CATALYST: Planning Layer Directives for Effective Design Closure

Yaoguang Wei[1], Zhuo Li[2], Cliff Sze[2]
Shiyan Hu[3], Charles J. Alpert[2], Sachin S. Sapatnekar[1]
[1]Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, 55455
[2]IBM Austin Research Lab, Austin, TX, 75758
[3]Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI, 49931
Email: weiyg@umn.edu; {lizhuo,csze}@us.ibm.com; shiyan@mtu.edu; alpert@us.ibm.com; sachin@umn.edu

Abstract—**For the last several technology generations, VLSI designs in new technology nodes have had to confront the challenges associated with reduced scaling in wire delays. The solution from industrial back-end-of-line process has been to add more and more thick metal layers to the wiring stacks. However, existing physical synthesis tools are usually not effective in handling these new thick layers for design closure. To fully leverage these degrees of freedom, it is essential for the design flow to provide better communication among the timer, the router, and different optimization engines. This work proposes a new algorithm, CATALYST, to perform congestion- and timing-aware layer directive assignment. Our flow balances routing resources among metal stacks so that designs benefit from the availability of thick metal layers by achieving improved timing and buffer usage reduction while maintaining routability. Experiments demonstrate the effectiveness of the proposed algorithm.**

I. Introduction

Physical synthesis is a critical component of modern design methodologies, enabling timing closure at the physical design stage. Technology scaling brings new challenges and opportunities to physical synthesis. Wire resistance per unit length increases quadratically with technology scaling and results in significant increases in wire delay. However, recent work [1], [2] shows that the availability of thicker wires in higher metal layers could potentially relieve this problem. At 65 nm technology, there are four 1× layers, three 2× layers and two 4× layers (Fig. 3). On a 2× layer, the single-width wires are 2× thicker and 2× wider than those on 1× layer, and therefore the per-unit wire resistance is reduced by roughly 4× (the per-unit capacitance is roughly similar across all layers, which is ensured by design rules including wire spacing and process specifications such as inter-layer dielectric thickness), greatly compensating for technology scaling effects. On the 2× [4×] layer, signals can roughly go 1.7× [2.5×] faster, with 2× [4.4×] reduction in buffer resources. Therefore, the difference in wire delays in different layers provides another dimension to timing optimization, beyond gate/wire sizing and buffering. Assigning timing-critical nets to thick layers can reduce area/power and improve timing closure by reducing delays and the buffer count. As illustrated in Fig. 1, the slack for a two-pin net A on a 4× layers is improved from -10 ps to 10 ps as compared to the corresponding route on the 1× layer, and the number of buffers is reduced from 7 to 1. Moreover, by using thick layers wisely, it could be shown that a 31% reduction on buffer area averaged over several industrial circuits can be achieved (Sec. VI-B). On the other hand, there are limited resources on thicker layers, and if too many nets are assigned to thicker layers, the design may not be routable or have large post-routing timing degradation.

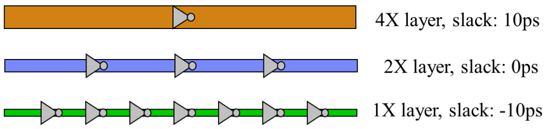
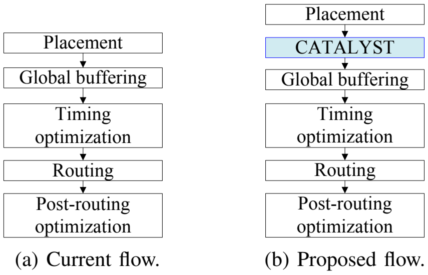


Fig. 1. Assigning the same net to thicker layers improves timing and buffering.

This extra dimension affects the traditionally predicted trend for interconnect synthesis and buffering, and presents a new problem to the physical synthesis: how to use these thick metal layers wisely. This problem is pervasive in all steps of the design flow, from the early stages to the late stages and relates to the classical tradeoff between flexibility and accuracy: as one progresses deeper into the design flow, timing estimation becomes more accurate but the level of flexibility in changing layer assignments diminishes.

Existing works on layer assignment have focused only on late stages of design, mainly during the routing and buffering stages. Most of the previous related works are from the global and detailed routing literature, but do not address the problem of early planning for layer assignment. Conventional routers perform layer assignment purely for routing congestion minimization and many works focus on how to perform layer assignment with via minimization [3]–[8]. In these works, the timing benefit of thick layers is not leveraged at all. Subsequent work on timing-driven layer assignment [9]–[12] has used timing information to drive layer assignment. While it is certainly necessary to consider layer assignment during routing, the timing gain in these works is limited since routing is performed after all optimizations are completed, or at least after a majority of buffers are placed. Recent papers [13]–[15] focus on how to obey the given layer assignment constraints from the prior synthesis stage in the routing algorithm, but do not discuss the process of generating these constraints.

The work in [1] performs layer assignment during the optimization stage and is combined with buffering. It presents two algorithms to perform simultaneous layer assignment and buffer insertion on a single net given the Steiner topology, and shows significant timing benefits and buffer area savings. However, it has two major limitations. First, it is not aware of the routing congestion. The approach attempts to control the number of nets promoted to thick layers, but its guess and trial approach could still easily cause over-promotion (assigning too many nets to thick layers) in the design, which causes the design to be unroutable [2]. Second, it does not explicitly minimize the buffer usage: it may underuse the thicker layers if the timing can be closed by buffer insertion on thinner layers, with excessive buffers inserted.



(a) Current flow. (b) Proposed flow.

Fig. 2. Current and proposed physical synthesis flows.

We propose a novel algorithm, **CATALYST**: **C**ongestion **A**nd **T**iming **A**ware **L**a**Y**er a**S**signmen**T**, to perform layer assignment to maximize the timing benefits with congestion control at early stages. CATALYST alters a traditional physical synthesis flow [16], [17], depicted in Fig. 2(a), and is inserted just before global buffering, as shown in Fig. 2(b). We believe it is a *catalyst* to enable faster and better design closure if thicker metal layers are wisely used earlier. Unlike [1], our algorithm tries to assign

978-3-9815370-0-0/DATE13/©2013 EDAA

a large number of nets to thick layers with the goal to minimize the buffer usage. Moreover, since our method has a global routing engine embedded inside, it has good control of the congestion when performing layer assignment.

Our work has several significant contributions. It presents

- a novel problem formulation for layer assignment at early stages;
- techniques to control congestion during layer assignment;
- techniques to maximize the timing benefits of layer assignment guided by a delay model (to be discussed in Sec. II-C);
- techniques to minimize buffer usage by assigning as many nets as possible to thick metal layers while controlling congestion.

Our algorithm has been tested on several industrial designs across 65 nm, 45 nm, and 32 nm technologies. Compared with another aggressive layer assignment algorithm, CATALYST can achieve similar timing improvements (improving the worst slack by 0.8 ns on average) but avoid high congestion. Moreover, CATALYST has been embedded and tested in an industrial physical synthesis flow (Fig. 2(b)). Experimental results demonstrate that CATALYST can save the buffer area by 10% on average and up to 18%, while maintaining the similar congestion, timing, and runtime. These savings also improve the design power and cost and help achieve effective design closure.

II. Preliminaries

In this section, we discuss the routing and timing models used in CATALYST, the notations used and the problem formulation.

A. Layer Directives and Notations

An important concept involved in this work is that of the **layer directive** (simply, **directive**): a directive is a constraint on a net which specifies the valid layers the net can be routed on, and is typically given by a pair of layer names. For example, a net n with directive $[M_5, M_9]$ means that net n can only be routed between layer M_5 and M_9. Here, M_i denotes the i^{th} metal layer.

CATALYST differs from the traditional layer assignment step in global routing [3]–[8], which assigns the wires to different layers as a last step to complete the routing of a net. In contrast, CATALYST generates layer directive *constraints* for timing-critical nets, and these directives are propagated throughout the physical synthesis flow. We will use the term **layer directive assignment (LDA)**, or simply **directive assignment**, to refer to our layer assignment process.

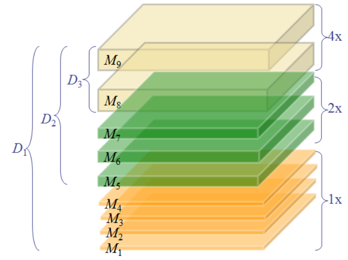


Fig. 3. A pictorial view of the 65 nm technology.

We now elaborate on the way layer directives are provided for a 65 nm technology. As illustrated in Fig. 3, this technology consists of three planes, where a **plane** refers to the layers with same thickness. Correspondingly, three layer directives can be derived: $[M_1, M_9]$ as D_1, $[M_5, M_9]$ as D_2 and $[M_8, M_9]$ as D_3. From the example, we can see that for j^{th} directive D_j, the bottom layer will be the bottom layer in j^{th} plane, and the top layer will be always the top metal layer. Note that D_1 imposes no constraint, and is not used in practice.

It is useful to assign the most timing-critical nets to D_3, so that these nets can obtain best delay gains from the thickest layers. If assigning a timing-critical net n to D_3 results in congestion bottlenecks, we could give it greater flexibility by assigning it to D_2, allowing n to be routed between M_5 and M_9. In such a case, we use the parasitics in 2× plane when computing the wire delay and gate delay associated with net n, since industrial routers tend to route nets in the lowest allowable metal layers, which is consistent with the objective of via minimization. If there are no resources in the lowest layers, a wire may be routed in a higher layer, and the timing computation based on the parasitics of the 2× plane[1] is guaranteed to be pessimistic.

We further introduce some notations. Let M be the number of planes in a design. Correspondingly, there will be M layer directives D_j, $1 \leq j \leq M$. For simplicity, we overload the notation to also use directive D_j to denote the set of layers specified by D_j. We refer to D_1 as the *bottom* directive, and D_M the *top* directive. The *promotion* of a net will refer to the case where we assign a net from D_j to a directive with a higher index (corresponding to higher metal layers); similarly, the *demotion* of a net will refer to the opposite. Since the parasitics of the lowest metal layer in a directive will be used to evaluate the delay of the nets in that directive, promoting (demoting) a net will improve (worsen) its estimated delay.

B. Global Routing with Layer Directives

In global routing, the chip is usually tessellated into grids (also called *g-cells*), and the global routing graph (GRG), $G_r = (V_r, E_r)$, is constructed. Each node in V_r represents a g-cell in the layout, and an edge (called a *g-edge*) in E_r denotes the boundary between two adjacent g-cells. In the presence of multiple layers, the 2D GRG becomes a 3D GRG, with a 2D GRG representing each layer, and the z-direction g-edges representing vias that connect the 2D GRG's.

Most of traditional global routers adopt a three-stage routing scheme: plane projection, 2D routing and layer assignment. However, routing with layer directives is quite different. Several methods have been proposed to address layer directives in global routing [13]–[15]. We use the progressive projection method in [15] due to the flexibility it affords. In this method, the nets are first partitioned to several sets according to their directives. Let $\mathcal{P}_k$ be the set of nets with directive D_k, where $1 \leq k \leq M$. A set of routing subproblems I_k are constructed with a set of nets $\mathcal{P}_k$ and a set of layers in D_k. Next, the subproblems are solved one by one. Since the nets in higher directive have stronger limitations (with fewer available layers), subproblem I_k with larger k will be solved first. In solving I_k, the 2D GRG is constructed in the following way: the capacity is aggregated by only using the layers in D_k, and the routing solution from the previous subproblem I_{k+1} (if available) will be treated as existing wires in the 2D GRG. In summary, the whole routing process will have M passes: in each pass, 2D-routing and then layer assignment will be performed on the 2D GRG constructed using the method discussed, and finally an accumulated 3D solution will be output.

The traditional goals of global routing are to minimize the wirelength, congestion and via count. Congestion can be evaluated by different metrics such as overflow. The overflow of a g-edge is defined as the excess routing demand over the capacity on the g-edge. Recently, a new metric, the average congestion of g-edges (ACE), has been proposed in [18] and shown to be more effective than conventional overflow-based metrics. Here, congestion of a g-edge is defined as the ratio of the routing demand to its capacity. Roughly, the ACE metric computes the average congestion of the top $x\%$ congested g-edges, denoted as $ACE(x)$. A derived metric, peak weighted congestion (PWC), was adopted in DAC 2012 contest [19], given by $(ACE(0.5) + ACE(1) + ACE(2) + ACE(5))/4$.

C. Timing Metrics and Model

In this paper, we use the following timing metrics: the worst slack (WSLK), the figure of merit (FOM) which is sum of the gap in slack to a target (s_{tar}) for all timing endpoints with slack below s_{tar}, and the number of timing endpoints with negative slack, n_{neg}. Here, we set s_{tar} to 0 ps, so that FOM is effectively the total negative slack.

As pointed out in [16], for appropriate tradeoffs between runtime and optimization accuracy, timing models with different levels of accuracy are used at different stages in a typical physical synthesis flow (Fig. 2(a)).

[1] The parasitics of lowest layer and highest layer specified by the layer directive may be used as a $[\min, \max]$ range in timing analysis to get more information if necessary.

Before placement, an optimistic zero-wire-load model is typically used during logic synthesis. Subsequent to placement, one could switch to a traditional RICE model [20]. However, such a timing analysis will result in huge numbers of critical paths because buffering has not yet been performed, and it becomes impossible to distinguish real critical paths from paths that simply lack buffers. Long wires without buffers will have quadratic delays and make the timing appear much worse than it potentially will be after timing optimization. Hence, to avoid this problem, a **linear delay model** [21]–[23] can be used for wires, where the interconnect delay is estimated to be linear with the wirelength assuming optimal buffering. It allows a reasonable estimate of post-placement timing, and allows the buffering to be deferred until layer assignment has been performed.

In this work, CATALYST is performed before buffering (Fig 2(b)), and therefore, we use a linear delay model similar to that used in [23][2].

D. Layer Directive Assignment

The LDA problem can be stated as follows: given a design, assign directives to nets with the goal of satisfying the timing and congestion constraints, and minimizing the number of buffers required.

To evaluate the effects of assigned layer directives on routing congestion, a global router with ability to obey the layer directives, such as [13]–[15], can be used to evaluate the congestion. We should try to keep the congestion with newly generated directives similar to that without any directives. Precisely, the congestion constraint to the LDA problem can be $g_{LDA} \leq \beta g_{org}$, where g_{LDA} and g_{org} are calculated by a congestion metric of the routing solution with and without assigned directives, respectively, and β is a user-defined parameter.

III. Overview of CATALYST

In this section, we discuss how we solve LDA problem and overview the CATALYST algorithm. We solve the LDA problem by decomposing it into two subproblems:

Subproblem 1: Timing-driven directive assignment.

Subproblem 2: Congestion- and timing-aware directive assignment.

The goal of the Subproblem 1 is to generate initial layer directive assignment solution to meet the timing constraints, while minimizing the number of nets with directive assigned. Subproblem 2 has two goals. First, it tries to keep the initial directive assignment from the solution of Subproblem 1 within the allowable congestion range. Second, it tries to assign as many nets as possible to higher directives to further improve timing and reduce potential buffer usage.

To solve Subproblem 1, we propose a simple but effective timing-driven directive assignment heuristic by promoting the timing-critical nets to higher directives one by one with incremental timing updates. To solve Subproblem 2, our tool embeds a global routing engine inside to control the congestion. First it will examine the directives obtained from the first step, and relax the constraints/directives if they cause congestion. Next, it will try to perform directive assignment on the rest of nets to further improve timing and reduce potential buffer usage. It then calls the timer to update the timing at the end to capture the effects of the nets touched during this step. Simply speaking, the first step focuses on timing improvement, and the second step focuses on congestion control and buffer usage improvement.

The CATALYST algorithm is an iterative process. After every solution of Subproblem 2, we rerun timing analysis based on the new directive assignments, which change the timing, and then start a new iteration. The iterations continue until the stopping criterion is satisfied: either WSLK or FOM becomes worse, or the improvement on both of them is smaller than a threshold θ, or the user-defined maximal number of iterations n_{itr} is reached.

IV. Timing-driven directive assignment

Subproblem 1 can be formulated as follows. Given a circuit, to compute a directive assignment solution to maximize the timing improvement and to minimize the total cost. Depending on the purpose, one can model this cost as congestion, to avoid unroutable regions, or a directive cost by assigning a higher cost to a higher directive to minimize the usage of higher layers. Our work uses a directive cost.

In this subproblem, the circuit can be modeled by a directed acyclic graph (DAG) [24], $G_m = (V_m, E_m)$. The nodes in V_m are the logic gates in the circuit and the primary inputs and outputs of the circuit. Each node in the DAG has a delay, corresponding to the gate delay. Each edge in E_m denotes the interconnect from one node to the other. Each edge has a wire delay under a given layer directive. Here, each edge will have M choices, each with different delay values, and also different associated costs. As shown in [12], even a simplified form of this problem, where only a single routing tree topology of a net is considered, is NP-complete.

In this paper, we propose a simple yet efficient heuristic. The main idea is to use as few higher directives as possible by only promoting the currently most timing-critical nets. We will explain our algorithm assuming we have just started the k^{th} iteration of CATALYST, and there are some nets with directives from last iteration. A demotion stage is first performed to check if a net can be demoted from its original directive to lower directive if no timing violations are created. This helps create a small set of layer directives for the congestion-aware directive assignment step. Note that in the first iteration of CATALYST, all the nets are in D_1 and no demotion is required. Next, the timing-driven promotion stage starts. This stage assigns timing-critical nets to higher directives gradually, by looping through each directive from D_2 to D_M. In each step with target directive D_j, we will first put all the timing-critical nets to a list A, and then sort them by their slacks[3] in an increasing order. Then we loop through each net n_i in list A, and promote n_i to D_j if its current directive $D(n_i)$ is lower than D_j (note that n_i may have already been assigned to a higher directive from a previous CATALYST iteration). If n_i is promoted, then the timing graph is updated with incremental static timing analysis. Since the timing of other nets may change due to the promotion of n_i, the subsequent nets in A may be skipped in later processing if they become non-critical. This heuristic promotes only "necessary nets" to higher directives to avoid potential congestion problems.

V. Congestion- and timing-aware directive assignment

There are two major goals of Subproblem 2: first, to maintain the initial directive assignment to the extent permitted by congestion, while demoting some of the directives that may cause congestion problems, and second, to assign as many nets as possible to higher directives/layers in order to reduce buffer usage without degrading congestion. This is different from the method for Subproblem 1 which tries to promote as few nets as possible to higher layers. Besides promoting timing-critical nets, this step also promotes non-timing-critical nets for potential buffer savings. The positive slack of a net under linear delay model is obtained assuming optimal buffering with the current directive, and if it is promoted to higher directives, fewer buffers will be required to keep the positive slack.

Subproblem 2 can be formulated as follows. Given a circuit, a congestion constraint and a group of nets with initial directives, try to maintain as many of the initial directives as possible, and further promote as many nets as possible to higher directives to maximize the sum of scores of the promoted nets. Here, the score of a net is used to quantify the benefits in timing improvement and buffer savings by promoting it to a higher layer. In this work, as a first attempt, we simply use the following score function for net n_i:

$$w(n_i) = \exp(-s(n_i)/T_c), \tag{1}$$

where $s(n_i)$ is the worst slack of all the sinks of n_i, and T_c is the clock period used for normalization.

A. Algorithm

In this section, we will introduce the algorithm for Subproblem 2. For later reference, we refer to our algorithm as CADA (**C**ongestion and

[2] Our enhanced implementation also considers the effects of vias on the wire delay.

[3] The slack of a net is the worst slack of all the sinks of the net.

timing **A**ware **D**irective **A**ssignment). The key of CADA is to control congestion, which is achieved by performing 2D directive-aware routing and directive assignment. In this work, a 2D-routing engine adapted from MaizeRouter [6] is used but the maze routing in MaizeRouter is replaced by extreme edge shifting (refer to [6] for details about this technique) to improve the speed; however, it should be pointed out our flow can work with any other 2D routing engine. To deal with directives in 2D routing, we use the progressive projection method in [15] as discussed in Sec. II-B. In the LDA problem formulation, the constraint on congestion requires $g_{LDA} \leq \beta g_{org}$. To consider this directly inside CADA, it is necessary to invoke global routers before and after LDA. However, this may be inefficient. Instead, for simplicity, we use a conservative method to control congestion: a net can be promoted to directive D_j only if routing a net in this directive causes zero overflow. Otherwise, the net will stay in lowest directive D_1. Note that in the solution of Subproblem 1, this strategy is not used since the congestion of a net at that step cannot be efficiently obtained.

The basic flow of CADA is shown in Fig. 4. CADA has three stages: initial directive adjustment, greedy directive assignment and directive assignment refinement. First, the initial directive assignment from the timing-driven directive assignment step (Sec. IV) is re-evaluated in terms of congestion. At the allowable congestion level, we will try to maintain as many initial directives as possible, and demote the nets to lower directives if the initial assignment causes congestion. The purpose of the second stage is to assign/promote as many nets as possible to the higher directives to further improve timing and reduce potential buffer usage. The goal of the third stage is to verify the effects of directive assignment on congestion and to fine-tune the layer directives by performing a trial routing process with the directive assignment.

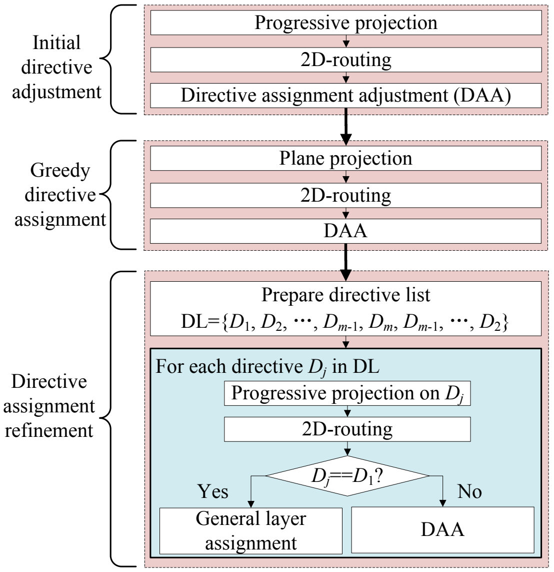

Initial directive adjustment
Progressive projection
2D-routing
Directive assignment adjustment (DAA)
Greedy directive assignment
Plane projection
2D-routing
DAA
Directive assignment refinement
Prepare directive list
DL={D_1, D_2, ⋯, $D_{m\text{-}1}$, D_m, $D_{m\text{-}1}$, ⋯, D_2}
For each directive D_j in DL
Progressive projection on D_j
2D-routing
D_j==D_1?
Yes
No
General layer assignment
DAA

Fig. 4. Basic flow of congestion- and timing-aware directive assignment.

As stated earlier, at the first stage, only the nets with initial directives from the timing-driven directive assignment step are processed. With the progressive projection method, the nets with top directive D_M will be routed first. After 2D-routing, we will first sort these nets by their scores as defined in Eq. (1), and will process the nets one by one. We first attempt to assign a net to its associated directive, and then we check the congestion. If no congestion violation is found, this net will be marked to be assigned to that directive; otherwise, an attempt to assign it to lower directives will be made until a directive without causing congestion is found, or the lowest directive D_1 is reached. This procedure to find the best directive with an initial target directive is called **Directive Assignment Adjustment (DAA)**, and will be discussed in more details in Sec. V-B2. After the nets with D_M are all routed, we repeat a similar process for nets with directive D_{M-1}. This process iterates until all the nets with directives are processed. Since the nets with initial directives are the most timing-critical nets, and their number is typically relatively small as compared to the total number of nets, the wiring resources consumed by these nets are locked down after this step, which means that their routing paths will be kept in the 3D GRG and not be changed in later stages.

The second stage works on the nets without initial directives, with the goal to maximizing the sum of timing scores of nets promoted to directives higher than D_1. First, all of the layers are projected to a 2D GRG and 2D-routing is performed. Next, we perform directive assignment. Here we use a greedy method. First, all the nets without directives are sorted by their timing scores in a decreasing order, and then each net is tried to be promoted to higher directives. For each net, directives are tried one by one from D_M to D_1. This trial repeats until a directive without causing congestion is found, or the lowest directive D_1 is reached. This process is done by calling DAA procedure on each net with D_M as the initial target directive. The intuition is that we first assume all the nets can be greedily assigned to highest directive D_M and then call the DAA procedure to find the best directive for each net.

The third stage fine-tunes the layer directives obtained from the second stage. After the second stage, a large number of nets could have been promoted to directives higher than D_1. Note that at the second stage, when performing 2D routing on these nets, no directives are assumed and the routing resources from all the layers can be used to route them. In the third stage, these nets are constrained by their directives, and then the previous unconstrained 2D routing results become inaccurate. Therefore, we must reroute some nets, and we do so using the progressive projection method. As suggested by [15], we should route the nets in the order: $D_M \rightarrow D_{M-1} \cdots \rightarrow D_1$. However, we find this order is not appropriate for the purposes of directive refinement. This can be illustrated by the following example, which shows the impact of net ordering. Consider a net A that has a small timing score and is left with directive D_1. It sits on top of a routing blockage and the only possible route for this net is to route on layers in D_3 passing a g-edge e with only one available track. Another net B, which was not assigned with a layer directive from the timing driven directive assignment step but has a higher score, may have two possible routes to choose: one passing through the same g-edge e using D_3, and another taking a path in layer directive D_2 without creating overflow. When B is processed during greedy directive assignment step, the information of net A is not seen since it has a higher score, and B is assigned to D_3. In this scenario, if we route the nets in D_3 including B first, B will take the pass through e in D_3. Then later when net A in D_1 is routed, an overflow is created. In contrast, if we route A in D_1 first, A will take resource on g-edge e, and then B would be re-assigned to D_2 and no overflow is created.

Therefore, we proceed in the order $D_1 \rightarrow D_2 \cdots \rightarrow D_{M-1} \rightarrow D_M \rightarrow D_{M-1} \cdots \rightarrow D_2$. We explain this using an example with three directives. We will first route the nets in D_1 and then perform general layer assignment (to be discussed in Section V-B1) without promotion and demotion. Next, we will route the nets in D_2 with the solution from previous step treated as existing wires. Then we will perform DAA for nets in D_2 to adjust the existing directive D_2, i.e., demote a net to D_1 if congestion occurs. Since the existing wires from the first routing subproblem are seen, the directive adjustment will be more realistic than that in the $D_3 \rightarrow D_2 \rightarrow D_1$ flow. After demotion for nets in D_2 finishes, we start the same process for nets in D_3, with all the previous routing solutions treated as existing wires. After pass D_3, we need to rip-up-reroute all the nets in D_2 and call DAA for them again including some nets just demoted from D_3, with existing wires from the nets in D_1 and D_3. Since our purpose is to generate layer directives and not to perform routing, we do not need another pass D_1 to reroute the nets.

B. Two procedures used

In this section, we will discuss two procedures used in our algorithm.

1) General layer assignment: In the third stage, general layer assignment will be performed for the nets in D_1 without layer directives. Our goal is to minimize the number of vias and keep the overflow for the 3D GRG the same as that for the 2D GRG. Several layer assignment algorithms have been proposed in the literature [3], [5], [25]. For ease of implementation, we use a greedy layer assignment algorithm, but note that any of the existing layer assignment can be used in our framework.

We first sort all the nets by the total wirelength in the nonincreasing order, and then perform layer assignment for each net in the order. The reason for this ordering is that generally, the nets with larger wirelength will take more routing resources, which implies less flexibility in layer assignment, and then should be processed earlier. For each net n_i, we will loop through each segment s on the routing path of n_i. For each segment s, we will first try to find a layer which can hold it without overflow. If such a layer cannot be found, we cut the segment and find the best layer for each edge on the segment. The best layer here is the lowest layer on which there is no overflow, or on which congestion is smallest among all the layers if all the layers have overflows. Here, we will try to assign these nets to lower layers first, since no layer directives will be assigned to them and assignment to higher layers is a wasteful use of resources there. Moreover, trying to use lower layers tends to help reduce the via count.

2) Directive assignment adjustment (DAA): This procedure is an important procedure in our algorithm, and is used in all the three stages to adjust the given directive of a net. The inputs are a net n_k and its initial target directive D_j, and the output is the adjusted directive D_k. Given a net n_k with D_j, DAA will first attempt to assign the net to the layers specified by D_j using the general layer assignment procedure, with two differences. First, this method uses a constrained layer range, rather than allowing assignment to any layer. Second, the purpose of this attempt is just to quickly check whether this assignment will cause overflow, and therefore, this attempt assignment will stop at once if overflow is found on a g-edge, instead of continuing to complete the assignment of the whole net. If the attempt of assigning n_k to D_j finally succeeds without overflow, the routing resources consumed by the net will be added to 3D GRG, and directive D_j will be returned. Otherwise, D_{j-1} will be tried with the same procedure. This process repeats until n_k can be assigned to a directive D_k without overflow, or the directive D_1 is reached.

VI. Experimental results

CATALYST has been implemented using C++ and TCL in an industrial physical synthesis tool. This section presents the experimental analysis on a set of high-performance industrial designs described in Table I. The first four letters of the circuit name represents its technology node. The circuits with "top" in the names are top-level designs (designs at the first level hierarchy where a majority of gates are buffers), while the rest of circuits are random-logic macros. All the experiments run on 64-bit Linux servers with 32 CPUs (Xeon$^{®}$ X7560 2.27GHz). The parameters used in the stopping criterion of CATALYST are: $\theta = 10\%$ and $n_{itr} = 2$.

TABLE I
Information for benchmark circuits. The column "Thick layers" lists thick layer distribution.

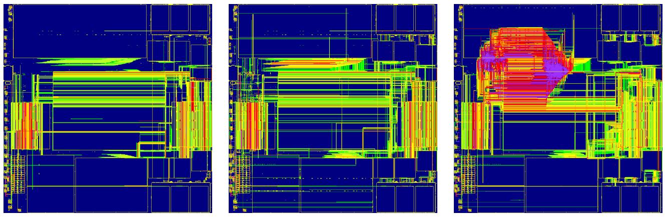
Circuits	#gates	#nets	Thick layers
cu32top1	329,082	467,889	[M6, M7]: 2X; [M8, M9]: 4X; [M10, M11]: 16X
cu32rlm2	1,392,744	1,505,994	[M6, M7]: 2X; [M8, M9]: 4X;
cu32rlm3	892,452	935,582	[M6, M7]: 2X; [M8, M9]: 4X;
cu45top1	45,655	76,062	[M6, M8]: 2X; [M9, M10]: 10X
cu45rlm2	2,464,339	2,555,753	[M6, M8]: 2X; [M9, M10]: 10X
cu45rlm3	1,282,736	1,405,029	[M6, M8]: 2X;
cu65rlm1	895,334	916,865	[M5, M7]: 2X; [M8, M9]: 4X

We first demonstrate the immediate impact on timing and congestion of CATALYST by comparing different LDA algorithms in Section VI-A, and then present the impact of CATALYST in the physical synthesis flow in Section VI-B. A full-blown industrial global router using a different routing algorithm than that used in CATALYST, is used to evaluate the congestion using the PWC metric. Note that in all of our experiments, some routing resources are reserved for the power grid and for clock routing. In addition, all timing numbers are generated using an industrial static timing analyzer.

A. The immediate impact of CATALYST

We compare three cases here, just before the global buffering phase: first, the traditional flow with no LDA (baseline); second, the results of CATALYST (denoted as "CATA"); and third, the results of an alternative LDA method (simpLDA). For the third set of results, since no public tools to generate layer directives are available, we create a simple layer directive assignment algorithm, simpLDA, which promotes nets only based on their worst slacks. We first sort all the nets based on their worst slacks in increasing order, and then promote the first n_M nets to the top directive D_M, the next n_{M-1} nets to the directive D_{M-1}, and so on. For a fair comparison, we ensure the number of nets promoted to each directive is the same for simpLDA and CATALYST. In this set of experiments, given the same placements, we run simpLDA and CATALYST and then evaluate the timing and congestion. Since this is done before global buffering, the linear delay model is used in computing all the timing metrics in this set of results.

Table II presents the comparison of timing and congestion results among the three cases, which shows the state just prior to global buffering. The row "Average" lists the average on differences from baseline (for WSLK and FOM), or the ratios to baseline (for n_{neg} and PWC), or real values (for r_{net} and r_{wl}) over all the circuits. The congestion entries shown as "N/A" ("not available") correspond to cases where global routing cannot finish after running 6 hours due to high congestion and is terminated manually. On average, CATALYST improves the worst slack by 0.84 ns, improves FOM by 6370.8 ns, and reduces the number of negative endpoints by 77%, while maintaining similar congestion to the baseline. On the other hand, although simpLDA improves the timing more than CATALYST, it degrades the congestion significantly. Fig. 5 shows the congestion plots for baseline, CATALYST and simpLDA on cu45top1. From the pictures, we can see that even after promoting 46% nets to higher directives, the congestion of CATALYST is still similar to that of baseline, while that of simpLDA is much worse. The analysis demonstrates the effectiveness of CATALYST in improving timing and controlling congestion. Though simpLDA promotes the same number of nets as CATALYST, the congestion is quite different, which shows that assigning which nets to higher directives/layers is important, and improper choices can choke the router.



(a) Baseline. (b) CATALYST. (c) simpLDA.

Fig. 5. Congestion plots for cu45top1.

B. The impact of CATALYST in the flow

This section discusses the impact of CATALYST when embedded in the physical synthesis flow. We compare three flows: baseline flow (Fig. 2(a)) that integrates the directive assignment algorithms from [1] in the global buffering stage, CATALYST flow (Fig. 2(b)) that adds CATALYST to the baseline flow after placement stage, and an altered flow that removes the two set of directive assignment algorithms, denoted as "NoDA" flow. Note that the baseline flow is the state of the art, corresponding to a relatively recent paper [1]. Prior to this, a NoDA-like flow was widely used and possibly is still in use. The comparison with NoDA highlights the impacts of directive assignment and reveals some data not found in the previous papers including [1], such as the overall impact of directive assignment through the design flow. Here simpLDA is not tested in the whole flow since it has already been seen to provide unacceptably high congestion. For runtime consideration, we stop the flows after global routing, and then evaluate the timing and congestion of the designs. At this stage, the accurate RICE model is used in timing analysis.

TABLE II

COMPARISON AMONG BASELINE, SIMPLDA AND CATALYST (CATA IN SHORT). TIMING METRICS ARE COMPUTED WITH THE LINEAR DELAY MODEL. r_{net} IS THE PERCENTAGE OF NETS PROMOTED TO THICK LAYERS, AND r_{wl} IS THE PERCENTAGE OF THE ROUTED WIRELENGTH OF THESE NETS TO THE TOTAL WIRELENGTH.

Circuit	WSLK (ns)			FOM (ns)			n_{neg}			PWC (%)			r_{net} (%)	r_{wl} (%)
	Base	CATA	simpLDA	Base	CATA	simpLDA	Base	CATA	simpLDA	Base	CATA	simpLDA	CATA	CATA
cu32top1	-14.46	-11.56	-2.95	-44794.2	-16870.0	-2118.6	87739	36270	22339	96.83	94.97	127.56	30.52	52.39
cu32rlm2	-1.93	-1.76	-1.78	-8861.6	-1031.6	-617.4	17547	7414	4093	88.49	89.87	295.58	5.67	24.41
cu32rlm3	-1.51	-0.41	-0.37	-5125.0	-264.1	-183.1	19175	2763	1626	87.42	88.36	198.39	8.63	36.11
cu45top1	-2.41	-2.39	-2.17	-2513.0	-2074.1	-1985.5	3026	967	960	87.44	86.95	121.62	46.49	73.76
cu45rlm2	-1.48	-0.74	-0.30	-2464.1	-134.1	-38.0	10359	1055	398	87.13	87.51	N/A	12.01	30.20
cu45rlm3	-0.33	-0.05	-0.05	-583.6	-4.7	-4.7	7600	258	258	87.58	87.26	N/A	19.41	24.05
cu65rlm1	-1.00	-0.37	-0.37	-726.8	-94.4	-113.5	7521	1323	1368	89.92	89.85	661.32	17.34	43.92
Average	0	0.84	2.16	0	6370.8	8572.5	1	0.23	0.16	1	1.00	3.13	20.01	40.69

TABLE III

COMPARISON AMONG BASELINE, CATALYST (CATA IN SHORT) AND NODA PHYSICAL SYNTHESIS FLOWS.

Circuit	WSLK (ns)			FOM (ns)			n_{neg}			PWC (%)			Buffer area ($\times 10^6$)			CPU time (h)		
	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA
cu32top1	-2.94	-2.96	-9.90	-2488.75	-2564.09	-23497.00	21982	21251	73652	90.13	90.05	72.61	9.54	7.78	14.43	59.7	44.7	60.2
cu32rlm2	-0.16	-0.14	-0.52	-18.87	-7.93	-1333.68	783	442	7001	88.02	87.70	88.51	0.82	0.78	1.17	34.1	42.4	53.0
cu32rlm3	-0.04	-0.02	-0.78	-0.15	-0.03	-408.34	6	2	2004	87.53	87.66	93.37	1.03	0.93	1.58	17.3	19.2	22.0
cu45top1	-2.42	-2.42	-2.60	-2139.97	-2130.29	-3187.95	963	963	4720	80.70	81.07	73.73	2.46	2.09	2.91	10.6	11.9	10.2
cu45rlm2	-0.34	-0.38	-1.45	-23.03	-17.03	-5258.76	336	109	21037	87.39	86.78	87.13	4.00	3.55	4.67	63.8	67.1	76.7
cu45rlm3	-0.05	-0.05	-0.35	-12.01	-3.37	-512.86	521	163	6924	87.96	87.85	89.15	3.99	3.85	4.75	37.5	41.5	54.6
cu65rlm1	-0.24	-0.24	-0.75	-4.30	-4.21	-68.38	153	154	749	89.79	89.57	91.94	1.21	1.15	1.53	15.3	17.1	19.3
Average	0	0.00	-1.45	0	-5.69	-4225.70	1	0.64	61.71	1	1.00	0.98	1	0.90	1.32	1	1.07	1.24

Table III presents the comparison of timing, congestion, buffer area, and CPU time at the end of the flow. The row "Average" lists the average on differences from baseline (for WSLK and FOM) or the ratios to baseline (for all other metrics) over all circuits. We first compare the CATALYST flow with baseline. Though WSLK and FOM are quite similar for two flows, the CATALYST flow reduces the n_{neg} by 36% on average, which indicates fewer further efforts are required to finally close timing. Moreover, the CATALYST flow can reduce the buffer area by 10% on average and up to 18%[4]. This indicates that the timing optimization techniques later in the baseline flow, other than CATALYST, can also achieve similar WSLK and FOM to the CATALYST flow, but with an expense of inserting more buffers, and a series of other consequences such as higher cost, larger power, etc. We also observe that the buffer saving tends to be larger for the top-level designs than macros (17% vs. 7% on average), since in top level designs, a larger portion of nets are very long nets, and without layer directive assignment, more buffers have to be inserted to close the timing. Furthermore, the CATALYST flow achieves similar congestion to baseline, which demonstrates again that CATALYST has better control on the congestion. Finally, the runtime of CATALYST flow is 7% more than baseline on average, and even for the absolute values, the runtime of CATALYST flow is still acceptable in practice. Note that for cu32top1, CATALYST runs 15 hours faster than baseline. This indicates that for some designs, a better initial solution obtained by CATALYST can speed up the later optimization stages significantly.

Next we further compare NoDA flow with the other two. The timing, buffer area and runtime of NoDA flow are significantly worse than those of the other two flows. Compared with the CATALYST flow, NoDA requires 47% more buffers (or CATALYST flow could save 31% than NoDA) on average.

The analyses above clearly show the impact of CATALYST: it can improve the timing and greatly reduce the buffer area. Since buffers could account for more than 20% of the gates in a modern design [17], the reduction of buffer usage can significantly decrease the design power and cost, and improve the design closure.

REFERENCES

[1] Z. Li *et al.*, "Fast interconnect synthesis with layer assignment," in *Proc. ISPD*, 2008, pp. 71–77.

[2] C. Alpert *et al.*, "What makes a design difficult to route," in *Proc. ISPD*, 2010, pp. 7–12.

[3] Y. Xu *et al.*, "FastRoute 4.0: Global router with efficient via minimization," in *Proc. ASPDAC*, 2009, pp. 576–581.

[4] Y.-J. Chang *et al.*, "NTHU-Route 2.0: A fast and stable global router," in *Proc. ICCAD*, 2008, pp. 338–343.

[5] M. Cho *et al.*, "BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router," in *Proc. ICCAD*, 2007, pp. 503–508.

[6] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," *IEEE Trans. on CAD*, vol. 27, no. 11, pp. 2017–2026, 2008.

[7] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proc. ICCAD*, 2007, pp. 496–502.

[8] H.-Y. Chen *et al.*, "High-performance global routing with fast overflow reduction," in *Proc. ASPDAC*, 2009, pp. 582–587.

[9] P. Saxena and C. L. Liu, "Optimization of the maximum delay of global interconnects during layer assignment," *IEEE Trans. on CAD*, vol. 20, no. 4, pp. 503–515, 2001.

[10] Y. Jia *et al.*, "Timing driven layer assignment considering via resistance and coupling capacitance," in *Proc. ICCCAS*, 2007, pp. 1172–1176.

[11] S. Hu *et al.*, "A polynomial time approximation scheme for timing constrained minimum cost layer assignment," in *Proc. ICCAD*, 2008, pp. 112–115.

[12] ——, "A fully polynomial-time approximation scheme for timing-constrained minimum cost layer assignment," *IEEE Trans. on CAS II*, vol. 56, no. 7, pp. 580–584, 2009.

[13] Y. Chang *et al.*, "GLADE: a modern global router considering layer directives," in *Proc. ICCAD*, 2010, pp. 319–323.

[14] T. Lee *et al.*, "An enhanced global router with consideration of general layer directives," in *Proc. ISPD*, 2011, pp. 53–60.

[15] M. D. Moffitt and C. N. Sze, "Wire synthesizable global routing for timing closure," in *Proc. ASPDAC*, 2011, pp. 545–550.

[16] L. Trevillyan *et al.*, "An integrated environment for technology closure of deep-submicron IC designs," *IEEE Design & Test of Computers*, vol. 21, no. 1, pp. 14–22, 2004.

[17] C. J. Alpert *et al.*, "Techniques for fast physical synthesis," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 573–599, 2007.

[18] Y. Wei *et al.*, "GLARE: global and local wiring aware routability evaluation," in *Proc. DAC*, 2012, pp. 768–773.

[19] N. Viswanathan *et al.*, "The DAC 2012 routability-driven placement contest and benchmark suite," in *Proc. DAC*, 2012, pp. 774–782.

[20] C. L. Ratzlaff and L. T. Pillage, "RICE: rapid interconnect circuit evaluation using AWE," *IEEE Trans. on CAD*, vol. 13, no. 6, pp. 763–776, 1994.

[21] J. Cong and D. Z. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Proc. ASPDAC*. IEEE, 1999, pp. 97–100.

[22] C. J. Alpert *et al.*, "Accurate estimation of global buffer delay within a floorplan," *IEEE Trans. on CAD*, vol. 25, no. 6, pp. 1140–1145, 2006.

[23] D. Papa *et al.*, "RUMBLE: an incremental timing-driven physical-synthesis optimization algorithm," *IEEE Trans. on CAD*, vol. 27, no. 12, pp. 2156–2168, 2008.

[24] S. S. Sapatnekar, *Timing*. Boston, MA: Kluwer Academic Publishers, 2004.

[25] T. H. Lee and T. C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Trans. on CAD*, vol. 27, no. 9, pp. 1643–1656, 2008.

[4] There are some buffers that are fixed in the designs and cannot be changed.