii) MILP-based approaches. Our present work is most closely related to the third category.

Dynamic programming-based approaches. Dynamic programming (DP) has been a popular framework, particularly for row-based detailed placement, for many years. Kahng et al. [5] propose a HPWL-driven ordered single-row detailed placement with free sites. Gupta et al. [2] propose a DP-based single-row placement optimization to enable sub-resolution assist feature insertion for improved manufacturability. Subsequent work addresses a 2D formulation [3], using DP in which vertical and horizontal costs are calculated with restricted perturbations. Hur and Lillis [8] propose a DP-based *optimal interleaving* for intra-row optimization in detailed placement. For double-patterning-aware detailed placement, Gupta et al. [1] propose a DP-based algorithm that solves coloring conflicts while minimizing the displacement of timing-critical cells.

Graph-based approaches. A literature of graph model-based approaches typically formulates placement optimization as a shortest-path computation in an appropriate directed graph. Kahng et al. [6] legalize placement of a single row with various minimization objectives, by calculating a shortest path in a directed acyclic graph constructed from the input ordering of cells. The work of [13] proposes a triple-patterning-aware detailed placement using a graph model. The authors formulate a graph to determine cell locations as well as coloring solutions for a single row placement. Du and Wong [7] address the abutment of source and drain in FinFET-based cell placement. The authors propose a graph model that captures cell flipping and adjacent-cell swapping as underlying operations for detailed placement perturbation. A shortest-path algorithm then minimizes the cost induced from the source-drain abutment. Lin et al. [12] propose a graph-based detailed placement to resolve inter-row middle-of-line conflicts. Similar to [7], a graph is constructed to handle cell flipping, swapping and shifting operation for *local reordered single row refinement*.

MILP-based approaches. While DP-based and graph model-based approaches are efficient for single-row placement, it is not easy to handle multiple-row placement optimizations (specifically, in the context of this work, vertical M1 routing-aware placement) with these approaches due to interaction between vertically adjacent cells. However, several mixed integer-linear programming (MILP)-based approaches have been proposed which handle both single-row and multiple-row placement. Lin and Chu [11] formulate a MILP for triple-patterning-aware detailed placement. The MILP is used to assign a coloring solution for each standard cell and determine the location of each cell in a single row, while minimizing placement perturbation and coloring conflicts. Li and Koh [9] propose MILP-based detailed placement approaches using *single-cell-placement (SCP) variables*. The SCP variables correspond to locations, orientations as well as placement sites of each cell. The MILP determines the best SCP variable for each cell. The same authors' extension [10] supports mixed-size circuits and improves runtime by bounding solution spaces. Han et al. [4] adopt the MILP model of [9] [10] and extend it to support N10-relevant design rules. Further, a distributable optimization is proposed based on partitioning of the layout into windows that can be independently legalized. In our present work, we use a similar strategy as the work of [4], extending it to handle vertical M1 routing for new cell architectures in sub-10nm. Overall, our work is distinguished from previous (MILP-based) approaches in that (i) we formulate inter-row cell alignment to maximize direct vertical M1 routing, which has not been addressed in previous works, and (ii) we improve the distributable optimization of [4] by a smart selection of target windows along with a metaheuristic strategy.

3. MILP-BASED OPTIMIZATION

In this section, we give our problem statement, followed by MILP formulations for vertical M1 routing-aware detailed placement optimization with two sub-10nm cell architectures, *OpenM1* and *ClosedM1*.

Vertical M1 Detailed Placement
Given: a post-routed placement, and per-cell placement perturbation range.
Perform: Perturb the input placement to optimize a weighted sum of (minimized) HPWL and (maximized) inter-row pin alignments, while satisfying cell location perturbation bounds and placement legality constraints.

3.1 MILP Formulation for ClosedM1

We formulate an MILP for our detailed placement problem for the *ClosedM1* cell architecture. In the following, we use notation as described in Table 1.

[2]The MILP formulation will differ according to the standard cell template and layer directionality. However, our distributable optimization and exploration of metaheuristic configurations can apply with any technology.

Table 1: Notations.

Notation	Meaning
d_{pq}	a binary indicator of whether pins p and q are aligned (*ClosedM1*) or overlapped (*OpenM1*)
w_n	half-perimeter wirelength (HPWL) of net n
α	a weighting factor for direct vertical M1 routing
β_n	a weighting factor for HPWL of net n
C, R, Q	sets of cells, rows, columns (placement sites)
N	set of nets
$x(y)_{min,n}$ $x(y)_{max,n}$	minimum x (y) and maximum x (y) coordinates of net n
P_n	set of pins in net n
G	a large positive constant number
H	placement row height
$x_c(y_c)$	x (y) coordinate of the center of cell c
$x_p(y_p)$	relative x (y) coordinate of pin p to its owner cell's x (y) coordinate
$x_{min,p}$ ($x_{max,p}$)	minimum (maximum) x coordinate of pin p relative to its owner cell's x coordinate
f_c	a binary indicator of whether cell c is flipped
s_{crq}	a binary indicator of whether cell c occupies site (r, q)
K_c	a set of candidates of cell c
λ_c^k	a binary indicator of whether candidate k for cell c is selected
$x_c^k(y_c^k)$	x (y) coordinate corresponding to λ_c^k
f_c^k	f_c corresponding to λ_c^k
s_{crq}^k	s_{crq} corresponding to λ_c^k
γ	maximum allowed length for a direct vertical M1 routing (unit: number of placement rows)
v_{pq}	a binary indicator of whether pins p and q are within a given range (γ) in y direction
o_{pq}	length of overlap between pins p and q
δ	minimum required overlap length for direct vertical M1 routing
ϵ	a weighting factor for the sum of overlap lengths (o_{pq})

$$\textbf{Minimize:} \quad -\alpha \cdot \sum d_{pq} + \sum_{n \in N} \beta_n \cdot w_n \tag{1}$$

Subject to:

$$w_n = x_{max,n} - x_{min,n} + y_{max,n} - y_{min,n}, \quad \forall n \in N \tag{2}$$

$$x_{max,n} \geq x_c + x_p, \quad x_{min,n} \leq x_c + x_p$$
$$y_{max,n} \geq y_c + y_p, \quad y_{min,n} \leq y_c + x_p$$
$$\forall p \in P_n, \text{ where } c \text{ is the owner cell of pin } p \tag{3}$$

$$(x_c + x_p) - (x_{c'} + x_q) \leq G(1 - d_{pq})$$
$$(x_c + x_p) - (x_{c'} + x_q) \geq -G(1 - d_{pq})$$
$$(y_c + y_p) - (y_{c'} + y_q) \leq G(1 - d_{pq}) + H$$
$$(y_c + y_p) - (y_{c'} + y_q) \geq -G(1 - d_{pq}) - H$$
$$\forall (p, q) \text{ in } n, \text{ where } c, c' \text{ are owners of pins } p, q \tag{4}$$

$$\sum_{k \in K_c} \lambda_c^k = 1, \quad \forall c \in C \tag{5}$$

$$f_c = \sum_{k \in K_c} f_c^k \lambda_c^k, \quad \forall c \in C \tag{6}$$

$$x_c = \sum_{k \in K_c} x_c^k \lambda_c^k, \quad y_c = \sum_{k \in K_c} y_c^k \lambda_c^k, \quad \forall c \in C \tag{7}$$

$$s_{crq} = \sum_{k \in K_c} s_{crq}^k \lambda_c^k, \quad \forall c \in C \tag{8}$$

$$\sum_{c \in C} s_{crq} \leq 1, \quad \forall q \in Q, r \in R \tag{9}$$

For a given input layout, our objective is to minimize the weighted sum of HPWL of all nets subtracted by the total number of pin alignments for direct vertical M1 routing, while achieving a legal placement (no overlap of cells).

HPWL calculation. Constraint (2) calculates the HPWL for each net n, where HPWL as usual corresponds to the half-perimeter of the minimum bounding box that contains all pins of n. The maximum and minimum x, y coordinates of pins of the net n are obtained by Constraint (3). The absolute coordinates of pin p is determined by adding the coordinates (x_c, y_c) of p's owner cell c to (x_p, y_p).

Checking pin alignment. Constraint (4) checks whether pins p, q are aligned, by comparing their absolute coordinates. If the (absolute) x coordinate of p, q are not the same, $d_{pq} = 0$. Otherwise, the left side of the first and second constraints in Constraint (4) becomes zero, which makes $d_{pq} = 1$ allowed. In our implementation, we always ensure $d_{pq} = d_{qp}$.

Placement of each cell. Similar to [4], we assume that a perturbation range is given for each cell c, and that a cell cannot move beyond its given perturbation range. As in [4], we adopt the single-cell-placement (SCP) model of [10] to represent each candidate location and orientation for a cell. The binary variable λ_c^k represents a candidate k for a cell c, including the coordinates (x_c^k, y_c^k), the orientation (f_c^k), and whether placement site (r, q) is occupied (s_{crq}^k). These relations are handled by Constraints (6), (7) and (8). Constraint (5) ensures that exactly one candidate is chosen for cell c among all $\lambda_c^k, k \in K_c$. Constraint (9) ensures a legal placement.

3.2 MILP Formulation for OpenM1

To maximize direct vertical M1 routing for the *OpenM1* cell architecture, we must maximize "overlap" between target pins, which is different from the objective for *ClosedM1*. In addition to maximizing the number of overlapping pin pairs, we also maximize the sum of overlap lengths of each pin-to-pin (sub)net so as to increase the probability that the router completes the direct vertical M1 routing. The *OpenM1* objective is:

$$\textbf{Minimize:} \quad -\alpha \cdot \sum d_{pq} - \epsilon \cdot \sum o_{pq} + \sum_{n \in N} \beta_n \cdot w_n \tag{10}$$

To support *OpenM1*, we slightly modify the previous MILP formulation for *ClosedM1* by introducing extra variables. In this case, d_{pq} becomes a binary indicator of whether pins p and q are "overlapped", and Constraint (4) is replaced with Constraints (11)–(13). Our notation is again as described in Table 1.

$$a \geq x_c + x_{min,p}, \quad a \geq x_{c'} + x_{min,q}$$
$$b \leq x_c + x_{max,p}, \quad b \leq x_{c'} + x_{max,q}$$
$$\forall p, q, \text{ where } c, c' \text{ are the owner cells of pins } p, q \tag{11}$$

$$(y_c + y_p) - (y_{c'} + y_q) \leq G \cdot v_{pq} + \gamma \cdot H$$
$$(y_c + y_p) - (y_{c'} + y_q) \geq -G \cdot v_{pq} - \gamma \cdot H$$
$$\forall p, q, \text{ where } c, c' \text{ are the owner cells of pins } p, q \tag{12}$$

$$o_{pq} \leq b - a - \delta + G(1 - d_{pq}), \quad o_{pq} \leq G \cdot d_{pq}$$
$$o_{pq} \geq -G(1 - d_{pq})$$
$$\forall (p, q) \text{ pin pairs in net } n, \forall n \tag{13}$$

$$d_{pq} + v_{pq} \leq 1, \quad \forall p, q \tag{14}$$

Checking pin overlaps. Constraint (11) calculates the length of overlap in x direction between pins p and q. It first identifies the left side (a) and the right side (b) of the overlap between pins p and q. The overlap length o_{pq} is determined by a and b in Constraint (14). Constraint (12) checks whether the absolute difference of y coordinates of pins p and q is *larger* than γH and, if so, forces $v_{pq} = 1$. γ is a user-defined value for the maximum allowed vertical span of a direct vertical M1 routing.[3]

We use $\gamma = 3$, which means that a direct vertical M1 routing can cross three placement rows. For the case $v_{pq} = 1$, we do not need to make overlaps in the x direction since pins are multiple rows apart vertically; in such cases, it is difficult (i.e., highly improbable) to make a direct vertical M1 routing across multiple rows. Thus, Constraint (14) forces $d_{pq} = 0$ if $v_{pq} = 1$ so that the optimization does not make unnecessary overlaps. Constraint (13) forces $d_{pq} = 1$ if $b - a$ is larger than a predefined δ, which is the minimum required overlap length. Then, the o_{pq} is bounded by $b - a - \delta$. Otherwise, o_{pq} is bounded by zero.

[3]For example, $\gamma = 1$ means that direct vertical M1 routing can traverse between two adjacent cell rows, and $\gamma = 2$ (resp. 3) means that direct vertical M1 routing can go through one (resp. two) intervening cell row(s).

4. OVERALL FLOW

We now describe the overall flow of our optimization.

4.1 Distributable Optimization

In practice, the most critical limitation of the MILP-based approach is runtime. To overcome the runtime limitation, we adopt the distributable optimization proposed in [4].

We partition the layout into small windows (with width b_w, height b_h and optimize these windows in several iterations. In each iteration, we select a subset of windows that are independently optimizable, and optimize them in parallel. More specifically, we select windows that do not have any horizontal or vertical overlap (i.e., have disjoint projections onto the x-axis and onto the y-axis). For example, as shown in Figure 3, windows that are diagonally adjacent can be selected and optimized in parallel. This is because a given window's optimization is unaware of cell displacements concurrently being made outside of the window; if windows share projections onto the x- or y-axis, the impact of solutions on HPWL from each window cannot be accurately captured.

Figure 4 illustrates two example cases of (a) target windows with intersecting projections (on the y-axis) and (b) target windows with disjoint projections. Since the target windows are optimized in parallel, the optimizer calculates $\Delta HPWL1$ for the displacement of p in $w1$ without knowing pin q's displacement, and vice versa ($\Delta HPWL2$ for q in $w2$). However, according to the final locations of p and q, the pins that determine the bounding box corresponding to HPWL can change, as shown in the figure. In the (a) case, this results in a discrepancy between the total $\Delta HPWL$ and the sum of $\Delta HPWL$ from each window. In the (b) case, since p and q always determine the top-left point and the bottom-right point of the bounding box, the sum of $\Delta HPWL$ from each window is equal to the total $\Delta HPWL$.

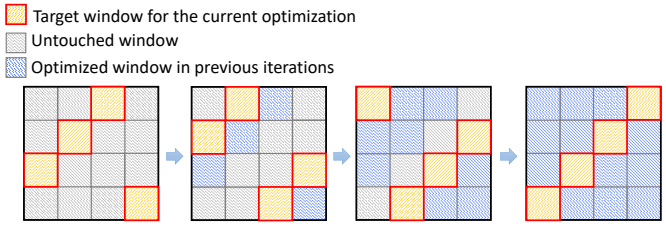

Figure 3: Illustration of distributable optimization.

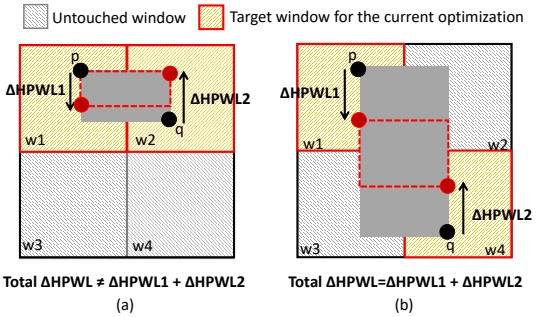

Figure 4: HPWL calculation for two cases. (a) Target windows with intersecting projections on the y-axis. (b) Windows with disjoint projections. In the case of (a), the total $\Delta HPWL$ is not equal to the sum of $\Delta HPWL$ values that are calculated from each window.

4.2 Overall Flow

Algorithm 1 ($VM1Opt()$) gives the metaheuristic outer loop of our detailed placement optimization considering direct vertical M1 routing. The inputs include a routed layout T, a weighting factor α and a sequence(queue) of input parameter sets U. Each parameter set in U includes window width (b_w), window height (b_h), maximum x displacement for cells (l_x), and maximum y displacement of cells (l_y). The sequence U is determined empirically based on experimental results (see Subsection 5.2). The output is an optimized layout T_{opt} with a heuristically minimized objective value Obj.

In Line 2, we obtain the first input parameter set u in the current U. In Lines 3–11, we iteratively run $DistOpt()$ with u until the normalized improvement (ΔObj) of the objective with respect to Obj of the previous iteration is less than a threshold θ. We use $\theta = 1\%$ as the threshold. In Line 4, we first store the previous Obj value as $preObj$. In Lines 6–7, we then perform $DistOpt()$ with window size and perturbation range defined in u (i.e., $u.b_w$, $u.b_h$, $u.l_x$, $u.l_y$) but without allowing the flip operation ($f = 0$). After that, $DistOpt()$ is performed again in Lines 8–9, with allowing of the flip operation ($f = 1$) but without allowing perturbation. Empirically, we observe that a sequential optimization that performs perturbation and flipping serially is faster than an optimization that performs perturbation and flipping simultaneously, while both optimizations give similar solution quality. In Line 10, we update the x and y shift values for windows (t_x, t_y). Although we avoid interference between windows by selecting diagonally-adjacent windows (recall Figure 3) for parallel optimization, cells at the boundary (i.e., cells that overlap two windows simultaneously) cannot be optimized. Thus, similar to the method of [4], we shift the windows to handle the unoptimized boundary region of the previous iteration. If ΔObj is less than θ (Line 3), we change u to the next input parameter set in U (Line 2). We iterate the optimization until we reach the last input parameter set in U.

Algorithm 2 describes details of $DistOpt()$. According to the given input parameters, we partition the layout into small windows (Line 1). We then select target windows that are independently optimizable and store them in D (Line 3) as explained in Section 4.1. Since we select target windows such that windows do not have any vertical or horizontal overlaps, the parallel optimization has $k = \sqrt{|W|}$ iterations, where $|W|$ is the total number of windows. In Lines 5–6, all windows $d \in D$ are optimized in parallel. For each window, we list candidates for each cell according to a given perturbation range (i.e., l_x and l_y, the maximum displacement of x and y, respectively). Along with input parameters α, β_n, γ and δ, we formulate the MILP instance for the window and use $CPLEX$ to solve the MILP instance. The solution is updated for each window, and is then used as a boundary condition for the target windows in the next iteration.

Algorithm 1 Overall flow of $VM1Opt$

Procedure $VM1Opt(T, \alpha, U)$
Input : Layout T, weighting factor α, queue of parameter sets U
Output : Layout T_{opt}

1: **while** $U \neq \emptyset$ **do**
2: $u \leftarrow U.pop()$; $\Delta Obj \leftarrow \infty$;
3: **while** $\Delta Obj \geq \theta$ **do**
4: $preOpj \leftarrow Obj$;
5: $l_x \leftarrow u.l_x$; $l_y \leftarrow u.l_y$; $f \leftarrow 0$;
6: $(T, Obj) \leftarrow DistOpt(T, t_x, t_y, u.b_w, u.b_h, l_x, l_y, f, \alpha)$;
7: $l_x \leftarrow 0$; $l_y \leftarrow 0$; $f \leftarrow 1$;
8: $(T, Obj) \leftarrow DistOpt(T, t_x, t_y, u.b_w, u.b_h, l_x, l_y, f, \alpha)$;
9: Update t_x, t_y
10: $\Delta Obj \leftarrow (preObj - Obj)/preObj$;
11: **end while**
12: **end while**
13: $T_{opt} \leftarrow T$;
14: return T_{opt};

Algorithm 2 Procedure $DistOpt$

Procedure $DistOpt(T, t_x, t_y, b_w, b_h, l_x, l_y, f, \alpha)$
Input : Horizontal (vertical) offset t_x (t_y), width (height) of window b_w (b_h), perturbation range in x (y) l_x (l_y), binary indicator of whether flip operation is allowed f, weighting factor α
Output : Updated layout T_{opt}, objective value Obj

1: A set of windows $W \leftarrow Partition(T, t_x, t_y, b_w, b_h)$;
2: **for** $i = 1$ to $\sqrt{|W|}$ **do**
3: $D \leftarrow$ set of current target windows;
4: // parallel optimization
5: $MILPFormulation(d, l_x, l_y, f, \alpha)$ for $\forall d \in D$;
6: Solve MILP and update MILP solutions to T;
7: // parallel optimization ends
8: **end for**
9: $T_{opt} \leftarrow T$
10: $Obj \leftarrow CalculateObj(T_{opt})$;
11: return T_{opt};

5. EXPERIMENTAL SETUP AND RESULTS

5.1 Experimental Setup

We implement our flow in C++ with *OpenAccess 2.2.43* [19] to support LEF/DEF [17], and with *CPLEX 12.6.3* [16] as our MILP solver. We apply our detailed placement optimization flow to ARM Cortex M0 core and three designs (aes, jpeg, vga) from OpenCores [20]. The design information is summarized in Table 2. The four designs are implemented with 7nm *OpenM1* and *ClosedM1* triple-V_t libraries from a leading technology consortium. We synthesize the testcases using *Synopsys Design Compiler K-2015.06-SP4* [21], and then perform placement and routing using *Cadence Innovus v16.1* [15]. The experiments are performed with 8 threads on a $2.6GHz$ Intel Xeon dual-CPU server. We note that with flexible computing resources, the number of usable threads could be as large as the number of layout windows that are independently optimizable ($\sqrt{|W|}$) to reduce runtime for larger designs.

5.2 Experimental Results

We have conducted two basic types of experiments. **ExptA** experiments seek to optimize our overall flow by finding input parameters and optimization sequences that give dominating runtime vs. solution quality tradeoffs. The aes design with *ClosedM1* is used for **ExptA** experiments. **ExptB** experiments apply our flow to both *ClosedM1*-based and *OpenM1*-based designs. For all experiments, we use $\beta = 1$ so that our MILP formulation minimizes pure HPWL.

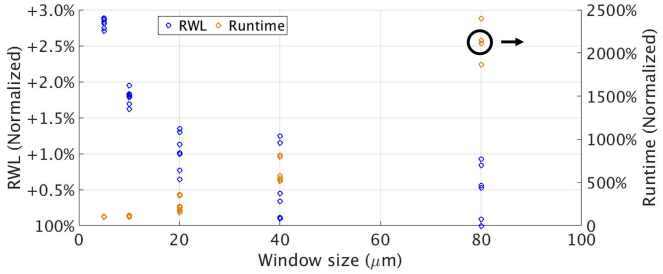

Figure 5: Scalability test with various window sizes and perturbation ranges.

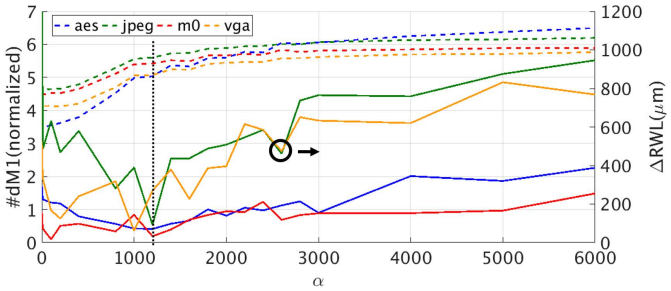

Figure 6: Sensitivity of total routed wirelength (RWL) and the number of direct vertical M1 routings (#dM1) to α.

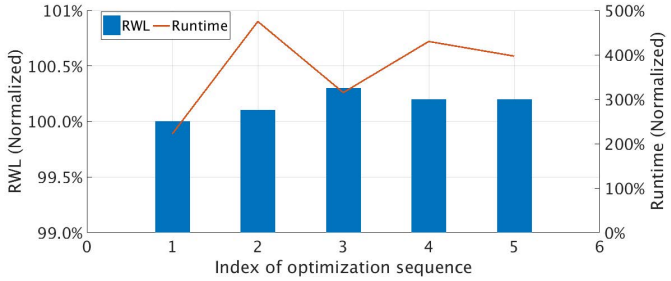

Figure 7: Results of various optimization sequences.

ExptA-1: Scalability study on window size and perturbation range. We sweep the window size and the perturbation range to study the tradeoff between solution quality and runtime. We assume square windows and vary $b_w = b_h$ from $5\mu m$ to $80\mu m$. For the perturbation range, we try $l_x \in \{2, 3, 4, 5\}$, $l_y \in \{0, 1\}$. In this experiment, we only run one iteration in Algorithm 1 (i.e., one pair of *DistOpt*()). Figure 5 shows the normalized routed wirelength (RWL) and runtime versus the window size. As the window size increases, the routed wirelength decreases, as expected. However, we observe huge runtime increases, e.g., 5× runtime increase with $b_w = b_h = 40\mu m$. To compromise the runtime overhead and the solution quality, we select the option with shortest runtime that gives $\leq$ 1% total routed wirelength increase compared to the minimum routed wirelength; this is $b_w = b_h = 20\mu m$, $l_x = 4$, and $l_y = 1$.

ExptA-2: Sensitivity study for α. We sweep α values and study the impact of α on the number of direct vertical M1 routings (#dM1) and the routed wirelength (RWL). We vary α from 0 to 6000 – e.g., for *ClosedM1*-based design, the objective with $\alpha = 10$ prefers one more aligned pin pair at the cost of at most 10 units increase in HPWL. Figure 6 shows total routed wirelength (RWL) and the number of direct vertical M1 routings (#dM1) versus α. As α increases, the number of direct vertical M1 routings increases. However, maximizing the number of direct vertical M1 routings does not always reduce routed wirelength, Based on our studies, we select $\alpha = 1200$ for *ClosedM1*. Similarly, we experiment on *OpenM1*-based designs and select $\alpha = 1000$.[4]

ExptA-3: Sequence of optimization. We explore various sequences of input parameter sets ($b_w = b_h$, l_x, l_y) to optimize our overall flow. We illustrate this with five example optimization sequences: (1) (20, 4, 1); (2) (10, 3, 1) → (10, 4, 0) → (20, 4, 0) ; (3) (10, 3, 1) → (20, 3, 1) → (20, 3, 0); (4) (10, 3, 1) → (20, 3, 0) ; and (5) (10, 3, 1) → (10, 3, 0) → (20, 3, 1) → (20, 3, 0). Figure 7 shows RWL and runtime for these optimization sequences. We observe that optimization sequences 1 and 2 with $l_x = 4$ give better solution quality (in terms of RWL). However, optimization sequence 2 consumes twice the runtime of optimization sequence 1. Therefore, $(20, 4, 1)$ would be a preferred choice of sequence.

ExptB-1: Detailed placement optimization for *ClosedM1*-based designs. Table 2 shows overall results for our detailed placement optimization. Our optimizer increases the number of direct vertical M1 routings by more than 4× compared to the initial post-routing solution, while decreasing overall M1 wirelength. This means that we remove long vertical M1 routings that are not used for direct vertical routing, while generating many short, direct vertical M1 routes; this results in smaller M1 wirelength and a larger number of M1 routing segments. Along with the increase in the number of direct vertical M1 routings, we achieve up to 6.4% routed wirelength (RWL) reduction and up to 14.4% #via12 reduction without design rule violations (DRVs).[5] Total power also decreases by up to 0.9%. For half of the designs, HPWL increases in favor of more dM1 to further reduce routed wirelength.

To study the impact of direct M1 routing on congestion reduction, we increase the initial utilization on the aes design so as to induce congestion hotspots, which lead to design rule violations. In Figure 8, we show that our optimizer has the added benefit of avoiding a substantial fraction of DRVs (#DRVs orig vs. opt in the figure). We note that even though our optimization consistently decreases DRVs, routing QOR is ultimately determined by the initial placement quality. Notably, placement QOR with utilization 83% from the commercial tool is worse than placement with utilization 84% in terms of DRVs. The cause of this phenomenon is beyond our present scope.

ExptB-2: Detailed placement optimization for *OpenM1*-based designs. Our optimizer increases the number of direct vertical M1 routings by around 60% compared to the initial post-routing solution. We observe that the increase of the number of direct vertical M1 routings for *OpenM1*-based designs is much smaller than that for *ClosedM1*-based designs. This small increase of the number of direct vertical M1 routings results in only up to 2.2% routed wirelength reduction, and up to 4.1% #via12 reduction, without design rule violations.

[4]We do not show data for *OpenM1* due to the page limit.

[5]Here we refer to routing DRVs. In this paper, we do not consider advanced node placement rules (e.g., drain-drain abutment, minimum implant area, etc.). However, our framework is fully compatible, and can be easily integrated, with the work of [4] and complex sub-14nm rules.

Table 2: Experimental results of **ExptB**.

Design	#Inst	Util (%)	α	#dM1		M1 WL (μm)		#via12		HPWL (μm)		RWL (μm)		WNS (ns)		Power (mW)		Runtime (sec)
				Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final	Init	Final (Δ%)	
***ClosedM1*-based designs**																		
M0	9922	75%	1200	545	2955 (442.2)	676	629 (-7.0)	35766	31932 (-10.7)	22850	23760 (4.0)	27636	26833 (-2.9)	0.000	0.000	2.444	2.431 (-0.5)	344
aes	12345	75%	1200	631	3177 (403.5)	970	710 (-26.8)	43248	38631 (-14.4)	30420	28890 (-5.0)	32560	30471 (-6.4)	0.000	0.000	3.240	3.212 (-0.9)	711
jpeg	54570	75%	1200	3694	20688 (460.0)	3605	3329 (-7.7)	179315	153500 (- 5.7)	91030	88900 (-2.3)	96621	90593 (-6.2)	0.000	0.000	28.592	28.399 (-0.7)	1216
vga	68606	75%	1200	2460	12473 (407.0)	5973	5428 (-9.1)	270930	255466 (-10.7)	169200	169800 (0.4)	206558	204269 (-1.1)	0.000	-0.002	53.614	53.542 (-0.1)	561
***OpenM1*-based designs**																		
M0	9891	75%	1000	1183	1931 (63.2)	3681	3790 (3.0)	35099	34336 (-1.7)	24790	24570 (-0.9)	29884	29575 (-1.0)	-0.003	0.000	2.475	2.468 (-0.3)	298
aes	12348	75%	1000	1341	1975 (47.3)	4646	4620 (-0.5)	43004	42269 (-4.1)	30670	29980 (-2.2)	34338	33592 (-2.2)	0.000	0.000	3.273	3.263 (-0.3)	325
jpeg	54689	75%	1000	8391	13763 (64.0)	18709	19244 (2.8)	173622	166411 (-3.8)	92100	91110 (-1.1)	103257	101463 (-1.7)	0.000	-0.001	29.024	28.957 (-0.2)	1026
vga	68729	75%	1000	7714	13132 (70.2)	26912	26823 (-0.3)	261424	251558 (-2.2)	170000	168700 (-0.8)	215218	213598 (-0.8)	0.000	-0.002	53.805	53.730 (-0.1)	515

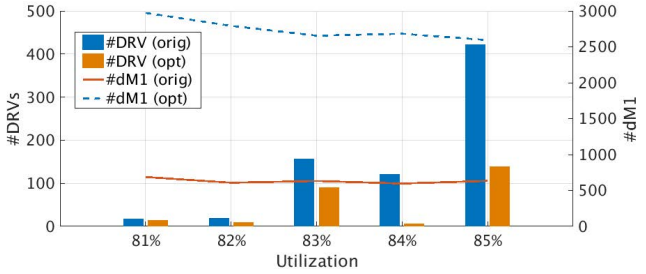

Figure 8: The number of DRVs after optimization for aes design with various utilizations. Also shown: the number of direct vertical M1 routings.

There can be several reasons for the lesser improvement seen for *OpenM1*-based designs. Our current hypothesis is that P&R for *OpenM1* is very similar to traditional P&R in terms of pin access. In traditional P&R flows with conventional libraries, where most pins are on M1, the M2 layer is used to access the pins. Similarly, *OpenM1* cells also have pins (on or) below M1, and M1 can be used for pin access. Thus, P&R for *OpenM1* can be seen as a variant of the conventional P&R flow, where the bottom routing layer is shifted down to M1. Indeed, in *OpenM1*-based designs, direct vertical M1 routing can block access to other pins, which limits the wirelength reduction. On the other hand, in *ClosedM1*-based designs, direct vertical M1 routing does not block any pin access, and is thus "free" in terms of routing resource. Compared to *ClosedM1*, where routed wirelength can be reduced even at the cost of HPWL increase, *OpenM1*-based designs prefer smaller α to reduce HPWL. However, given our use of a black-box commercial router, it is difficult to identify root causes of the improvement difference between *OpenM1* and *ClosedM1*. This is the subject of one of our ongoing studies.

6. CONCLUSIONS

In this work, we propose a vertical M1 routing-aware detailed placement optimization based on mixed-integer linear programing (MILP) for two new cell architectures in sub-10nm nodes, i.e., *ClosedM1* and *OpenM1*. With our optimization, up to 6.4% (resp. 2.2%) total routed wirelength reductions and 14.4% (resp. 4.1%) #via12 reductions are achieved for *ClosedM1*-based (resp. *OpenM1*-based) designs, with no adverse timing impact.

We note that the library characterization model for *ClosedM1* library cells might need to change since the vertical M1 routings might affect cells' library model (change in gate capacitance, etc.). However, according to our study with an INV cell in *ASAP ASU 7nm PDK* [14], the timing impact is negligible ($\leq$ 0.1ps).[6]

Our future works include (i) a comprehensive study of timing library characterization for *ClosedM1*, to accurately capture the timing impact of direct vertical M1 routing; (ii) extension of our placement objective function to consider other design criteria, including timing criticality, pin density, routing congestion, and routing design rules; (iii) (meta)heuristic innovation to improve QOR and scalability; and (iv) theoretical understanding of *OpenM1*-based layout design to inform an improved optimization strategy.

[6]We modify pin shapes (increase the pin length by 32nm) in a cell layout, run parasitic extraction with *Calibre xRC v2016.1_31.21* [18], and measure cell delay and slew with *HSPICE I-2013.12* [22]. We observe that the delay and slew impacts of the pin modifications are negligible ($\leq$ 0.1ps). Further, there are only a small number of possible uses of vertical M1 incident to a cell (this number is a function of the number of pins, and of upward versus downward alignments). In a regime where these delay and slew changes must be modeled, each of these contexts could be characterized.

7. REFERENCES

[1] M. Gupta, K. Jeong and A. B. Kahng, "Timing Yield-Aware Color Reassignment and Detailed Placement Perturbation for Bimodal CD Distribution in Double Patterning Lithography", *IEEE TCAD* 29(8) (2010), pp. 1129-1242.

[2] P. Gupta, A. B. Kahng and C.-H. Park, "Detailed Placement for Improved Depth of Focus and CD Control", *Proc. ASPDAC*, 2005, pp. 343-348.

[3] P. Gupta, A. B. Kahng and C.-H. Park, "Manufacturing-Aware Design Methodology for Assist Feature Correctness", *Proc. SPIE 5756 (DPIMM III)*, 2005, pp. 131-140.

[4] K. Han, A. B. Kahng and H. Lee, "Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints", *Proc. ICCAD*, 2015, pp. 867-873.

[5] A. B. Kahng , P. Tucker and A. Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites", *Proc. ASPDAC*, 1999, pp. 241-244.

[6] A. B. Kahng, I. L. Markov and S. Reda, "On Legalization of Row-Based Placements", *Proc. GLSVLSI*, 2004, pp. 214-219.

[7] Y. Du and M. D. F. Wong, "Optimization of Standard Cell Based Detailed Placement for 16nm FinFET Process", *Proc. DATE*, 2014, pp. 1-6.

[8] S.-W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement", *Proc. ICCAD*, 2000, pp. 165-170.

[9] S. Li and C.-K. Koh, "Mixed Integer Programming Models for Detailed Placement", *Proc. ISPD*, 2012, pp. 87-94.

[10] S. Li and C.-K. Koh, "MIP-based Detailed Placer for Mixed-size Circuits", *Proc. ISPD*, 2014, pp. 11-18.

[11] T. Lin and C. Chu, "TPL-Aware Displacement-driven Detailed Placement Refinement with Coloring Constraints", *Proc. ISPD*, 2015, pp. 75-80.

[12] Y. Lin, B. Yu, B. Xu and D. Z. Pan. "Triple Patterning Aware Detailed Placement Toward Zero Cross-Row Middle-of-Line Conflict", *Proc. ICCAD*, 2015, pp. 396-403.

[13] B. Yu, X. Xu, J.-R. Gao and D. Z. Pan, "Methodology for Standard Cell Compliance and Detailed Placement for Triple Patterning Lithography", *Proc. ICCAD*, 2013, pp. 349-356.

[14] ASAP ASU 7nm PDK, `http://asap.asu.edu/asap/`

[15] Cadence Innovus User Guide, `http://www.cadence.com`

[16] IBM ILOG CPLEX. `http://www.ilog.com/products/cplex/`

[17] LEF/DEF reference 5.7. `http://www.si2.org/openeda.si2.org/projects/lefdef`

[18] Mentor Graphics Calibre, `https://www.mentor.com`

[19] Si2 OpenAccess. `http://www.si2.org/?page=69`

[20] OpenCores: Open Source IP-Cores, `http://www.opencores.org`

[21] Synopsys Design Compiler User Guide, `http://www.synopsys.com`

[22] Synopsys HSPICE User Guide, `http://www.synopsys.com`