NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing with Bounded-Length Maze Routing

Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao

***Abstract*—Modern global routers employ various routing methods to improve routing speed and quality. Maze routing is the most time-consuming process for existing global routing algorithms. This paper presents two bounded-length maze routing (BLMR) algorithms (optimal-BLMR and heuristic-BLMR) that perform much faster routing than traditional maze routing algorithms. In addition, a rectilinear Steiner minimum tree aware routing scheme is proposed to guide heuristic-BLMR and monotonic routing to build a routing tree with shorter wirelength. This paper also proposes a parallel multithreaded collision-aware global router based on a previous sequential global router (SGR). Unlike the partitioning-based strategy, the proposed parallel router uses a task-based concurrency strategy. Finally, a 3-D wirelength optimization technique is proposed to further refine the 3-D routing results. Experimental results reveal that the proposed SGR uses less wirelength and runs faster than most of other state-of-the-art global routers with a different set of parameters [12], [16], [17], [20]. Compared to the proposed SGR, the proposed parallel router yields almost the same routing quality with average 2.71 and 3.12-fold speedup on overflow-free and hard-to-route cases, respectively, when running on a 4-core system.**

***Index Terms*—global routing, maze routing, multithreaded routing, physical design, rip-up and reroute.**

I. Introduction

In VLSI physical design flow, the large gap between two estimated wirelengths by placement and routing makes it harder to achieve design closure. The modern placers [1]–[4] bring fast global routing into placement stage to offer accurate wirelength estimation, implying the urgent demand of fast global routers. By holding global routing contests [5], [6], ISPD'07 and ISPD'08 have attracted many researchers to develop several global routers [7]–[13], [16]–[20]. Most of these routers apply the negotiation-based rip-up and rerouting, which is first introduced in PathFinder [21]. They focus on minimizing overflow first, and then wirelength and runtime. Maze routing is a slow but the most important algorithm to seek a feasible or better connection after other routing algorithms fail in each global router. Many global routing papers design various types of routing cost functions to balance total wirelength and total overflows (TOs), such as Lagrange multipliers in FGR [7], dynamic base cost in NTHU-Route2.0 [12], and two-stage cost functions in NCTU-GR [17], but they have no specific way to control the increase in wirelength.

As multicore architecture has become the mainstream of CPU design, the development of EDA algorithms on a multicore platform can boost performance and quality in solving EDA problems [23]. In parallel routing, a collision is said to occur if multiple routing threads simultaneously demand a routing edge. To avoid collision, the partitioning-based concurrency strategy partitions a routing graph into several independent regions, and tries to process them simultaneously. However, in global routing, the passing routing regions of a net often overlap with those of other nets, making it hard to achieve a good independent region partitioning and a quite balanced loading among all subproblems. Thus, the speedup of this strategy tends to be limited. A parallel global router (PGR) GRIP [18], [19] performing on a cluster computing platform is based on partitioning-based strategy, so the balanced loading problem of GRIP is a vital issue in GRIP. GRIP obtains the best wirelength among all open literature, but GRIP requires prohibitive runtime to complete global routing as compared to other modern global routers. To avoid the limitations of the partitioning-based strategy, [22] reduced the global routing problem into a min–max resource sharing problem and, then, developed a parallel algorithm to solve this problem.

The first part of this paper addresses the BLMR problem, which involves identifying a minimal-cost path from a net's source to target with a specified length constraint. In addition, this paper develops a parallel multithreaded global router adopting the proposed task-based concurrency strategy (TCS) on a multicore platform based on the proposed sequential global routing algorithm. This paper makes the following contributions to global routing research.

1) The proposed heuristic-BLMR algorithm with a new *history-length* scheme improves the accuracy of estimated wirelength so that a better runtime speedup and wirelength decrease is obtained.
2) The proposed rectilinear Steiner minimum tree (RSMT) aware routing scheme can guide the proposed router to build a routing tree with shorter wirelength.
3) An efficient collision-aware rip-rip and rerouting scheme is proposed to resolve the problem of routing quality degradation caused by thread collision.
4) The proposed 3-D wirelength optimization technique can yield the 3-D routing result of similar routing quality to [18] and [19] in a reasonable runtime.

Manuscript received January 13, 2012; revised June 17, 2012, September 4, 2012, October 27, 2012; accepted November 8, 2012. Date of current version April 17, 2013. This paper was recommended by Associate Editor C. J. Alpert.

W.-H. Liu, W.-C. Kao, and Y.-L. Li are with Department of Computer Science, National Chiao-Tung University, Hsin-Chu 30010, Taiwan (e-mail: dnoldnol@gmail.com; owokko@gmail.com; ylli@cs.nctu.edu.tw).

K.-Y. Chao is with Enterprise Microprocessor Group, Intel Corporation, Hillsboro, OR 97124 USA (e-mail: kaiyuan.chao@intel.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2235124

The rest of the paper is organized as follows. Section II briefly describes the preliminary of global routing. Section III presents a global router that adopts BLMR, RSMT-aware routing scheme and dynamically adjusted history cost function. Section IV develops a parallel global routing algorithm based on the TCS. Section V presents a 3-D wirelength optimization technique. Section VI summarizes the experimental results. Finally, Section VII draws conclusions.

II. Background

A. 3-D Global Routing Problem

The 3-D global routing is formulated as the routing problem on a 3-D grid graph $G(V, E)$. Typically the layout is partitioned into an array of global cells. Each grid node in the grid graph refers to a global cell, in which each grid edge corresponds to a boundary between two abutting global cells in the same layer. Meanwhile, each via edge connects two abutting global cells in two adjacent layers. The number of routing tracks that can be accommodated across the abutting boundary is defined as the capacity $c(e)$ of a grid edge e, and the number of wires that pass through e is called grid edge's demand $d(e)$. The overflow of e, $overflow(e)$, is defined as $\max(0, d(e)–c(e))$. The TO is the sum of overflows on all grid edges, and the maximum overflow (MO) is the MO among all edges. For simplicity, the capacity of each via edge is not limited, which is also adopted in most of global routing researches [7]–[20], [30]. A global router largely focuses on producing a highly routable global path for every net. The metrics of routability can be measured based on the congestion of all global edges and wirelength of every net. Few TOs and short total wirelength imply high routability.

The approach used in this work condenses 3-D grid graph into 2-D grid graph, and then adopts 2-D global routing to obtain a 2-D routing result. Finally, layer assignment assigns each net edge to the corresponding metal layer to obtain final 3-D routing results [7]–[17], [30].

B. Net Decomposition

The RSMT and rectilinear minimum spanning tree (RMST) construction algorithms are commonly used to decompose multipin nets into two-pin subnets before routing stages. FLUTE [24] can quickly construct optimal RSMTs for nets with nine or fewer pins, and is widely used by many global routers. However, the RSMT has less routing flexibility than the RMST as it owns Steiner points and generates more flat segments than the RMST, and the data structure of RSMTs is more complex than that of RMSTs [7]. On the contrary, the RMST can simply complete each subnet's routing with pattern or monotonic routing to avoid congestion regions. Consider wirelength and routing flexibility, in which a RMST that encourages two two-pin routings to merge together with two paths that pass through the same grid edges. This ideal solution avoids passing through congested regions by using a shorter total wirelength than that of a RMST that does not encourage finding joint wires. However, identifying a RMST with joint wires is a challenge.

C. Negotiation-Based Rip-Up and Rerouting (NRR)

Rip-up and re-routing technique is widely used in global and detailed routing to expel all violations (overflows in global routing). The negotiation technique in [21] is widely associated with rip-up and re-routing technique (NRR) in modern global routers to reduce overflows. The main idea of NRR is to increase the penalty of a grid edge at current iteration that overflowed at the previous iteration. Thus, path searching intends to avoid passing previously overflowed grid edges. McMurchie and Ebeling [21] formulates the negotiation-based routing cost of grid edges e as follows:

$$\text{cost}(e) = (b_e + h_e) \times p_e \tag{1}$$

where $\text{cost}(e)$, b_e, h_e, and p_e denote the routing cost, the base cost, the history cost, and the congestion penalty of e, respectively. The history cost h_e increases as overflow occurs. The value of h_e in the $(k + 1)$th iteration is given by

$$h_e^{k+1} = \begin{cases} h_e^k + h_{inc} & \text{if } e \text{ is overflowed} \\ h_e^k & \text{otherwise} \end{cases} \tag{2}$$

where $h_e^1 = 1$, h_{inc} is a constant, and h_e^k is updated in every iteration. In addition, FGR [7] presents another formula to preserve the base cost as follows:

$$\text{cost}(e) = b_e + h_e \times p_e. \tag{3}$$

Several variations of negotiation-based cost functions have been discussed in [11]–[13], [16], and [17].

III. Proposed Global Router Using BLMR

This section introduces three kernel techniques of the proposed router, i.e., BLMR, RSMT-aware routing scheme, and dynamically adjusted history cost. The final subsection summarizes the detailed design flow of the proposed router.

A. BLMR

Many global routers adopt a bounding box to limit the searching region of maze routing to accelerate maze routing, and gradually relax the bounding box if an overflow-free routing solution cannot be found. However, maze routing may produce many detours or fail to find a short path within the bounding box. Thus, this paper develops BLMR to speed up maze routing by limiting the search region as well as to well improve routing resource utilization by lessening redundant wirelength. The BLMR problem is formulated as follows. In a 4-tuple (s, t, G, L), s and t denote a source and a target, respectively; G denotes the grid graph, each grid edge in G has specified congestion cost, and L denotes the bounded-length constraint (BLC) (L is not less than the Manhattan distance between s and t). The objective of the BLMR problem is to identify a minimal-cost path from s to t on G, and the wirelengh of the path cannot exceed L.

BLMR problem can be regarded as a restricted version of constrained shortest path (CSP) problem. In general CSP problem, a delay and a cost for every edge in a graph are specified. The delay of an edge may be the length of the edge or the signal latency from a terminal of the edge to another

terminal. CSP algorithm attempts to identify a minimal-cost path from s to t, ensuring that the total delay of the identified path does not exceed an upper bound. The general version of CSP problem is NP-complete when the delay of each routing edge is a real number [25]. However, this problem can be solved in polynomial time if the delay of the each routing edge is an integer [26]. Several studies [25]–[27] have addressed this problem for its applications in quality of service area.

Compared to CSP problem, BLMR problem owns some particular properties that are not in the general CSP problem. For instance, while CSP algorithm works on a general graph, BLMR algorithm works on a grid graph of specific 2-D array structure with each vertex having at most four neighbors, which makes BLMR algorithm may be faster than CSP algorithm on solving global routing problem. Also, global routing problem owns the following particular properties.

1) The graph in global routing is a grid graph, and the distance from any node to the target can be estimated by simply Manhattan distance. Thus, a path possibly violating length constraint can be detected in the constant time.
2) The region to explore in a grid graph during routing can be restricted within a specified area; however, the studies [26] and [27] scan entire routing graph, and thus consume much time.
3) In global routing, a net may be ripped up and re-routed several times to identify its final path. The paths of a net identified in two successive rerouting iterations make use of most routing edges in common. The proposed *history-based estimated wirelength* scheme utilizes this property.

The definitions of the notations used in following sections are listed as follows. $P(s, v)$ denotes a path from source s to node v, $w_l(P(s, v))$ denotes the wirelength of $P(s, v)$, $p_c(P(s, v))$ denotes the routing cost of $P(s, v)$, and $Manh(v, u)$ is the Manhattan distance from v to u. Notably, the proposed BLMR algorithm adopts $A*$ search scheme for acceleration of path search. The search key is $p_c(P(s, v))$ comprising the routed cost from s to the current node v and the estimated lower bound cost from the v to target t. Path length in BLMR is a constraint and all paths that violate this constraint will be pruned.

1) *Optimal-BLMR:* Optimal-BLMR algorithm adopts two different policies than traditional maze routing to obtain a minimum-cost routing solution under BLC. First, we define the potential wirelength (*pwl*) for each incomplete routing path $P(s, v)$ as the sum of $w_l(P(s, v))$ and $Manh(v, t)$ where v is an currently explored grid node, and a path with *pwl* exceeding L is regarded as a path violating BLC. Optimal-BLMR discards the paths violating BLC, and then restricts the searching region. Fig. 1(a) shows the searching region of a net on the graph while L is set to 9. Second, assuming that there are two or more paths from s to v, traditional maze routing only preserves the minimum-cost path and discards others. However, in optimal-BLMR, this scheme does not guarantee to identify a feasible solution because the length slack of the minimum-cost path may be less than the wirelength required to detour around congested regions, where the length slack of $P(s, v)$ is L minus

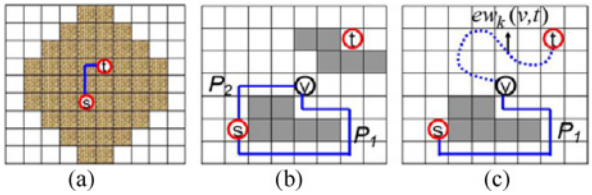

Fig. 1. (a) Search region of the net while L is set to 9. (b) Two path candidates P_1 and P_2 from s to v. (c) $ew_k(v, t)$ represents estimating wirelength from v to t in iteration k..

$w_l(P(s, v))$. For instance, Fig. 1(b) shows two path candidates P_1 and P_2 from s to v: the gray regions are congested regions, the bounded-length is 16, and $p_c(P_1)$, $p_c(P_2)$, $w_l(P_1)$, and $w_l(P_2)$ are 80, 90, 11 and 5, respectively. If optimal-BLMR only preserves the minimum-cost path P_1, the length slack of P_1 is 5, which is too small to detour around congested regions to reach t. Because the wirelength from v to t is uncertain before the end of routing, optimal-BLMR must preserve both paths. However, if the following inequalities hold, P_1 is considered to be inferior to P_2 and can be discarded.

$$w_l(P_1) \geq w_l(P_2) \quad \text{and} \quad p_c(P_1) \geq p_c(P_2). \tag{4}$$

$S(v)$ is a *list* of node v to store the paths from the source s to node v, any two of which do not conform to (4). For each $v \in V$, $S(v)$ is initially set as empty. While storing all currently explored paths, a Fibonacci heap H is initialized to have only s. At the beginning of each routing iteration, the minimum-cost path is selected from H for further routing. The optimal-BLMR algorithm is designed as follows.

1) Extract the minimum-cost path $P(s, v)$ from H. If v is t, return $P(s, t)$ as the solution and exit.
2) Explore each neighboring node of v, say node u. If the newly explored path $P(s, u)$ does not conform to BLC, discard $P(s, u)$; otherwise perform Step 3. Go back to Step 1 after exploring each neighbor of v.
3) Scan every path candidate in $S(u)$ and remove the inferior paths to $P(s, u)$ from $S(u)$ and H. If $P(s, u)$ is not inferior to any path in $S(u)$, $P(s, u)$ is inserted into $S(u)$ and H.

Notably, if a net lacks adequate length slack to detour around all congestions to reach the target, optimal-BLMR would identify a routing path passing through congestions. A net passing through overflowed grid edges is called overflowed net, which will be rerouted in the next iteration.

Step 3 of optimal-BLMR takes the time complexity of $O(|S(u)|)$ to scan every path in $S(u)$, where $|S(u)|$denotes the number of path candidates in $S(u)$. The maximum size of $S(u)$ is derived as follows.

Lemma 1: For an optimal-BLMR subject to the constraint $Manh(s, u)+Manh(u, t) \leq L$, *where s, t, and u are respectively the source, the target and an intermediate node, the maximum size of* $S(u)$ *is* $\lceil (L–Manh(s, u)–Manh(u, t) + 1)/2 \rceil$.

Proof: The shortest path candidate in $S(u)$ is the path of wirelength $Manh(s, u)$, and the longest path candidate in $S(u)$ must not exceed $L–Manh(u, t)$ due to BLC. Thus, the paths in $S(u)$ have at most $((L–Manh(u, t))–Manh(s, u)+1)$ possible wirelength levels. Moreover, because a detour increases the

wirelength by two in unit of grid edge, the number of possible wirelength levels becomes $(L–Manh(u, t)–Manh(s, u)+1)/2$. When (4) is adopted to prune the inferior paths, $S(u)$ only reserves the lowest cost path in each wirelength level. Hence, the maximum size of $S(u)$ is $(L–Manh(u, t)–Manh(s, u)+1)/2$. ■

2) *Heuristic-BLMR:* Although the proposed heuristic-BLMR approach cannot guarantee the optimal solution, it is much faster than optimal-BLMR. The difference between optimal-BLMR and heuristic-BLMR is that heuristic-BLMR preserves only one path from the source to the current node, and the other paths are discarded. Accordingly, the major issue of heuristic-BLMR is determining which path candidate should be preserved.

Path selection involves examining each path to determine whether or not the required wirelength to bypass the congested regions from the current node v to target t does not exceed the length slack, then heuristic-BLMR preserves the minimal-cost path with enough length slack. If no path candidates have enough length slack, the shortest path candidate is preserved because the shortest path has a greater chance to bypass the congested regions. However, the congestion information from v to t is not explored yet, hence the *history-based estimated wirelength* of the path from v to t is estimated as follows:

$$ew_k(v, t) = HL_{k-1}(s, t) \times \frac{Manh(v, t)}{Manh(s, t)} \tag{5}$$

where $ew_k(v, t)$ is the history-based estimated wirelength from v to t in iteration k [Fig. 1(c)], k denotes the iteration number of the NRR stage, and $HL_{k-1}(s, t)$ is the history length, i.e., actual routed wirelength from s to t in iteration $k–1$. The concept behind (5) is that the length from v to t is proportional to the length from s to t at previous iteration. Based on (5), heuristic-BLMR predicts that $P(s, v)$ has sufficient length slack to bypass the congested regions from v to t if the following equation holds

$$w_l(P(s, v)) + ew_k(v, t) \leq L. \tag{6}$$

If multiple paths conform to (6), the minimum-cost path is preserved. If no path conforms to (6), the shortest path is preserved. This policy ensures that a path is only preserved to greatly reduce the number of explored routing paths during heuristic-BLMR. If (5) overestimates the wirelength from v to t in the previous iteration, the long path candidates tend to be discarded by (6), which selects the short path candidates as the final routing path. Hence the estimated wirelength from v to t using (5) at current iteration decreases. Similarly, if (5) underestimates the wirelength in the previous iteration, the estimated wirelength will increase in the current iteration. As a result, as the iteration number increases, (5) gradually becomes more accurate in estimating the actual wirelength from v to t.

This paper evaluates the accuracy of (5) by the following experiment. As heuristic-BLMR identifies a routing path at iteration k, an internal node v is selected randomly in the identified path and, then, the difference between $HL_k(v, t)$ and $ew_k(v, t)$ is computed. According to the experiment on benchmark adaptec1, the average difference of all nets is 54% at the first iteration of the NRR stage; the average difference then gradually decreases to 30% at iteration five. Following iteration five, the average difference swings between 25% and 30%. However, if only the average difference of the first 20% long nets is calculated, the average difference ranges from 10% to 20% after iteration five, implying that (5) more accurately estimates long nets than short nets.

While closely resembling each other, the heuristic-BLMR algorithm and optimal-BLMR differ only in that the former always preserves at most one path in $S(v)$, $v \in V$. When $S(v)$ already contains a path and a new path $P(s, v)$ is explored, the path selection scheme proposed in this subsection chooses the proper path to be preserved in $S(v)$ and H. Inaccurate estimation by (5) in heuristic-BLMR may yield unnecessary overflows, increasing the required iterations to remove unnecessary overflows.

3) *Bounded-Length Relaxation:* In the NRR stage of this paper, BLC is designed to control the routing resource utilization. Initially, the routed wirelength is strictly limited since overusing routing resources in the early stage likely inhibits subsequent routings from finding overflow-free paths, and increases runtime as well. While a routing cannot avoid congested regions under the strict BLC, BLC is gradually relaxed to encourage heuristic-BLMR to yield fewer overflows at the expense of a longer wirelength as the process iteration proceeds. However, overly relaxing BLC in a later stage only gives rise to a large increase in the wirelength, yet cannot help to resolve overflows. A bounded-length relaxation scheme is thus formulated as follows:

$$L_n^k = Manh(s_n, t_n) \times (1 + \arctan(k - \alpha) + \beta) \tag{7}$$

where L_n^k is the bounded-length of two-pin net n in the k-th routing iteration. The first term is the Manhattan distance between two terminals of n, and the second term is the scaling factor of the bounded-length; α and β are user-defined positive constants. As for fundamental discussion about the relation between the routing iteration number and the scaling factor, please refer to [30, Section III-C].

The runtime and routing quality of the proposed router are impacted by the values of α and β. The value of α refers to the number of iterations in phase 1, while the value of β refers to the search region of each net. Increasing α encourages the proposed router using shorter wirelength to eliminate overflows, thus the proposed router can yield the final result with less wirelength at the cost of longer convergence period. Increasing β encourages an increase in the wirelength of the routing result with the number of routing iterations to be decreased. For the hard-to-route cases, large α and small β are effective for the proposed router to avoid producing too much wirelength and consuming additional routing resources at early stage, which increases the difficulty of eliminating overflows. For the easy-to-route cases, small α and large β are effective for the proposed router to complete the routings. In this paper, α and β are set to 9 and 1.5, respectively.

B. RSMT-Aware Routing Scheme

Regarding the ability of the proposed scheme to decompose a net into multiple two-pin subnets, the proposed RSMT-aware

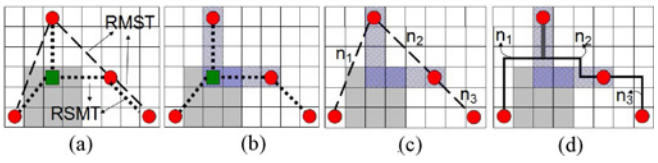

Fig. 2. (a) FLUTE and Kruskal's algorithm are employed to yield a RSMT and a RMST. (b) Grid edges in the shadow regions are the skeleton edges of this net. (c) RMST is combined with *SE*. (d) RSMT-aware routing result.

routing scheme is characterized by RSMT and RMST. In the RMST decomposition stage, each net is first decomposed into several two-pin nets by RMST. Thereafter, the RSMT-aware routing scheme first constructs one RSMT for reference and then encourages the routing of each subnet to pass through the regions passed by RSMT. The terminals of a RSMT consist of pins and Steiner points. Two connected terminals are linked with a straight path or an L-shape path. For a RSMT of net N, a grid edge e is a *skeleton edge* associated with N if e is passed by a straight path of the RSMT. $SE(N)$ denotes the set of *skeleton edges* associated with N.

Fig. 2(a)–(d) illustrates the three steps to build RSMT-aware routing scheme for a four-pin net N. Three steps are performed at the stages of RMST decomposition and RSMT-aware scheme construction. Then, during the monotonic routing stage, the NRR stage and the postrefinement stage, BLMR, and monotonic routings use this scheme to evaluate the routing cost of grid edges. The details are listed as follows.

1) FLUTE and Kruskal's algorithm are first employed to yield a RSMT as the ideal routing tree and a RMST to decompose net N into several two-pin nets [Fig. 2(a)].
2) Identify skeleton edge set $SE(N)$ [shadow regions in Fig. 2(b)] using the identified RSMT in Step 1.
3) Each two-pin net of RMST is associated with $SE(N)$ to form the RSMT-aware scheme. This association is used as follows. In Fig. 2(c). Net N's RMST is split into three two-pin nets, n_1, n_2, and n_3. The edges in $SE(N)$ will be assigned with less costs to encourage the routing wires of net N to pass through the edges in $SE(N)$ [Fig. 2(d)] when the router performs monotonic or BLMR routing of n_1, n_2, and n_3.

The proposed RSMT-aware routing scheme differs from other routers mainly in that, in the proposed scheme, a paragon RSMT is used for reference; the routing tree can then be easily amended to approach the paragon one as nearby regions become un-congested. Other routers normally refine the cost function to enable the subsequent routing to share the grid edges used in previous routing. However, for instance, if the path of n_1 in Fig. 2(d) is an upper L-shaped path, the next routing, e.g., n_2, cannot reuse the grid edges in the path of n_1. In addition, RSMT-aware routing scheme can restore the routing of a net to its RSMT at the postrefinement stage as congestions are eliminated. For instance, if the overflows (congested regions in gray color) in Fig. 2(d) are eliminated and subnets n_1 and n_2 are rerouted with RSMT-aware routing scheme, the routing path will pass through the Steiner point in Fig. 2(a) and look very similar to the RSMT in Fig. 2(a).

C. Dynamically Adjusted History Cost Function

With RSMT-aware cost function, a grid edge is assigned with different cost values to different nets to encourage a net to pass the grid edges in its skeleton edge set. For the routings of two nets, say N_1 and N_2, if grid edge e is in the skeleton edge set of N_1 but not in that of N_2, the cost of edge e for net N_1 is assigned as a less value than that for net N_2, which can better routing resource utilization by supporting each net to use the edges in its skeleton edge set. In the NRR stage, the concept of routing cost of grid edge e for net N is based on that in [12] and re-formulated as follows:

$$\text{cost}(e) = \begin{cases} 0 & \text{if } e \in GE(N) \\ (1 + dah(e,k)) \times p_e + b_e - w & \text{elseif } e \in SE(N) \\ (1 + dah(e,k)) \times p_e + b_e & \text{otherwise} \end{cases} \tag{8}$$

where cost(e), b_e, and p_e are defined in (1), $dah(e, k)$ denotes the dynamically adjusted history cost function [further discussed in (9)], $GE(N)$ denotes the set of grid edges that are passed by N, and w is a weighted constant that is set to 1 in this paper. Equation (8) encourages a route to pass through the grid edges in either $GE(N)$ or $SE(N)$.

Traditionally, the history cost of a grid edge in negotiation-based cost function scheme (2) continues to increase if the grid edge keeps congested. The history cost remains unchanged even after the grid edge becomes un-congested, which results in overestimated routing cost for the routing passing the grid edge and then generates unnecessary detours. Hence, we have to lower the history cost if the grid edge becomes un-congested. The $dah(e, k)$ is proposed as follows:

$$dah(e,k) = \frac{h_e^k}{C_1 + C_2 \times \sqrt{k}} \quad \text{where } h_e^{k+1} = h_e^k + of_e^k \tag{9}$$

where h_e^k and of_e^k, respectively, denote the history cost and overflow frequency of e at iteration k. $h_e^1 = 1$, and h_e^k is updated at the end of every iteration. C_1 and C_2 are user-defined constants and set to 7 and 4, respectively. If edge e keeps un-congested during subsequent iterations, by (9), $dah(e, k)$ will decrease as iteration number k increases while h_e^k remains unchanged. In addition, the proposed history cost is updated with overflow frequency instead of a constant value in traditional scheme (2). The overflow frequency of_e^k of grid edge e is set to zero at the beginning of each iteration k. Once an overflowed net is rerouted, of_e^k increases one if e overflows and e is passed by the rerouted net. A larger of_e^k implies a greater number of nets demanding the routing resource of e in iteration k, further implying that e is critical. The idea behind (9) is that critical routing resources should have large overflow frequency. Thus, a grid edge with high overflow frequency is assigned with large history cost. As compared to traditional history cost updating scheme, the proposed one requires less iterations to distinguish which grid edges are critical. In this paper, p_e and b_e are formulated as follows, which are inspired by [14] and [12], respectively

$$p_e = 1 + \frac{C_3}{1 + e^{C_4(c(e) - d(e))}} \text{ and } b_e = C_5 + C_6/2^k \tag{10}$$

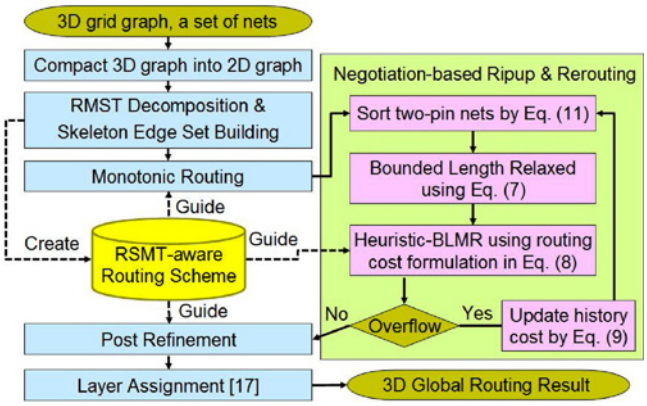

Fig. 3. Detailed design flow of the proposed global router.

where C_3, C_4, C_5, and C_6 are set to 150, 0.3, 30, and 200 in our implementation, respectively. The constant value setting is determined by closely examining how their variations impact routing results in the experiments.

D. Detailed Design Flow

Fig. 3 shows the detailed design flow. At first, the 3-D routing problem is compacted into a 2-D routing problem. The algorithm then decomposes each net to two-pin nets based on the topology of RMST, builds skeleton edge sets for each net to set up RSMT-aware routing scheme, and generates an initial congestion graph via monotonic routing that routes all two-pin nets. Next, the NRR stage iteratively reroutes the overflowed nets until an overflow-free routing result is obtained. Upon commencement of each iteration in the NRR stage, all two-pin nets are first sorted; they are then examined sequentially to determine whether an overflow occurs. BLC for the overflowed net, e.g., N, is relaxed by (7) and, then, is ripped up and rerouted by heuristic-BLMR with the routing cost function in (8), which requires testing if a grid edge belongs to $GE(N)$ or $SE(N)$. $GE(N)$ and $SE(N)$ are stored using hash table, specifically, *unordered_multiset* in C++ STL. $SE(N)$ is constructed in the RMST decomposition stage while $GE(N)$ is dynamically updated during rip-up and rerouting to record the grid edges in the routing path of net N. The time complexities of inserting, erasing and identifying a grid edge e in *unordered_multiset* remain constant. As NRR enters a new iteration, the history cost is updated by (9).

The wirelength of each net is then greedily minimized in the postrefinement stage by ripping up and rerouting each two-pin net once with the proposed RSMT-aware routing scheme. The overflow-free two-pin routings without a detour are rerouted by monotonic routing; meanwhile, the other two-pin routings are rerouted by heuristic-BLMR with the original path length as BLC. Rerouting overflow-free nets can reduce their wirelength and can vacate the routing resource to other overflowed nets as well. Because most nets are routed by monotonic routing or heuristic-BLMR with small BLC, this stage is very efficient. In this stage, the routing cost of a grid edge, say e, is formulated as follows:

$$\text{cost}(e) = 1 + \kappa \left(\frac{d(e)}{1 + c(e)} \right). \tag{11}$$

TABLE I
NET ORDERING METHODS COMPARISON

Order	adaptec1		adaptec3		adaptec5		Average	
	WL (10^5)	CPU (m)	WL (10^5)	CPU (m)	WL (10^5)	CPU (m)	WL (10^5)	CPU (m)
LenD	36.42	2.17	96.00	1.93	105.18	4.65	70.97	2.18
LenI	36.65	2.26	96.28	1.85	106.73	5.37	71.53	2.31
OFD	36.39	2.13	96.02	1.86	105.39	4.46	70.99	2.10
OFI	36.74	2.51	96.42	1.98	107.02	5.63	71.69	2.46
Equation (12)	36.31	1.78	95.97	1.77	105.14	4.06	70.88	1.90

If $c(e) \leq d(e)$, κ is set to 10^5 to avoid increasing overflows; otherwise, κ is set to 0.1. Finally, the layer assignment in [17] is employed to transform the 2-D routing result to the 3-D result.

The routing ordering of all two-pin nets impacts the routing quality and runtime. We introduce the net routing ordering adopted in each routing stage as follows. In the monotonic routing stage, the proposed router sorts the two-pin nets in the increasing order according to its bounding box size. Smaller bounding box for a net implies less solution space for monotonic routing of the net. Routing a net with less solution space earlier can improve the possibility to complete its monotonic routing due to fewer previously routed wires. In the NRR stage, wirelength and congestion control and overflow reduction are main objectives. We evaluate four net ordering methods in terms of wirelength and overflow in the NRR stage. Table I displays the wirelength (WL) and runtime (CPU) of benchmarks adaptec1, adaptec3, adaptec5, and average of all overflow-free cases when the NRR stage adopts four different net ordering in decreasing wirelength (LenD), increasing wirelength (LenI), decreasing overflows (OFD), and increasing overflows (OFI). Notably, WL here does not include vias because layer assignment has not yet been performed. Table I reveals that routing the long two-pin nets with more overflows early can identify the results with less total wirelength in a shorter runtime. Based on this observation, all two-pin nets are sorted at the beginning of each iteration in the NRR stage, based on nets' score (12) in decreasing order.

$$score_k(n) = C_7 \times oe_{k-1}(n) + C_8 \times len_{k-1}(n) \tag{12}$$

where $score_k(n)$ denotes the score of two-pin net n in iteration k, $oe_{k-1}(n)$ and $len_{k-1}(n)$ denote the number of overflowed grid edges passed by n and the length of n in iteration k–1, respectively. Notably, $oe_0(n)$ and $len_0(n)$ depend on the routing outcome of the monotonic routing stage. Additionally, C_7 and C_8 are user-defined constants and set as 30 and 1 in this paper. The bottom row in Table I displays the routing results as the NRR stage adopts (12) to sort two-pin nets, which obtains a shorter wirelength and runtime than other ordering methods. Finally, the postrefinement stage adopt wirelength-decreasing net ordering since most overflows have been removed.

IV. TCS ON MULTICORE PLATFORM

This section proposes a parallel multithreaded global router using TCS on multicore platform to accelerate the NRR stage,

which consumes most computation time of global routing. The concept of TCS is to keep all threads working with almost full load to explore more concurrency than the partitioning-based concurrency strategy. In TCS, a two-pin net routing is defined as a task, and a task queue is maintained to contain all tasks. The task queue is updated at the beginning of each iteration in the NRR stage, and each thread repeatedly acquires a task from the task queue when the thread completes its task. Since all tasks are dynamically committed to available threads, load variation among threads is minimized. To avoid lowering parallelism degree, threads are synchronized only at the end of each iteration. Details of TCS are discussed in Section IV-C.

A. Challenges of TCS

The partitioning-based strategy prevents the simultaneous usage of the same routing resource by more than one thread in a region due to subregion restriction. In contrast, the proposed TCS does not partition the routing region, possibly leading to a situation in which one thread may simultaneously compete with other threads for the same routing resource. Consequently, this situation causes the race condition of routing resources and likely produces additional overflows.

BLMR consists of wave propagation and back-tracing. During propagation, each thread explores every possible move. During back-tracing, the new routing path is identified on the grid graph based on all explored moves, and then the congestion map is updated. If two threads access one routing resource one right after the other during wave propagation, they think the usage of the routing resource will only increase by one. Actually, the final usage of routing resource will increase by two after back-tracing, which may result in overflows. [30, Section IV-A] discusses a collision example (Fig. 6) that is caused by race condition.

The experiments on benchmark newblue1 depict the side effects of collision. Based on the sequential global router (SGR) proposed in the previous section, this paper develops a PGR to simultaneously perform multiple net routings on multiple threads using TCS. SGR can complete the routing of newblue1 in the 83th iteration while PGR still produces several overflows in 200-th iteration. Although one-iteration routing of PGR is faster than that of SGR, PGR takes more iterations to yield worse routing results than SGR.

B. Collision-Aware BLMR

Collision often occurs when several nets are close to each other in a congested region, when the nets are routed simultaneously. Our analysis indicates that more than 41% of the rerouted nets are influenced by collision during the NRR stage. To avoid collision, threads must recognize when a grid edge is also used by other routing paths. Two important phenomena are observed from the analysis of the rip-up and rerouting process of a net. Firstly, a rerouted net often only contains a few overflow grid edges, so the rerouted net only needs a few detours to avoid congested regions. Moreover, the bounded length of a net is incrementally relaxed such that the search region and the identified routing path of a net change only slightly. This implies that the thread may reuse most grid edges of the original path in the new path. Analysis shows that each new routing path reuses about 80% grid edges of the original routing path. Therefore, it is useful to avoid collision by preventing new routing paths from passing through the original routing paths of other currently routed nets.

Algorithm Collision-aware BLMR
Input: grid graph G, net n, bounded-length L
1. Mark_grid_edge(path(n) ,G);
2. Rip_up_from_congestion_map(path(n) ,G);
3. Collision_aware_wave_propagation(n, G, L);
4. *newPath*←Back_tracing(n, G);
5. Route_on_congestion_map(*newPath*, G);
6. Unmark_grid_edge(path(n) ,G);
7. path(n)←*newPath*;
8. **end**

Fig. 4. Algorithm of collision-aware BLMR.

A heuristic collision-prevention approach is developed to enable each thread to notify other threads if certain grid edges have been used by some currently routed nets in the previous iteration. As a result, each thread can estimate the risk of using these grid edges. For more details, each thread marks the grid edges used by the currently routed net in the previous iteration on the congestion map before starting wave propagation of current iteration, and clears their marks after back-tracing ends. The marked grid edges are assigned the extra routing cost. When a thread propagates to a marked grid edge, the extra routing cost of marked grid edges encourages the thread to bypass the marked grid edges for preventing collision. Fig. 4 shows the flow of BLMR adopting this collision-prevention approach. Notably, the race condition may happen when threads simultaneously update the cost of a grid edge (line 2 and line 5). To avoid this situation, most parallel programming languages provide specific instructions to protect variables against multiple writes. This paper updates the congestion map via OpenMP's instruction *atomic*, which can avoid the race condition at the cost of little decrease in routing speed.

As for an example of collision-aware BLMR, please refer to [30, Section IV-B (Fig. 10)].

C. Collision-Aware Rip-Up and Rerouting

Fig. 5 shows the design flow of collision-aware rip-up and rerouting by using TCS and collision-aware BLMR. The task queue *TQ* is implemented by d-queue. *TQ*-related tasks are sorted by the ordering method in (12), and line 5 iteratively extracts the task in the front of *TQ*. Additionally, parallel routing more than one two-pin net of a net is avoided because several demands to update the structure of a net from different threads may incur the race condition. To solve this problem, this paper adopts a lock and unlock scheme. For instance, a two-pin net n is a subnet of net N. Initially, N is unlocked. Notably, N becomes locked when n is launched to commence the routing. After the routing of n ends, N turns back to an unlocked condition. If n is extracted and N is in the *locked* state, n is pushed back to the end of *TQ*. Additionally, an attempt is made to avoid a situation in which multiple threads modify the lock of N, causing the race condition. Therefore, a critical section protects lines 6–7 and line 10 in Fig. 5 to avoid

```
Algorithm Collision-aware Rip-up and Rerouting;
Input: grid graph G
1.  TaskQueue TQ;
2.  while ( G has overflows)
3.    update(TQ) // insert overflowed nets into TQ;
4.    #parallel by each thread
5.      while((2_pin_net n←Extract_a_task(TQ)) ≠ NULL)
6.        if N is locked { Push_back(n, TQ); continue; }
7.        else Lock(N); end if
8.        I_n^k ←Relax_bounded_length(n); //by Eq. (7)
9.        Collision_aware_BLMR(G, n, I_n^k);
10.       Unlock(N);
11.     end while
12.   #end parallel
13.   Update_history_cost(G); // by Eq. (9)
14. end while
15. end
```

Fig. 5. Design flow of collision-aware rip-up and rerouting.

modifying the lock of N from multiple threads, subsequently leading to the race condition.

Although the routing cost formulation of grid edge e in collision-aware rip-up and rerouting resembles (8), p_e is redefined as follows:

$$p_e = 1 + \frac{C_3}{1 + e^{C_4(c(e)-d(e)-vd_i(e))}} \text{ where } vd_i(e) = C_9 \times \sqrt{m_{all}(e) - m_i(e)} \quad (13)$$

where $vd_i(e)$ denotes the virtual demand for thread i; $m_{all}(e)$ represents the number of marks on e; and $m_i(e)$ determines whether e is marked by thread i. Additionally, C_3 and C_4 is defined in (10), and C_9 is a user defined constant, and is set to 0.4. The concept underlying (13) is that e with additional marks should have a higher routing cost since e with more marks implies that it is much likely to be passed by other currently routed nets, subsequently incurring a collision. Correspondingly, the virtual demand (13) enlarges p_e when e is marked. The square root in (13) avoids overestimating the virtual demand since many threads are available. Additionally, thread i must disregard the mark labeled by itself, so $m_i(e)$ is subtracted from $m_{all}(e)$ in (13). At line 1 in Fig. 4, thread i increases both $m_{all}(e)$ and $m_i(e)$ by 1, for each edge e in $path(n)$; at line 6 in Fig. 4, thread i decreases both $m_{all}(e)$ and $m_i(e)$ by 1 with *atomic* instruction to avoid a race condition caused by multiple updates on $m_{all}(e)$.

The proposed parallel router adopting collision-aware rip-up and rerouting in the NRR stage (collision-aware PGR) achieve good overflow reduction. Experiments show that collision-aware PGR not only runs faster than SGR in completing one-iteration routing, but also uses fewer iterations to yield an overflow-free result. Because, the virtual demand encourages routing path more actively escaping the congested regions, but it may slightly increase the total wirelength.

V. 3-D Wirelength Optimization

Current researches address the global routing problem from three aspects, 2-D global routing, 3-D global routing, and layer assignment. Most relevant research examines these aspects individually without discussing the potential of coordinating 2-D routing, 3-D routing, and layer assignment. This paper develops a 3-D wirelength optimization technique that combines the 3-D postrouting (discussed later) and a negotiation-based layer assignment algorithm (NVM) [28] together to iteratively optimize the wirelength (including vias) of a given 3-D routing result. This technique can append to the end of global routers to yield better results. The following paragraph briefly introduces NVM.

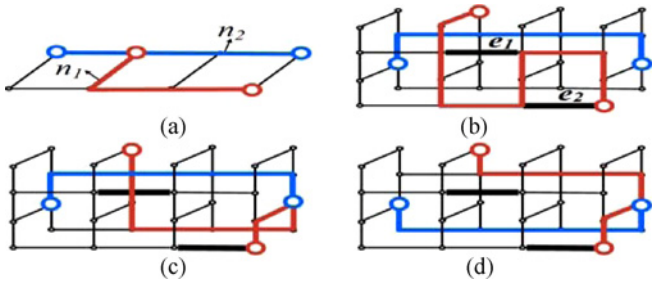

Fig. 6. Example of the quality improvement in 3-D wirelength optimization technique. (a) 2-D routing result obtained by 2-D stage. (b) 3-D result after layer assignment. (c) 3-D result after the 3-D postrouting. (d) 3-D result after NVM.

Most layer assignment algorithm used in global routers [7], [12], [16], [17] determined the assignment order of nets first and, then, assigned each single net sequentially to minimize vias. However, to avoid increasing overflows, the nets in the later assigning order have less available layer resources than those in the early assigning order, so the later assigned nets may not be assigned to their desired layers, resulting in vias increasing. In contrast, NVM adopts a history cost accumulating method to overcome this problem. NVM first identifies a minimal via count solution without considering the overflows for each net and then iteratively re-assigns the nets with overflows until overflows cannot be reduced anymore. During rip-up and reassigning, the history cost of each overflowed grid edge gradually increases to encourage that each net would not be assigned to the overflowed grid edges. In NVM, every net can fairly compete for their desired routing resources.

A. Cooperation of 3-D Postrouting and NVM

Given a 3-D routing result, the proposed 3-D wirelength optimization technique performs the 3-D postrouting and NVM iteratively on this result to further minimize the wirelength without increasing overflows, it iterates until reaching a user-defined iteration limitation. Although capable of altering routing topology, the 3-D postrouting must avoid passing through the exhausted grid edges to prevent increasing overflows, where a grid edge e is regarded as an *exhausted grid edge* if $d(e) \geq c(e)$. In contrast, although capable of negotiating with other nets to acquire the routing resource from the exhausted grid edges in the corresponding layers, each net cannot change the routing topology during NVM. When the 3-D postrouting and NVM are used individually to optimize the routing result, the improved quality is inclined to fall fast into suboptimality. Through iterative topology change and resource relaxation by the negotiation scheme, 3-D wire optimization can refine the routing results well. For example, a 2-D routing result

[Fig. 6(a)] is mapped to a 4-layer 3-D grid graph [Fig. 6(b)]. Since grid edges e_1 and e_2 cross obstacles, net n_1 uses a z-axis detour (via) to bypass obstacles. By assuming the length of a horizontal edge, a vertical edge and a via is one unit and the capacity of each grid edge is one too; the total wirelength of two paths in Fig. 6(b) is 15. Feed the result in Fig. 6(b) into the proposed 3-D wirelength optimization. The 3-D postrouting finds a shorter path for n_1 [Fig. 6(c)] with the reduced total wirelength to be 13. NVM then relaxes the resource at layer 3 used by n_2 for n_1 to further reduce the total wirelength to be 11 [Fig. 6(d)].

B. Inherited History Cost Function

During the 3-D postrouting, every net is ripped-up and re-routed. The 3-D postrouting adopts an *inherited history cost* function to guide the routing of each net, which can improve total wirelength. In 2-D routing and NVM, a grid edge with a high history cost implies that this grid edge frequently overflows and many nets desire to pass this grid edge, so the routing resource of this grid edge is critical. The history cost information acquired by 2-D routing and NVM is delivered to the 3-D postrouting in order to broaden the view of the routed nets in terms of knowing which grid edge is critical. The inherited history cost formulation is defined as follows:

$$\text{cost}\,(e_{i,z}) = \begin{cases} \text{A very large constant} & \text{if } d(e_{i,z}) \geq c(e_{i,z}) \\ 1 + C_{10} \times \left(\frac{d(e_{i,z})}{1+c(e_{i,z})}\right)^2 & \\ \quad + C_{11} \times hisC_I(e_{i,z}) & \text{otherwise} \end{cases} \tag{14}$$

where $hisC_I(e_{i,z})$ denotes the inherited history cost of the 3-D grid edge $e_{i,z}$, and C_{10} and C_{11} are user defined constants and set as 0.1 and 0.05 in this paper, respectively. The $hisC_I(e_{i,z})$ is formulated as follows:

$$hisC_I(e_{i,z}) = NORM(h_{2-D}(e_i)) + NORM(h_{NVM}(e_{i,z})) \tag{15}$$

where grid edge e_i in the 2-D graph is projected from $e_{i,z}$ in 3-D graph; $h_{NVM}(e_{i,z})$ denotes the history cost of $e_{i,z}$, as computed in the previous NVM of the 3-D postrouting, and $h_{2-D}(e_i)$ is the history cost from 2-D routing. Since the 3-D wirelength optimization is designed to follow the execution of a 2-D global router, $h_{2-D}(e_i)$ is the final history cost of e_i, such as that defined in (9) as NRR of 2-D routing ends. Because NVM follows the 3-D postrouting, the first round of the 3-D postrouting have no $h_{NVM}(e_{i,z})$ information. Thus, $NORM(h_{NVM}(e_{i,z}))$ is ignored in the first round of the 3-D postrouting. Function $NORM$ normalizes $h_{2-D}(e_i)$ and $h_{NVM}(e_{i,z})$ between zero to one. Function $NORM(h_{2-D}(e_i))$ derives the normalized values as follows. The grid edges in the 2-D grid graph with nonzero history costs are sorted in a nondecreasing order of their history costs. By assuming that the length of the ordering sequence is n_{2-D}, $h_{2-D}(e_i)$ is normalized to i/n_{2-D} as e_i is the i-th element in this sorting sequence. Similarly, $NORM(h_{NVM}(e_{i,z}))$ is computed in the same manner. Fig. 7 shows the design flow of the proposed 3-D wirelength optimization method.

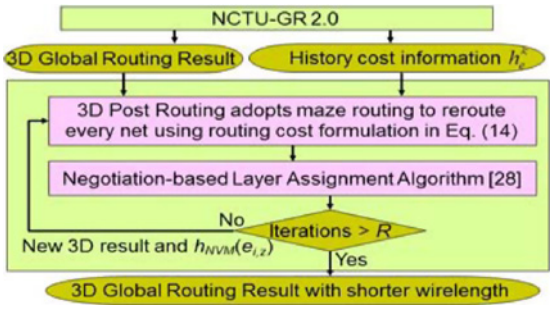

Fig. 7. Design flow of 3-D wirelength optimization.

VI. Experimental Results

The proposed algorithms were implemented in C/C++ language on an 8-core 3.0 GHz Intel Xeon-based server with 32 GB memory. ISPD [5], [6] benchmark circuits were used in our experiments. We classify the benchmarks into two types, i.e., overflow-free cases and hard-to-route cases. All state-of-the-art global routers cannot identify an overflow-free routing result for each hard-to-route case. As for overflow-free cases, most state-of-the-art routers can identify an overflow-free routing result for each one. To overflow-free cases, the routers in the following experiments perform until an overflow-free outcome is achieved; to hard-to-route cases, the routers stop when overflows are not improved in five successive iterations. Because the overflow improvement on hard-to-route cases is insignificant in Sections VI-A–C, this paper shows only the experimental results on overflow-free cases in Sections VI-A–C. Subsections D-E show the experimental results of all cases. Notably, the routers in Sections VI-A–F do not invoke the 3-D wirelength optimization, the 3-D wirelength optimization is used in Section VI-G to refine the routing results.

A. Comparing Maze Routings and BLMR

To compare traditional maze routings with and without bounding box and BLMR, we implement three global routers with different maze routing approaches in the NRR stage. MR-GR, MRB-GR, and H-BLMR-GR denote three different global routers, where MR-GR employs maze routing without using bounding box; MRB-GR employs bounding box to limit the search region, and H-BLMR-GR adopts the proposed heuristic-BLMR and bounded-length relaxation scheme in the NRR stage. The initial bounding box used by MRB-GR extends by 10 units of grid edges four boundaries of the minimum rectangle enclosing all terminals of the routed net. If an overflow-free path cannot be obtained within the bounding box, each boundary of the bounding box is extended by 10 units of grid edges again in the next iteration. This bounding box expansion scheme is also used in [15]. Notably these routers adopt the same routing cost function and the proposed RSMT-aware routing scheme is not used by these routers. Table II shows the routing results of these routers, where WL and CPU are total wirelength and CPU time, respectively. Table II indicates that MRB-GR is faster than MR-GR, but MRB-GR produces more wirelength than MR-GR because MRB-GR may detour often to avoid congested regions. H-BLMR-GR produces less wirelength than MR-GR and

TABLE II
ROUTING RESULTS OF GLOBAL ROUTING WITH BLMR AND MAZE ROUTING WITH AND WITHOUT BOUNDING BOX

ISPD'08	MR-GR		MRB-GR		H-BLMR-GR	
benchmark	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)
adaptec1	53.86	5.90	54.29	3.85	53.55	2.65
adaptec2	52.14	3.36	52.49	0.96	51.97	0.79
adaptec3	131.16	4.47	131.85	3.72	129.88	3.50
adaptec4	120.88	1.32	120.98	1.25	120.76	1.20
adaptec5	155.76	14.66	157.63	9.20	155.52	9.16
newblue1	x	x	x	x	46.26	2.80
newblue2	74.74	0.70	74.77	0.67	74.63	0.63
newblue5	231.33	53.94	233.45	18.73	230.45	7.22
newblue6	178.71	17.66	179.69	6.23	177.23	5.46
bigblue1	57.17	10.69	59.02	6.13	57.61	8.49
bigblue2	90.05	48.40	89.90	29.01	89.42	7.73
bigblue3	129.95	15.79	130.15	2.44	129.68	2.19
Ratio	1	1	1.007	0.575	0.997	0.494

MRB-GR because it has less detours than MR-GR and MRB-GR. The BLC of H-BLMR-GR restricts the searching region such that H-BLMR-GR is faster than MR-GR and MRB-GR. Furthermore, the proposed bounded-length relaxation scheme offers more efficient routing resource utilization than the other two routers. As a result, H-BLMR-GR can eliminate all overflows of newblue1 but MR-GR and MRB-GR cannot.

B. Effectiveness of RSMT-Aware Routing

Table III shows the effectiveness of RSMT-aware routing scheme, in which the second, third and fourth (fifth, sixth, and seventh) columns show the wirelength, routing iterations, and runtime of H-BLMR-GR without (with) RSMT-aware routing scheme, respectively. H-BLMR-GR with RSMT-aware routing scheme reduces 0.825% wirelength than that without the scheme. Although RSMT-aware routing scheme spends additional effort to identify RSMT, less wirelength usage makes H-BLMR-GR demand less iterations and then converge faster. The iteration number and runtime of H-BLMR-GR with RSMT-aware routing scheme are reduced by 18.32% and 19.58%, respectively, than that without the scheme.

C. Comparing Optimal-BLMR and Heuristic-BLMR

Table IV compares the routing results of optimal-BLMR, heuristic-BLMR, and CSP algorithm in [26]. The BLMR problem is a restricted version of CSP, so the algorithm of [26] can be adopted to solve BLMR problem. The algorithm of [26] can identify the optimal solution of BLMR problem via dynamic-programming technique. The time complexity of the CSP algorithm in [26] is O($|E|L$) where $|E|$ is the number of grid edges in the routing graph and L is BLC. O-BLMR-GR and CSP-GR adopts the optimal-BLMR and the CSP algorithm [26] in the NRR stage, respectively. Note that the routers in Table IV all employ RSMT-aware routing scheme. Table IV shows that H-BLMR-GR runs averagely 269.21 times faster than O-BLMR-GR, and only increases 0.1% total wirelength. The experiments indicate that heuristic-BLMR can take much less runtime to yield similar routing results with optimal-BLMR. On the other hand, the routing results of CSP-GR and

TABLE III
COMPARISON OF THE RESULTS OF H-BLMR-GR WITH AND WITHOUT RSMT-AWARE ROUTING SCHEME

'ISPD'08	H-BLMR-GR(w/o RSMT)			H-BLMR-GR(w RSMT)		
benchmark	WL	Rounds	CPU (m)	WL	Rounds	CPU (m)
adaptec1	53.55	11	2.65	53.04	10	2.52
adaptec2	51.97	12	0.79	51.51	10	0.65
adaptec3	129.88	9	3.50	129.24	9	3.40
adaptec4	120.76	7	1.20	120.53	8	1.23
adaptec5	155.52	16	9.16	154.01	11	6.03
newblue1	46.26	83	2.80	45.76	67	2.37
newblue2	74.63	5	0.63	74.51	5	0.64
newblue5	230.45	21	7.22	228.68	15	5.26
newblue6	177.23	14	5.46	175.49	11	4.44
bigblue1	57.61	18	8.49	56.29	9	4.09
bigblue2	89.42	86	7.73	88.66	48	4.21
bigblue3	129.68	22	2.19	129.35	19	1.91
Improve				0.825%	18.3%	19.58%

TABLE IV
COMPARISON OF THE ROUTING RESULTS OF GLOBAL ROUTERS WITH HEURISTIC-BLMR, OPTIMAL-BLMR, AND [26]

ISPD'08	H-BLMR-GR (w RSMT)		O-BLMR-GR (w RSMT)		CSP-GR (w RSMT)	
benchmark	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)
adaptec1	53.04	2.52	52.93	615.21	52.93	3586.67
adaptec2	51.51	0.65	51.49	18.75	51.49	61.31
adaptec3	129.24	3.40	129.09	777.14	129.09	7406.14
adaptec4	120.53	1.23	120.52	30.48	120.52	159.11
adaptec5	154.01	6.03	153.66	1607.43	153.66	10126.81
newblue1	45.76	2.37	45.68	1044.53	45.68	7551.95
newblue2	74.51	0.64	74.50	5.68	74.50	33.63
newblue5	228.68	5.26	228.55	1766.12	228.55	12009.62
newblue6	175.49	4.44	175.44	664.18	175.44	5074.34
bigblue1	56.29	4.09	56.04	2537.78	56.04	11952.94
bigblue2	88.66	4.21	88.56	2576.10	88.56	13447.24
bigblue3	129.35	1.91	129.29	516.24	129.29	4315.77
Ratio	1	1	0.999	269.21	0.999	1705.67

O-BLMR-GR are same but the runtime of CSP-GR is much larger than that of O-BLMR-GR since CSP algorithm [26] does not take three properties of global routing into account, as described in Section III.

D. Routing Result Comparison of Sequential Routers

The proposed SGR adopting heuristic-BLMR controls the wirelength increasing during the NRR stage, and SGR employs RSMT-aware routing scheme to further reduce wirelength and runtime. Tables V and VI compare the routing results of SGR with four state-of-the-art routers. NTHU-Route2.0 [12], FastRoute4.1 [16], and NCTU-GR [17] are 2-D router with layer assignment. MGR is a multilevel 3-D router that runs much faster than traditional 3-D routers [7], [21], [22]. The results of MGR are quoted from [20] because the binary of MGR is unavailable, while the other routers are performed on the same platform. Notably, MGR performs on a 2.6 GHz Intel CPU with 16 G memory, the runtime of MGR is normalized by the clock rate ratio 1.154. Various control parameters in routers affect the routing quality and performance. SGR and

TABLE V
COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND THE OTHER ROUTERS ON OVERFLOW-FREE CASES

Benchmark	SGR		SGR_B		NTHU-Route2.0[12]		FastRoute 4.1 [16]		NCTU-GR [17]		MGR [20]	
	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)	WL	CPU (m)
adaptec1	53.04	2.52	52.35	2.30	53.49	4.86	53.73	3.31	53.50	3.90	52.82	4.39
adaptec2	51.51	0.65	51.30	0.64	52.31	1.42	52.17	0.95	51.69	1.45	51.46	1.04
adaptec3	129.24	3.40	128.34	2.96	131.11	6.16	130.82	3.69	130.35	4.88	128.92	4.83
adaptec4	120.53	1.23	120.17	1.18	121.73	2.08	121.24	1.25	120.67	2.28	119.96	1.41
adaptec5	154.01	6.03	151.85	4.97	155.55	11.95	155.81	6.70	154.70	9.07	153.23	7.95
newblue1	45.76	2.37	45.62	1.93	46.53	4.07	46.33	12.01	45.99	3.63	45.58	4.51
newblue2	74.51	0.64	74.51	0.63	75.85	1.17	75.12	0.85	74.88	0.90	74.46	0.80
newblue5	228.68	5.26	225.94	4.62	231.73	10.88	230.94	9.82	230.31	15.03	228.00	6.54
newblue6	175.49	4.44	171.10	4.02	177.01	10.34	177.87	8.78	176.87	9.67	174.86	7.04
bigblue1	56.29	4.09	55.33	3.44	56.35	6.93	56.64	4.22	56.56	6.35	55.82	5.04
bigblue2	88.66	4.21	86.71	3.45	90.59	6.47	91.18	12.12	89.40	11.18	88.92	6.00
bigblue3	129.35	1.91	127.67	1.78	130.76	3.91	130.04	2.06	129.66	4.38	128.75	2.89
Ratio	1	1	0.99	0.89	1.01	1.90	1.01	1.77	1.01	1.92	0.997	1.45

TABLE VI
COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND THE OTHER ROUTERS ON HARD-TO-ROUTE CASES

	SGR				SGR_B				NTHU-Route 2.0 [12]			
	MO	TO	WL	CPU (m)	MO	TO	WL	CPU (m)	MO	TO	WL	CPU (m)
newblue3	194	31710	105.36	143.34	194	31526	106.80	63.34	204	31454	106.49	64.97
newblue4	2	144	127.27	17.33	2	132	129.27	17.48	4	138	130.46	52.01
newblue7	2	58	342.90	85.67	2	54	341.90	74.53	2	62	353.35	50.28
bigblue4	2	194	225.00	60.23	2	132	227.10	63.55	2	162	231.04	52.63
Ratio	1	1	1	1	1.00	0.88	1.01	0.84	1.26	0.96	1.02	1.23
	FastRoute 4.1 [16]				NCTU-GR [17]				MGR [20]			
	MO	TO	WL	CPU (m)	MO	TO	WL	CPU (m)	MO	TO	WL	CPU (m)
newblue3	736	31276	108.40	15.99	198	31808	104.28	131.43	x	31026	107.22	19.99
newblue4	2	136	130.46	65.23	2	134	126.79	40.92	x	136	128.54	15.64
newblue7	4	54	353.38	868.74	2	114	338.63	71.52	x	56	349.02	110.12
bigblue4	2	130	230.24	93.25	2	164	223.99	65.37	x	134	225.73	21.31
Ratio	1.95	0.88	1.03	3.89	1.01	1.19	0.99	1.30	x	0.90	1.01	0.67

FastRoute4.1 adopt a single set of control parameters to solve all benchmarks while NCTU-GR, NTHU-Route2.0 use different control parameters to identify their best routing result for each benchmark. MGR automatically adapts parameters based on the characteristics of each benchmark. For comparison, we also adopt different sets of control parameters to yield the best routing result for each benchmark, listed at column SGR_B of Tables V and VI. Table V compares each router by overflow-free cases, in which the wirelength and runtime are the primary items for comparison because all routers produce overflow-free routing results. In table5, SGR identifies 1.1%, 1.1% and 0.6% less wirelength and averagely runs 1.90×, 1.77×, and 1.92× faster than NTHU-Route2.0, FastRoute4.1, and NCTU-GR, respectively. Compared to MGR, SGR identify 0.3% longer wirelength.

For hard-to-route cases, the routers [12], [13], [16] focus on minimizing TOs without minimizing the MO, since TO provides a more global perspective on congestion information than the MO. However, the inability to address the MO may lead to very congested hot spots, possibly becoming unroutable for detailed routing. Therefore, this paper considers both TO and the MO. SGR regards MO as the first minimization objective, and TO as the second objective. Table VI reveals that SGR achieves good performance for MO, wirelength, and runtime, but the TO still has room for improvement. As for the best routing result for each benchmark (SGR_B), the proposed router performs very well in the TO and runtime. Notably, the MO information of MGR is unavailable, so we do not list MO of MGR in Table VI.

E. Proposed Collision-Aware PGR

Table VII compares the runtime of the NRR stage (NRRT) between SGR and collision-aware PGR for overflow-free cases. Because the race condition may cause nondeterministic routing results of collision-aware PGR, collision-aware PGR is performed 10 times to display the worst, average, and best results. Table VII reveals that collision-aware PGR using 4/8 cores offers an average 2.54×/4.11× and 2.71×/4.32× speedup for the worst case and average case, respectively. However, collision-aware PGR may produce longer wirelength because the virtual demand in (13) makes the nets have additional detours to avoid collisions. Since increasing the used cores also increases collisions, collision-aware PGR using eight cores produces more detours to avoid collisions than that using four cores. Also, the increased collisions may jeopardize the scalability of collision-aware PGR. Fig. 8 displays the scalability of collision-aware PGR for some benchmarks, in which the x-axis and y-axis denote the number of cores and

TABLE VII
COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND PARALLEL ROUTER ON OVERFLOW-FREE CASES

	SGR	Collision-aware PGR (4-core)						Collision-aware PGR (8-core)					
	NRRT (m)	WL inc % ($((W_{p4}-W_s)/W_s)$)			NRRT imprv. (T_s/T_{p4})			WL inc. % ($((W_{p8}-W_s)/W_s)$)			NRRT imprv. (T_s/T_{p8})		
		Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best
adaptec1	1.78	0.45	0.43	0.40	3.36	3.48	3.54	0.76	0.72	0.69	5.60	5.75	5.81
adaptec2	0.20	0.06	0.05	0.04	2.06	2.32	2.57	0.11	0.08	0.07	2.23	2.54	2.68
adaptec3	1.77	0.06	0.05	0.04	3.19	3.28	3.32	0.17	0.14	0.13	5.14	5.32	5.42
adaptec4	0.10	0.01	0.01	0.01	1.92	2.06	2.21	0.03	0.02	0.01	1.69	1.92	2.14
adaptec5	4.06	0.32	0.31	0.29	3.39	3.54	3.60	0.54	0.51	0.46	6.30	6.42	6.52
newblue1	2.00	−0.03	−0.04	−0.05	2.32	2.69	3.03	0.05	0.03	0.01	2.69	3.20	3.50
newblue2	0.04	0.01	0.01	0.00	1.42	1.55	1.72	0.03	0.02	0.02	1.57	1.65	1.70
newblue5	2.92	0.07	0.06	0.06	2.78	2.88	3.06	0.20	0.17	0.13	4.98	5.15	5.30
newblue6	2.52	0.18	0.14	0.06	2.80	2.90	2.99	0.36	0.31	0.25	5.13	5.32	5.44
bigblue1	3.25	0.83	0.78	0.75	2.93	3.10	3.29	1.17	1.12	0.99	5.80	5.92	6.09
bigblue2	3.54	0.18	0.14	0.12	2.02	2.34	2.95	0.22	0.21	0.19	3.51	3.80	4.28
bigblue3	0.61	0.07	0.06	0.05	2.28	2.39	2.56	0.14	0.11	0.09	4.69	4.85	5.00
Average		0.18	0.17	0.15	2.54	2.71	2.90	0.32	0.29	0.25	4.11	4.32	4.49

TABLE VIII
COMPARISON BETWEEN THE PROPOSED SEQUENTIAL ROUTER AND PARALLEL ROUTER ON HARD-TO-ROUTE CASES

	SGR	Collision-aware PGR (4-core)									Collision-aware PGR (8-core)								
	NRRT	TO			WL inc %			NRRT imprv.			TO			WL inc. %			NRRT imprv.		
	(m)	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best
newblue3	142.70	31810	31776	31710	0.01	−0.03	−0.06	2.13	2.25	2.33	32044	31808	31702	0.04	0.00	−0.02	4.28	4.56	4.89
newblue4	15.76	150	144	138	0.04	0.01	−0.03	2.50	2.58	2.73	158	148	140	0.09	0.05	0.03	4.91	5.02	5.09
newblue7	84.19	74	66	60	−0.10	−0.11	−0.13	3.10	3.20	3.35	82	72	68	0.01	−0.03	−0.04	5.74	5.97	6.19
bigblue4	59.48	200	196	190	−0.02	−0.03	−0.04	4.12	4.47	4.79	212	204	196	0.01	−0.01	−0.02	6.55	7.08	7.23
Average					−0.02	−0.04	−0.07	2.96	3.12	3.30				0.04	0.00	−0.01	5.37	5.66	5.85

In Tables VII and VIII, the increasing rate and improving rate are calculated as compared to the results of SGR.

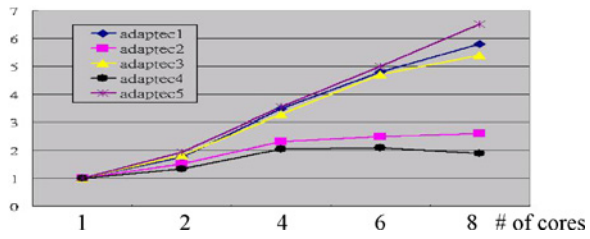

Fig. 8. Scalability of collision-aware PGR.

the runtime speedup, respectively. Collision-aware PGR yields good acceleration for the large benchmark adaptec1, adaptec3, and adaptec5, but limited speedup for adaptec2 and adaptec4, the smallest designs in those benchmarks, since the increased collisions and the overhead of thread scheduling decreases the benefit of parallelism in the small designs.

Tables III and VII reveal an interesting phenomenon. The second last column titled "rounds" in Table III lists the iteration number of SGR for each benchmark. The benchmarks that demand additional iterations by SGR to identify an overflow-free result may have some hard-to-route regions; most of the rerouted nets are in these regions. When the used cores increase, the collisions in these regions also increase, subsequently degrading the speedup of collision-aware PGR. Thus, collision-aware PGR using eight cores has less speedup on these benchmarks.

Table VIII shows the routing results of collision-aware PGR for hard-to-route cases. The MO of collision-aware PGR is not listed since it is the same as SGR in Table VI. Collision-aware PGR produces more TO than SGR due to collisions, but it runs 3.12× and 5.66× faster than SGR in the average case on the 4-core and 8-core platforms, respectively. Notably, collision-aware PGR using four cores achieves a super linear speedup (4.47×) in bigblue4. With parallel computation, the total size of used caches by working cores increases as the number of used cores increases. This in turn leads to more cache accesses and diminishes time-consuming memory accesses, which provides the potential for a super linear speedup.

F. Discussion on Collision-Aware Task Ordering

The potential collision between two tasks with an overlapping searching region can be eliminated by avoiding the simultaneous routing of these two tasks. With this consideration, collision minimization can be achieved by properly arranging the execution order of tasks. However, several works [10], [12], [13], [16], and [17] indicate that the routing order impacts the routing quality and runtime as well, implying that collision minimization based on task ordering may degrade routing quality and enlarge runtime. To elucidate how task ordering for collision minimization affects routing, this paper designs a collision-aware task ordering (CTO) method as follows.

1) Identify the searching region of each task.
2) Build a relation graph of all tasks with a node to represent a task and an edge to connect two nodes whose related tasks have an overlapping searching region.
3) Assign each node a color by using the vertex coloring algorithm [29] on the relation graph. Notably, the colors of adjacent nodes differ from each other.

TABLE IX
COMPARISON BETWEEN THE PROPOSED PARALLEL ROUTER WITH DIFFERENT TASK ORDERING METHODS

ISPD'08 benchmark	$CPGR_1$		$CPGR_2$		$CPGR_3$	
	WL	NRRT (m)	WL	NRRT (m)	WL	NRRT (m)
adaptec1	53.27	1.78	55.35	1.94	55.24	2.53
adaptec2	51.53	0.20	52.61	0.21	52.51	0.39
adaptec3	129.30	1.77	135.89	1.94	135.77	2.80
adaptec4	120.54	0.10	124.76	0.11	124.64	0.23
adaptec5	154.48	4.06	163.59	4.18	162.98	5.48
newblue1	45.74	2.00	46.70	2.42	46.65	3.80
newblue2	74.51	0.04	76.60	0.04	76.45	0.08
newblue5	228.82	2.92	232.02	3.14	231.79	5.05
newblue6	175.73	2.52	182.58	2.70	182.23	4.16
bigblue1	56.73	3.25	58.77	3.38	58.66	5.59
bigblue2	88.79	3.54	92.34	4.07	92.16	5.42
bigblue3	129.43	0.61	132.67	0.66	132.15	0.88
Ratio	1	1	1.034	1.084	1.032	1.723

TABLE X
WIRELENGTH IMPROVEMENT AND RUNTIME OF THE PROPOSED 3-D WIRELENGTH OPTIMIZATION TECHNIQUE

ISPD'08	3-D WL Opti.		ISPD'08	3-D WL Opti.	
benchmark	WL (%)	CPU (m)	benchmark	WL (%)	CPU (m)
adaptec1	2.66	8.82	newblue4	2.62	14.70
adaptec2	3.03	7.84	newblue5	2.96	28.42
adaptec3	2.47	36.26	newblue6	2.23	10.78
adaptec4	2.98	20.58	newblue7	2.89	41.16
adaptec5	2.44	18.62	bigblue1	2.31	6.86
newblue1	2.99	7.84	bigblue2	2.59	7.84
newblue2	3.49	9.80	bigblue3	3.01	21.56
newblue3	3.04	577.22	bigblue4	2.89	30.38

4) Cluster the nodes with the same color as a group, in which each node has no overlapping searching region with the other nodes, allowing for all nodes in a group to be performed simultaneously without collision.
5) Calculate an ordering score by (11) for each task and, then, calculate the score of each group by averaging the scores of all tasks in the group.
6) Sort all groups in decreasing order of the group scores. Next, sort all tasks in each group in decreasing order of the task scores.

Table IX compares the results of the proposed parallel router in which different task ordering schemes are adopted at the beginning of each iteration during the NRR stage. $CPGR_1$ refers to the original collision-aware PGR, which applies the ordering method in (11) directly. $CPGR_2$ refers to collision-aware PGR by applying the CTO method to sort all tasks. The threads of $CPGR_2$ first acquire tasks from the first group of the sorted group list. For a situation in which the first nonempty group in the sorted list turns out to be empty, the threads of $CPGR_2$ acquire the tasks from the next group, i.e. the new first nonempty group in the sorted group list. In $CPGR_2$, collisions occur only when the tasks in different groups are routed simultaneously, explaining why the probability of a collision for $CPGR_2$ is less than that of $CPGR_1$. $CPGR_3$ also applies the CTO method with an additional constraint to acquire a task from a new nonempty group. This constraint delays the task dispatching of a new nonempty group in the sorted group list until all previously dispatched tasks are completed. Via the task dispatching constraint, $CPGR_3$ never produces collisions. Table IX reveals that $CPGR_1$ outperforms $CPGR_2$ and $CPGR_3$ in wirelength and runtime, indicating that the CTO method favors collision avoidance, yet degrades the quality and runtime. Additionally, $CPGR_2$ and $CPGR_3$ require additional runtime for the computational effort of CTO. Finally, the additional constraint to initiate the task dispatching of a new nonempty group in the sorted group list lowers the parallelism degree explaining why $CPGR_3$ runs slower than $CPGR_1$ and $CPGR_2$.

G. Effectiveness of the 3-D Wirelength Optimization

Table X shows the wirelength improvement (WL%) and the runtime of using the proposed 3-D wirelength optimization to further refine the routing results produced by SGR (Tables V and VI). The overflow information is not listed here because it is same as the numbers in Table V and VI. The 3-D wirelength optimization, which iteratively performs the 3-D postrouting and NVM twice, can achieve an improved wirelength of 2.79% on average. Compared to the runtime of SGR in Table V, runtime of the 3-D wirelength optimization is enormous due to large 3-D searching space. However, compared to the pure 3-D global routers [7], [18], [19], runtime of the 3-D wirelength optimization is negligible, and SGR with the proposed 3-D wirelength optimization can achieve a routing quality similar to the pure 3-D router [18]. Despite the ability of TCS to accelerate the 3-D postrouting, collisions may incur overflows. 3-D wirelength optimization attempts to minimize wirelength and vias without increasing overflows, explaining why this paper does not parallelize the 3-D postrouting.

VII. CONCLUSION

This paper presented a novel SGR, characterized by a heuristic-BLMR algorithm, RSMT-aware routing scheme, and dynamically adjusted history cost function. Excellent performance of NCTU-GR 2.0 on ISPD benchmarks with a single set of parameters is owing to systematic and effective wirelength control, flexible tree structure change, and valid update of edge congestion status with overflow frequency. Additionally, this paper parallelizes the proposed global router on a multicore platform. The TCS avoids the side effect of partitioning-based strategy, yet tends to underestimate the value of $d(e)$, further exacerbating the overflowing problem. The proposed collision-aware rip-up and rerouting algorithm resolves this problem. Finally, this paper presented a 3-D wirelength optimization technique that performs 3-D postrouting and layer assignment by turns. Through iterative topology change and resource relaxation by the negotiation scheme, 3-D wire optimization can refine the 3-D routing results well. Furthermore, the proposed method can optimize the wirelength of 3-D routing results yielded by any global router.

REFERENCES

[1] K.-R. Dai, C.-H. Liu, and Y.-L. Li, "GRPlacer: Improving routability and wire-length of global routing with circuit replacement," in *Proc. ICCAD*, 2009, pp. 351–356.

[2] M.-C. Kim, J. Hu, D.-J. Lee, and I. L. Markov, "A SimPLR method for routability-driven placement," in *Proc. DAC*, 2011, pp. 80–84.
[3] M. Pan and C. Chu, "IPR: An integrated placement and routing algorithm," in *Proc. DAC*, 2007, pp. 67–73.
[4] J. Roy, N. Viswanathan, G.-J. Nam, C. J. Alpert, and I. L. Markov, "CRISP: Congestion reduction by iterated spreading during placement," in *Proc. ICCAD*, 2009, pp. 357–362.
[5] G.-J. Nam. (2007). *ISPD 2007 Contest* [Online]. Available: http://archive.sigda.org/ispd2007/contest.html
[6] Cliff Sze. (2008). *ISPD 2008 Contest* [Online]. Available: http://archive.sigda.org/ispd2008/contests/ispd08rc.html
[7] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE TCAD*, vol. 27, no. 6, pp. 1066–1077, Jun. 2008.
[8] J. Hu, J. Roy, and I. Markov, "Completing high-quality global routes," in *Proc. ISPD*, 2010, pp. 35–41.
[9] M. Moffitt, "MAIZEROUTER: Engineering an effective global router," *IEEE TCAD*, vol. 27, no. 11, pp. 2017–2026, Mar. 2008.
[10] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router," in *Proc. ICCAD*, 2007, pp. 503–508.
[11] M. M. Ozdal and M. D. F. Wong, "ARCHER: A history-driven global routing algorithm," in *Proc. ICCAD*, 2007, pp. 488–495.
[12] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, "NTHU-Route 2.0: A roubust global router for modern design," *IEEE TCAD*, vol. 29, no. 12, pp. 1931–1944, Dec. 2010.
[13] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang. "High-performance global routing with fast overflow reduction," in *Proc. ASP-DAC*, 2009, pp. 582–587.
[14] M. Pan and C. Chu, "FastRoute : A step to integrate global routing into placement," in *Proc. ICCAD*, 2006, pp. 464–471.
[15] M. Pan and C. Chu, "FastRoute 2.0: A high-quality and efficient global router," in *Proc. ASP-DAC*, 2007, pp. 250–255.
[16] Y. Xu, Y. Zhang, and C. Chu, "FastRoute 4.0: Global router with efficient via minimization," in *Proc. ASP-DAC*, 2009, pp. 576–581.
[17] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "NCTU-GR: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing," *IEEE TVLSI*, vol. 20, no. 3, pp. 459–472, Mar. 2012.
[18] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "GRIP: Scalable 3-D global routing using integer programming," in *Proc. DAC*, 2009, pp. 320–325.
[19] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "A parallel integer programming approach to global routing," in *Proc. DAC*, 2010, pp. 194–199.
[20] Y. Xu and C. Chu, "MGR: Multi-level global router," in *Proc. ICCAD*, 2011, pp. 250–255.
[21] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, 1995, pp. 111–117.
[22] D. Muller, K. Radke, and J. Vygen, "Faster min–max resource sharing in theory and practice," in *Mathematical Programming Computation*. Berlin: Springer, 2011.
[23] S. Sapatnekar, E. Haritan, K. Keutzer, A. Devgan, D. A. Kirkpatrick, S. Meier, D. Pryor, and T. Spyrou, "Reinventing EDA with manycore processors," in *Proc. DAC*, 2008, pp. 126–127.
[24] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE TCAD*, vol. 27, no. 1, pp. 70–83, Jan. 2008.
[25] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* New York: W. H. Freeman and Co., 1979.
[26] H. C. Joksch, "The shortest route problem with constraints," *J. Math. Anal. Appl.*, vol. 14, pp. 191–197, May 1966.
[27] S. Chen, M. Song, and S. Sahni, "Two techniques for fast computation of constrained shortest paths," *IEEE/ACM Trans. Network.*, vol. 16, no. 1, pp. 105–115, Feb. 2008.
[28] W.-H. Liu and Y.-L. Li, "Negotiation-based layer assignment for via count and via overflow minimization," in *Proc. ASP-DAC*, 2011, pp. 539–544.
[29] B. Manvel, "Coloring large graphs," in *Proc. Southeastern Conf. Graph Theory, Combinatorics Comput. Sci.*, 1981, pp. 197–204.
[30] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Multi-threaded collision-aware global routing with bounded-length maze routing," in *Proc. DAC*, 2010, pp. 200–205.

Wen-Hao Liu received the B.S. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 2008, where he is currently pursuing the Ph.D. degree in computer science.

Mr. Liu was a recipient of the first-place prize in the clock tree contest of the ACM/IEEE International Symposium on Physical Design (ISPD) 2009. He has developed a robust global router NCTU-GR 2.0 which is selected as the evaluation tool for ISPD11, DAC12, and ICCAD12 placement contests.

Wei-Chun Kao received the B.S. degree in computer science from Taipei Municipal University of Education, Taipei City, Taiwan, in 2007, and the M.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2009.

He is currently working as a Senior Engineer in mobile system division of the MStar Semiconductor, Inc., Hsinchu Hsien, Taiwan. His current research interests include global routing and parallel computing.

Yih-Lang Li received the B.S. degree in nuclear engineering and the M.S. and Ph.D. degrees in computer science, majoring in developing and implementing a highly parallel cellular automata machine for fault simulation, all from the National Tsing Hua University, Hsinchu, Taiwan.

He has been an Associate Professor with the Department of Computer Science, National Chiao Tung University (NCTU) since February 2003. Prior to joining the faculty of NCTU, from 1995 to 1996 and from 1998 to 2003, he was a Software Engineer and an Associate Manager with Springsoft Corporation, Hsinchu, Taiwan, where he first completed the development of design rule checking tool for the custom-based layout design, and then established and leaded a routing team for developing a block-level shape-based router for custom-based layout design. His current research interests include physical synthesis, parallel architecture, and VLSI testing.

Mr. Li joined the technical committee of the first CAD contest in Taiwan and had served as the committee member for 10 years. He served as the contest chair of the first CAD contest at ICCAD in 2012.

Kai-Yuan Chao received the B.S. degree in nuclear engineering from National Tsing Hua University, Hsinchu, Taiwan, in 1986, the M.S. degree in medical engineering from National Yang-Ming Medical College, Taipei, Taiwan, in 1988, and the M.S.E. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, Austin, TX, in 1992 and 1995, respectively.

He is currently a Principal Engineer leading the 14 nm design and manufacture interface for microserver system-on-chip CPU and server memory buffer products in the Intel Architecture Group, Intel Corporation, Hillsboro, OR. He has coauthored more than 60 technical papers and two book chapters in the areas of VLSI/CAD, packaging, and radiology. His current research interests include architecture/design convergence, deep submicron design/manufacture/packaging interface, and design collaboration/management system.

Dr. Chao served in 2006–2008 ACM/IEEE ISPD and SASIMI 2009-2010 technical program committees.