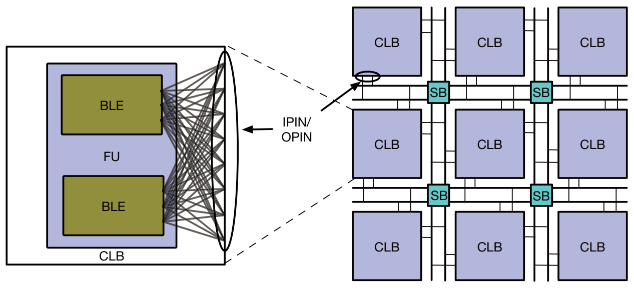
# A Robust FPGA Router with Concurrent Intra-CLB Rerouting

Jiarui Wang[1,2], Jing Mai[1,2], Zhixiong Di[3], Yibo Lin[2,4*]
[1]School of Computer Science, Peking University, Beijing, China
[2]School of Integrated Circuits, Peking University, Bejing, China
[3]School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China
[4]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China
{jiaruiwang,jingmai}@pku.edu.cn,dizhixiong2@126.com,yibolin@pku.edu.cn

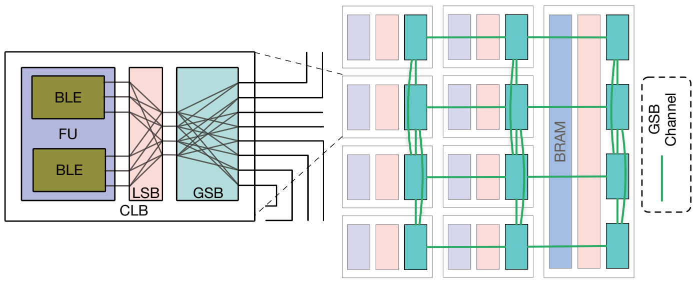
## ABSTRACT

Routing is the most time-consuming step in the FPGA design flow with increasingly complicated FPGA architectures and design scales. The growing complexity of connections between logic pins inside CLBs of FPGAs challenges the efficiency and quality of FPGA routers. Existing negotiation-based rip-up and reroute schemes will result in a large number of iterations when generating paths inside CLBs. In this work, we propose a robust routing framework for FPGAs with complex connections between logic elements and switch boxes. We propose a concurrent intra-CLB rerouting algorithm that can effectively resolve routing congestion inside a CLB tile. Experimental results on modified ISPD 2016 benchmarks demonstrate that our framework can achieve 100% routability in less wirelength and runtime, while the state-of-the-art `VTR 8.0` routing algorithm fails at 4 of 12 benchmarks.

**ACM Reference Format:**
Jiarui Wang, Jing Mai, Zhixiong Di, Yibo Lin. 2023. A Robust FPGA Router with Concurrent Intra-CLB Rerouting . In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567898



(a)



(b)

**Figure 1: (a) An Island architecture commonly used in academic routers [6] where each logic pin inside a CLB is logic equivalent. (b) An FPGA architecture where each logic pin inside a CLB is not logic equivalent.**

## 1 INTRODUCTION

Routing is the most time-consuming stage in the design flow for Field Programmable Gate Arrays (FPGAs). As shown in Figure 1(b), An FPGA device consists of configurable logic blocks (CLBs) connected by *global switch boxes* (GSBs). A typical CLB has a *function unit* (FU) inside, which can be further divided into basic logic elements (BLEs) containing logic elements like look-up tables (LUTs) and flip-flops (FFs). A typical FPGA device like `Xilinx Ultrascale` series [1, 2], e.g., *UltraScale VU095*, contains millions of logic elements, resulting in a huge searching space, and meanwhile, FPGA designs nowadays can be highly congested with high-fanout nets. A router needs to find legal embeddings of nets in pre-fabricated routing resources with configurable switch boxes on an FPGA device. As the sizes of both FPGA devices and designs grow rapidly, routing is becoming the bottleneck for FPGA design closure, i.e., taking 41-86% runtime in both commercial and academic flows [3]. Therefore, it is important for FPGA routers to be efficient and high-quality.

Existing FPGA routing algorithms mostly follow the negotiation-congestion based algorithm [4], where nets are sorted in a specific order, and each net is routed with a PathFinder kernel. The routing congestion in the negotiation-congestion based algorithm is resolved by iterative rip-up and reroute schemes. The literature has explored how to improve the performance of PathFinder [4]. Lemieux et al. [5] also propose to divide the routing into global routing and detailed routing to improve efficiency. `VTR 8.0` [6] proposes a *lazy* method to improve routing scalability for large designs and high-fanout nets. Many works [6–8] have proved only rip-up and reroute congested sink pins of congested nets rather than rip-up the whole congested nets is helpful to reduce runtime. There is also a line of studies exploring parallelizing the routing algorithms for better efficiency [9–13], which can reduce the runtime at certain costs of the solution quality.

Academic routers like `VTR 8.0` router [6] assume that each logic pin inside a CLB is logic equivalent. As shown in Figure 1(a), each logic pin inside a CLB can connect to any I/O pin of that CLB. Signals pass through I/O pins of a CLB will be connected to routing tracks and be switched to other tracks by switch boxes

*Corresponding author

*ASPDAC '23, January 16–19, 2023, Tokyo, Japan*
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567898

(SBs). However, the increasing scale and complexity of modern FPGAs have made connections inside a CLB more complex. As shown in Figure 1(b), logic pins inside a CLB shall go through *local switch box* (LSB) to connect to GSB and can only connect to some certain GSB routing tracks, which cause more routing congestion at the logic pins inside the CLBs. Such large solution space and high design complexity challenge the routing algorithm, as an FPGA router needs to find paths at both the CLB level and the logic element level without conflicts between nets.

In this work, we propose a logic element level router for large-scale FPGA designs. Our router aims to generate a routing solution on both inter-CLB and intra-CLB levels. To resolve the large number of rip-up and reroute iterations, our router abstracts routing inside a CLB as an Integer Linear Programming (ILP) formulation to resolve congestion inside CLBs. This work also unveils the limitations of existing routing strategies under such a large and fine-grain routing task. The major contributions of this work are summarized as follows.

- We propose a robust FPGA router with concurrent intra-CLB rerouting to generate routing solutions at the logic element level.
- We propose a concurrent tile assignment algorithm to do intra-CLB rerouting, which can significantly reduce the routing congestion inside CLB tiles.
- We propose a stencil-based parallelization technique to resolve the inter-tile data dependency while doing concurrent tile assignment.

Compared with the `VTR 8.0` algorithm [6, 14], our router can generate a legal routing solution for all 12 benchmarks, while `VTR` fails at 4 of 12 benchmarks and ends up with more runtime (8.87× slower) and worse quality (19.4% larger wirelength) on the modified ISPD 2016 benchmarks for the logic element level routing.

The rest of the paper is organized as follows. Section 2 describes the FPGA architecture and problem formulation; Section 3 explains the overall flow of our algorithms; Section 4 demonstrates our ILP-Based concurrent tile assignment. Section 5 validates the routing algorithm with experimental results; Section 6 concludes the paper.

## 2 PRELIMINARIES

In this section, we introduce our routing architecture and the problem formulation of the routing problem.

### 2.1 FPGA Routing Module

In modern FPGAs, logic pins inside CLBs are not logic equivalent, which is different from the asummption of many academic routers like `VTR` [14]. Figure 1(b) shows a simplified version of our routing architecture. Some tracks will be unreachable for a certain logic pin. Meanwhile, congestion will appear at both GSB channels and inside CLBs.

To route a placed netlist, the most commonly used idea is to abstract the FPGA architecture as a routing resource graph (RRG). An RRG is a directed graph $G(V, E)$. Each vertex $v \in V$ represents a logic pin or a cluster of logic pins due to pin merging and swapping technique (Section 3.3). and has a finite capacity for nets to share. Each directed edge $e = (u, v) \in E$ represents a logic connection between two routing resources, where signal is sent from vertex $u$ to vertex $v$.

### 2.2 Negotiation-Congestion Based Routing

Most academic routers use Pathfinder [4] as the basic routing strategy to route an FPGA design. Its basic idea is to iteratively rip-up congested nets and reroute them using a path search algorithm.

PathFinder [4] determines the cost $c(n)$ of an RRG vertex $n$ using the following function when searching for a routing path to a sink:

$$c(n) = (b(n) + h(n)) * p(n) \tag{1}$$

The base cost $b(n)$ is determined at the initial of the routing phase. History cost $h(n)$ is related to history congestion on vertex $n$ in the previous iterations. Present cost $p(n)$ is related to the number of other nets using RRG vertex $n$. Readers are referred to [4] for calculation method of them.

The kernel algorithm of academic FPGA routers like `VTR 8.0` router [6] is similar to PathFinder while they use A* search to find a routing path from a source pin to a sink pin. They add congestion cost each iteration to resolve routing congestion. However, their router has fewer effects when considering CLBs whose pins are not logic equivalent. To deal with such circumstances, we need a router to concentrate more on resolving routing congestion while focusing on reducing the wirelength of the design.

### 2.3 FPGA Routing Constraints

A legal routing solution shall satisfy the following constraints. For any net, there must exist a connection path from the source pin of the net to each pin of the net. This means that we will generate a routing tree for each net. For each routing tree, its root is the source of the net, and each leaf of the routing tree is a sink of the net. Furthermore, for any routing resource in the routing architecture, it shall not be used by nets more than its capacity.

In this work, our router focuses on finding a legal routing result of each net while optimizing the wirelength of the routing solution.

## 3 ROUTING FRAMEWORK

In this section, we introduce our routing framework.

### 3.1 Overview of the routing flow

The overall flow of our router is shown in Figure 2. Our router takes placement result and FPGA routing architecture as input and generates a routing solution for the given netlist. The routing flow consists of 2 phases: (1) global routing and (2) detailed routing. As Figure 3 shows, we know each logic element locates after placement. We generate an inter-CLB level routing solution in the global routing phase and generate a logic element level routing solution in the detailed routing phase.

The kernel algorithm of our global router and detailed router is based on PathFinder [4], which is commonly used in academic FPGA routers. As shown in previous section, the kernal idea is to route the given netlist iteratively by rip-up and reroute congest nets at each iteration. During rip-up and reroute phase, we only reroute those congested sinks to save route time.

After the first few iterations of detailed routing, most nets will have a routing result with no routing congestion. Most congestion is located in logic pins inside CLBs and solving them will cost many rip-up and reroute iterations. Our router will use an ILP-based concurrent metric after the first few iterations to resolve most of those congestion inside CLBs and iteratively rip-up and reroute other congestion.

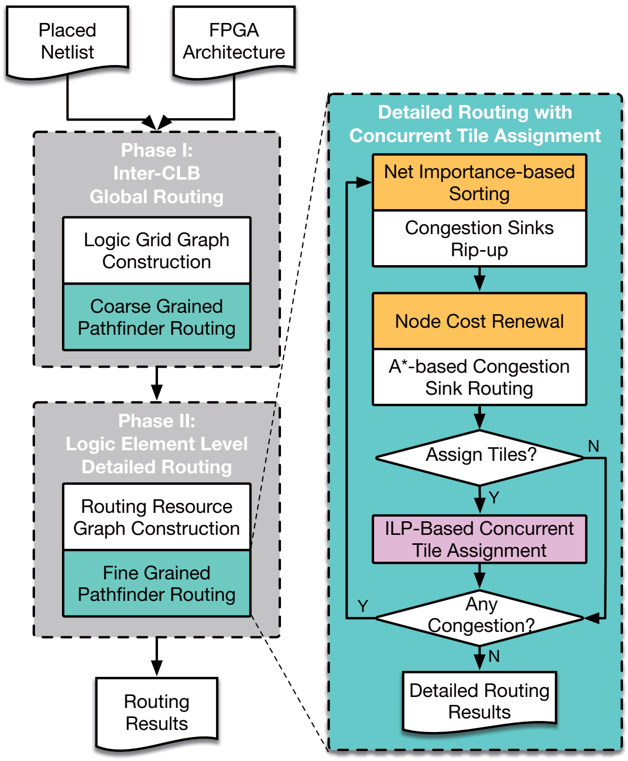

Placed Netlist
FPGA Architecture
Phase I: Inter-CLB Global Routing
Logic Grid Graph Construction
Coarse Grained Pathfinder Routing
Phase II: Logic Element Level Detailed Routing
Routing Resource Graph Construction
Fine Grained Pathfinder Routing
Routing Results
Detailed Routing with Concurrent Tile Assignment
Net Importance-based Sorting
Congestion Sinks Rip-up
Node Cost Renewal
A*-based Congestion Sink Routing
Assign Tiles?
N
Y
ILP-Based Concurrent Tile Assignment
Any Congestion?
Y
N
Detailed Routing Results

**Figure 2: Our proposed routing flow.**

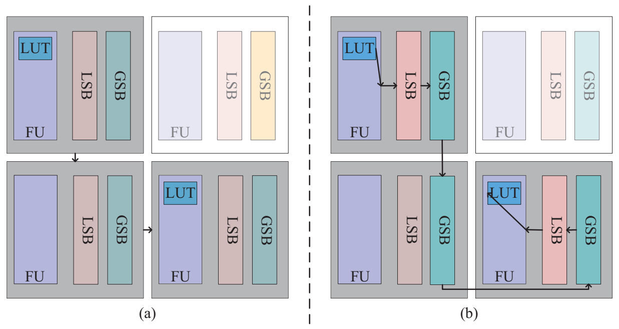


**Figure 3: An overview of two stages of our router: (a) inter-CLB level global routing (b) logic element level detailed routing.**

## 3.2 Global Routing

The goal of global routing is to generate a coarse routing result at inter-CLB level. The result of global routing can guide detailed router's behavior when routing nets connecting different CLBs. Most existing FPGA routers [6, 12] do not have a global routing stage. They directly try detailed routing by applying PathFinder [4] algorithm. However, we find that directly solving detailed routing is impractical when handling large-scale designs.

In this work, we propose a global routing algorithm that considers the potential routability issues in detailed routing. In other words, we try to avoid congested routing regions and leave enough resources for detailed routing. Using the global routing result to guide the detailed router will help reduce the search space and the number of iterations.

The global router abstracts the FPGA layout as a grid graph by regarding each CLB as a vertex. If two CLBs are connected, we add one capacity to the directed edges connecting two vertices representing two logic grids. We define the weight of a certain edge as the Manhattan distance between two CLBs.

After constructing the grid graph, we call pathfinder to route all nets and use the following function to evaluate the cost of a vertex $v$ when adding grid vertex $v$ into the priority queue expanding from grid vertex $u$:

$$cost(v) = prev(u) + weight(u, v) + pred(v). \tag{2}$$

The previous cost $prev(u)$ is the sum of edge weight connecting to grid vertex $u$. We use Manhattan distance to estimate the future cost to route to sink grid vertex $s$. So the prediction cost $pred(v)$ is defined as the Manhattan distance from the current grid vertex $v$ to sink grid vertex $s$. $weight(u, v)$ is initially defined as Manhattan distance between two grid vertex $u$ and $v$. The weight of a certain edge will increase linearly increase as its remaining capacity drops to a certain threshold. This helps to achieve sparse routing solutions and improve detailed routability.

## 3.3 Detailed Routing

The goal of detailed routing is to generate the logic element level routing result of each net with the guidance of global routing results. Different from global routing working on a coarse-grained graph, detailed routing needs to finish all the routing on a fine-grained graph considering both inter- and intra- CLB tiles. Our detailed routing follows a similar negotiation-congestion based routing scheme to global routing. And we use following metrics to enhance our detailed routing algorithm.

**Pin merging and swapping**. Since each logic pin inside a CLB is not logical equivalent, for those nets with multiple sinks, different sinks may be connected to different global switch box tracks. Thus, route those nets may have to cover multi-tracks of global switch boxes. Using multiple global switch box tracks to route a single net leads to an increase of the wirelength and lack of routing resources. For example, the most used logic element in an FPGA design is LUT. A Single LUT has multiple inputs. If we reorder the input order of a LUT, we can recalculate the truth table of this LUT to guarantee the correctness of its result. Using this feature, our router can resolve the issue above by merging vertices representing inputs of a LUT into a single vertex whose capacity is the number of inputs of the LUT.

**Search space reduction and expansion**. When searching for routing paths, our router only searches those RRG vertices covered by global routing result to reduce runtime. For a net, routing congestion may be caused by not enough routing resources inside its routing guide. So when ripping-up congestion vertices, we expand the nearby grids of the congested vertex located into the routing guide of the congested net every few iterations to slowly expand the search space.

**Net ordering**. In each iteration, our router first sorts all nets in the netlist considering their size and reroute times to determine the importance of each net. When ripping up congestion nets at each routing iteration, our detailed router does not rip-up the top-$K$ important nets congested at a vertex to reduce runtime. $K$ is the capacity of the congested vertex.

**Concurrent tile assignment**. Most routing congestion at vertices representing logic pins of GSBs will be resolved in the first few rip-up and reroute iterations by using the global routing result. However, as there exist complex connections between switch boxes and functional units, resolving congestion inside a CLB will cost much more rip-up and reroute iterations. Therefore, we assign the routing result of all nets inside a CLB where congestion exists concurrently after the first few iterations, which will decrease the number of nets to be rerouted in future iterations, and also may decrease the number of rip-up and reroute iterations. The detail of concurrent tile assignment is explained in Section 4.

## 4 CONCURRENT TILE ASSIGNMENT

In this section, we consider each CLB as a logic tile, and describe our ILP concurrent tile assignment metric. Section 4.1 introduces how we apply ILP-based concurrent tile assignment to resolve congestion intra tiles, and Section 4.2 explains how we resolve data dependency between neighbor tiles by using the stencil-based parallelism strategy.

### 4.1 ILP-Based Concurrent Tile Assignment

ILP-based mapping metrics have been successfully used in Coarse-Grained Reconfigurable Architecture (CGRA) mapping problem [15], which equals place and route in the FPGA design flow. ILP mapper concurrently generates P&R results by formulating mapping as an ILP problem, which inspires us to generate routing results using ILP concurrently. As the scale of an FPGA is much larger than a CGRA, it is difficult to use ILP mapping to solve the whole routing problem. However, as there are only limited routing resources inside a logic tile, we can use ILP to assign the routing paths of all nets inside a logic tile concurrently. Therefore, after the first few iterations of detailed routing, our router will use ILP-based tile assignment to resolve congestion inside CLB for those tiles with congestion inside, which will make the number of nets to reroute in future rip-up and reroute iterations less.

**Table 1: List of symbols.**

| Symbol | Description |
|---|---|
| $\mathcal{V}$ | Routing vertices in the RRG. |
| $\mathcal{E}$ | Directed edges in the RRG. |
| $\mathcal{N}$ | Nets inside the logic tile. |
| $R_{e,j}$ | Binary variable representing whether edge $e$ is used to route net $j$. |
| $S_{e,j,k}$ | Binary variable representing whether edge $e$ is used to route sink $k$ of net $j$. |
| cap$(v)$ | routing capacity of vertex $v$. |
| FI$(v)$ | Fan-in edges of vertex $v$. |
| FO$(v)$ | Fan-out edges of vertex $v$. |
| COST$(e)$ | Routing cost of edge $e$ in the RRG. |
| SOURCE$(j)$ | Routing source vertex of net $j$. |
| SINK$(j)$ | Routing sink vertices of net $j$. |
| SINK$(j,k)$ | Routing sink vertex $k$ of net $j$. |

We list all the symbols and their descriptions in Table 1. The target of our ILP router is to find paths to route each net inside a logic tile with no congestion. Also, the total routing cost shall be minimized. We list all the ILP constraints as follow:

**Capacity Constraint**. For any vertex of RRG, it cannot be used to route nets more than its capacity.

$$\begin{gathered}\textstyle\sum_{e,j} R_{e,j} \le \text{cap}(v),\\ \forall v \in \mathcal{V}, j \in \mathcal{N}, e \in \text{FI}(v).\end{gathered} \tag{3}$$

**Implied Routing**. For any edge $e$ of RRG, if it is used to transfer signal for a certain net $j$, then it must transfer the signal to at least a certain sink $k$ of net $j$.

$$\begin{gathered}S_{e,j,k} \le R_{e,j},\\ \forall e \in \mathcal{E}, j \in \mathcal{N}, k \in \text{SINK(j)}.\end{gathered} \tag{4}$$

**Net Source Constraint**. If vertex $v$ in the RRG represents the routing source of a certain net $j$, then for any sink $k$ of net $j$, there must be a fan-out edge $e$ of $v$ used for transferring signal to $k$.

$$\begin{gathered}\textstyle\sum_e S_{e,j,k} = 1,\\ \forall e \in \text{FO}(v), v = \text{SOURCE(j)}, \forall k \in \text{SINK(j)}.\end{gathered} \tag{5}$$

**Net Sink Constraint**. For any vertex $v$ in the RRG representing a certain sink $k$ of net $j$, there must be a fan-in edge of $v$ used to transfer signal to $v$.

$$\begin{gathered}\textstyle\sum_e S_{e,j,k} = 1,\\ \forall e \in \text{FI}(v), v = \text{SINK(j,k)}.\end{gathered} \tag{6}$$

**Path progression**. For vertex $v$ and net $j$, if $v$ is neither source of $j$ nor sink $k$ of $j$, then the number of input signal shall equal the number of output signal. This means if there is a fan-in edge of $v$ used to transfer signal to sink $k$ of net $j$, there must be a fan-out edge of $v$ used to transfer signal to sink $k$ of net $j$.

$$\begin{gathered}\textstyle\sum_{e_{in}} S_{e_{in},j,k} = \sum_{e_{out}} S_{e_{out},j,k},\\ \forall j \in \mathcal{N}, k \in \text{SINK(j)}, v \neq \text{SOURCE(j)}\&v \neq \text{SINK(j,k)},\\ e_{in} \in \text{FI}(v), e_{out} \in \text{FO}(v).\end{gathered} \tag{7}$$

**ILP Objective**. The ILP constraints above have ensured there exists a routing path for any sink of any net. Therefore, we only need to minimize the routing cost.

$$\begin{gathered}\textstyle\sum_{e,j} \text{COST}(e) R_{e,j},\\ \forall e \in \mathcal{E}, j \in \mathcal{N}.\end{gathered} \tag{8}$$

Note that in our routing resource graph, vertices representing source pin and sink pins of a net don't have input edges or output edges. Therefore, our ILP formulation ensures a legal resolution if there exists.

Routing nets connecting different logic tiles will also impact the routing result inside a CLB. Therefore, when routing a certain logic tile, our concurrent tile assignment will also consider its neighbor logic tile. As shown in Algorithm 1, for each logic tile, we dump a local copy of RRG only containing those vertices in the tile and its neighbor tile first. Then, we use nets whose source vertex is located in that tile as the local netlist $\mathcal{N}$. After that, we call the solver to solve the ILP problem above. If we find an optimal solution to the ILP problem, then we will update the ILP routing result to the routing solution. For those tiles whose result is infeasible, which is caused by nets connecting logic tiles which are not nearby, we use rip-up and reroute scheme to resolve routing congestion.

**Algorithm 1:** ILP-Based Concurrent Tile Assignment

```
  Input: A logic tile with routing congestion
  Output: Tile assignment result
1 Dump local copy of RRG consisting of vertices in input tile
   and its neighbor tiles.                          ▷ Section 4.2
2 Collect Nets start from input tile.
3 solve_ILP()                                       ▷ Section 4.1
4 if ILP optimal result found then
5     Update Route Result                           ▷ Section 4.2
6 end
```

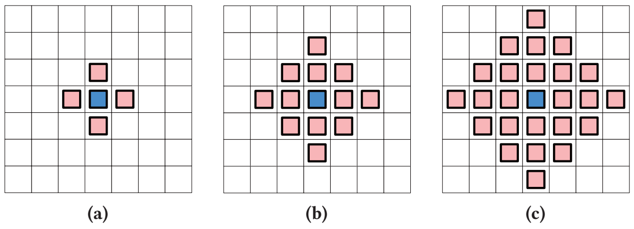


**Figure 4: (a), (b) and (c) are three stencil shapes for a structured tile grid with radius 1, 2, and 3 respectively. The blue tile is the center tile and the red tiles are the data-dependent neighboring tiles.**

## 4.2 Stencil-Based Parallelism for Inter-Tile Dependency

*4.2.1 Stencil-like Read-Write Dependency.* The local routing graph construction for a tile to update to read the tile itself and its data-dependent neighboring tiles. Meanwhile, after solving the local ILP-based tile assignment problem, we also need to write back results to the tile to update the neighboring tiles. This read-write dependency on the structured tile grids forms a inter-tile *stencil* computation pattern.

The stencil is characterized by a regular shape consisting of a center tile and its data-dependent neighboring tiles. Figure 4(a)-4(c) show three different stencil shapes on a 2D structured grid. Because the routing tracks are either horizontal or vertical, we define the radius of a stencil as the largest *Manhattan* distance between the tile to update and its neighboring dependent tiles. The larger the radius, the more accurate the perception of local routing congestion will be, but also result in a more complex local ILP problem and poorer condition for parallel execution, and vice versa. Our router chooses the stencil scheme with radius as *one* to fully exploit the parallelism.

*4.2.2 Synchronization-free Stencil Scheduling.* Due to the read-write conflict, two stencil computations can not be performed in parallel if their stencil shapes overlap with each other, as illustrated in Figure 5(a). This conflicting relation can be equivalently transformed onto the center tiles, i.e., if the *Manhattan* distance between two center tiles is less than or equal to two, they can not perform the ILP-based tile assignment in parallel, which is unfriendly to parallel execution scheduling. Meanwhile, the tiles' local ILP problem workload and execution time vary. This is also unfriendly to partitioning-based scheduling for parallel computation, because the synchronization is bounded by waiting for the slowest workload, which significantly ruins the CPU utilization. Therefore, we propose a synchronization-free stencil scheduling method to remedy the aforementioned issues.

Figure 5(c) shows our proposed running dependency graph on the 2D structured grid. Immediate predecessors for a tile (the purple one) are the one above it, the one to the left of it, the one on the top-left corner of it, and the one on the bottom-left corner of it (the four cyan ones). A tile can perform stencil computation if all of the immediate predecessors are already finished. This method has the feasible guarantee that any parallelism schedule corresponding with this dependency graph can ensure no two tiles with *Manhattan* distance less than or equal to two perform stencil computation simultaneously. Besides, our proposed method is synchronization-free over the partitioning-based parallel method,

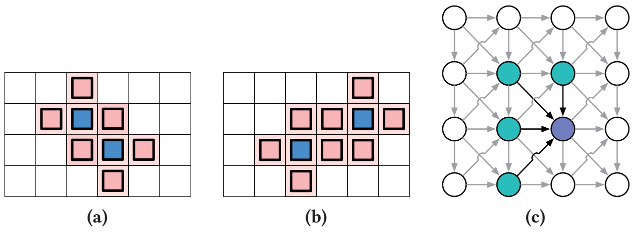


**Figure 5: (a) shows the scenario that two stencil shapes with radius one overlapping with each other, in which case the two stencils are conflict and can not be scheduled in parallel. (b) illustrates another non-conflicting scenario for two stencil shapes with radius one. The conflicting condition can be equivalently transformed into two stencil shapes conflict with each other if the *Manhattan* distance between their center tiles (the blue ones) is less than or equal to two. (c) Our proposed running dependency graph on a simplified $4 \times 4$ grid. The running dependency graph is a directed acyclic graph, where the nodes are the tiles and the edges are the running dependency between the tiles.**

and have a dynamically-adjusted schedule according to the actual runtime of each tile.

## 5 EXPERIMENTAL RESULTS

We implemented our algorithm in C++ and conducted experiments on a Linux machine equipped with an Intel Xeon Gold 6230 CPU (2.10 GHz) and 512 GB RAM. We call Gurobi [16] optimizer to solve ILP problem, and we use Taskflow [17], an open-source parallel computing tool, to implement our stencil-based scheduling when doing tile assignment. We tested our work on the modified ISPD 2016 FPGA Contest Benchmark [18].

We take the ISPD 2016 FPGA placement contest benchmarks and incorporate the industrial FPGA routing architecture similar to `Xilinx Ultrascale VU095`. Control signals and clock signals are excluded due to lack of clock routing architecture. We also obtain the placement solutions from the previous contest winner [19]. We adapt the state-of-the-art VTR routing algorithm [6] (abbreviated as `Adapted VTR`) to support the industrial routing architecture.

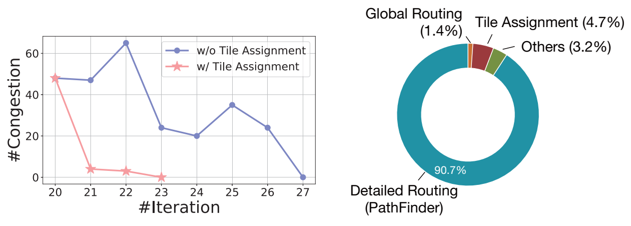
Table 2 shows the comparison of our algorithm with the VTR algorithm on modified ISPD 2016 benchmark. Our logic element level router successfully generate a legal result on all designs, but the VTR algorithm cannot generate a legal solution on 4 of 12 benchmarks within 24 hours. On modified ISPD2016 benchmarks, our router is 8.87× faster than `VTR`, and wirelength of routing results generated by `VTR` algorithm are 19.4% more than ours.

In practice, we do tile assignment after the first 21 iterations of detailed routing. Using design FPGA08 as example, we show how our tile assignment impacts the routing scheme in Figure 6(a). Before doing tile assignment, there are 48 congested vertices to be resolved. After doing tile assignment, most of congested vertices are resolved and only need 3 more iterations to resolve other congested vertices while routing without tile assignment takes 4 more iterations to resolve the congested vertices and need to reroute more nets in each iteration.

Figure 6(b) shows the runtime breakdown of our router on design FPGA08. The process of detailed routing is the major portion of runtime, taking 90.71% of the total runtime. Doing tile assignment and global routing only takes 4.68% and 1.42% runtime of the

**Table 2: Routing status, routed wirelength ($\times 10^5$) and runtime comparison on modified ISPD 2016 contest benchmarks [18]**

| Design | #Nets(K) | #Nodes(K) | Adapted VTR [6] | | | | | Ours. | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Routed Rate | WL | RWL | RT | RRT | Routed Rate | WL | RWL | RT | RRT |
| FPGA01 | 105 | 105 | 100.00% | 4.147 | 1.038 | 76m | 4.00 | 100.00% | **3.997** | **1.000** | **19m** | **1.00** |
| FPGA02 | 167 | 166 | 100.00% | 8.232 | 1.096 | 5m | 1.00 | 100.00% | **7.507** | **1.000** | 5m | 1.00 |
| FPGA03 | 428 | 421 | 100.00% | 39.529 | 1.254 | 88m | 5.50 | 100.00% | **31.528** | **1.000** | **16m** | **1.00** |
| FPGA04 | 420 | 423 | 100.00% | 74.776 | 1.169 | 614m | 14.28 | 100.00% | **63.973** | **1.000** | **43m** | **1.00** |
| FPGA05 | 433 | 425 | 68.30% | 146.502 | 1.112 | >24H | >7.54 | **100.00%** | **131.786** | **1.000** | **191m** | **1.00** |
| FPGA06 | 713 | 704 | 100.00% | 76.171 | 1.205 | 514m | 10.71 | 100.00% | **63.200** | **1.000** | **48m** | **1.00** |
| FPGA07 | 716 | 707 | 96.41% | 140.575 | 1.273 | >24H | >13.59 | **100.00%** | **110.457** | **1.000** | **106m** | **1.00** |
| FPGA08 | 725 | 717 | 100.00% | 134.834 | 1.301 | 365m | 5.14 | 100.00% | **103.487** | **1.000** | **71m** | **1.00** |
| FPGA09 | 876 | 867 | 99.62% | 166.927 | 1.215 | >24H | >11.61 | **100.00%** | **137.412** | **1.000** | **124m** | **1.00** |
| FPGA10 | 961 | 952 | 100.00% | 66.474 | 1.098 | 506m | 12.34 | 100.00% | **60.547** | **1.000** | **41m** | **1.00** |
| FPGA11 | 851 | 845 | 89.30% | 192.360 | 1.331 | >24H | >9.47 | **100.00%** | **144.572** | **1.000** | **152m** | **1.00** |
| FPGA12 | 1111 | 1103 | 100.00% | 93.146 | 1.236 | 697m | 11.24 | 100.00% | **75.371** | **1.000** | **62m** | **1.00** |
| Norm. | - | - | 96.14% | - | 1.194 | - | >8.87 | **100.00%** | - | **1.000** | - | **1.00** |



(a) (b)

**Figure 6: (a) Comparision of number of rip-up and reroute iterations and number of congested vertices in each iteration between routing with tile assignment and without tile assignment on design FPGA08. (b) Runtime breakdown on design FPGA08.**

total flow, and they directly impact the efficiency and quality of the routing result. The other part of runtime is the cost of reading and writing files, which takes 3.29% runtime of total flow.

## 6 CONCLUSION

In this paper, we propose an FPGA router at the logic element level. By analyzing the routing challenges in FPGA, we propose an ILP-based concurrrent tile assignment metric to resolve routing congestion inside CLBs. We also propose routing enhancement techniques to pin merging and swapping, search space reduction and expansion, and routing ordering strategy that are effective to reduce rip-up and re-route iterations in a negotiation-congestion based routing algorithm. Experimental results on modified ISPD 2016 benchmarks demonstrate that our router can achieve 100% routability while the state-of-the-art VTR algorithm fails at 4 of 12 benchmarks and ends up with more than 8.87× runtime and 19.4% larger wirelength.

## ACKNOWLEDGE

This project is supported in part by the National Key Research and Development Program of China (No. 2021ZD0114702).

## REFERENCES

[1] Ultrascale architecture clocking resources. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug572-ultrascale-clocking.pdf

[2] Ultrascale architecture clb slices. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf

[3] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial cad," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 2, Mar. 2015.

[4] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for fpgas," in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[5] G. G. Lemieux and S. D. Brown, "detailed routing algorithm for allocating wire segments in field-programmable gate arrays," in *Proc. Physical Design Workshop*, 1993, pp. 215–226.

[6] K. E. Murray, S. Zhong, and V. Betz, "Air: A fast but lazy timing-driven fpga router," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 338–344.

[7] D. Wang, Z. Duan, C. Tian, B. Huang, and N. Zhang, "A runtime optimization approach for fpga routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1706–1710, 2018.

[8] D. Vercruyce, E. Vansteenkiste, and D. Stroobandt, "Croute: A fast high-quality timing-driven connection-based fpga router," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 53–60.

[9] M. Shen and N. Xiao, "Fine-grained parallel routing for fpgas with selective expansion," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 577–586.

[10] ——, "Load balance-aware multi-core parallel routing for large-scale fpgas," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 595–602.

[11] C. H. Hoo and A. Kumar, "Paradro: A parallel deterministic router based on spatial partitioning and scheduling," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 67–76.

[12] Y. Zhou, D. Vercruyce, and D. Stroobandt, "Accelerating fpga routing through algorithmic enhancements and connection-aware parallelization," *ACM transactions on reconfigurable technology and systems*, vol. 13, no. 4, pp. 1–26, 2020.

[13] M. Shen, G. Luo, and N. Xiao, "Combining static and dynamic load balance in parallel routing for fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1850–1863, 2021.

[14] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. ElDafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "Vtr 8: High performance cad and customizable fpga architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, 2020.

[15] Y. Guo, J. Wang, J. Zhang, and G. Luo, "Formulating data-arrival synchronizers in integer linear programming for cgra mapping," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 943–948.

[16] Gurobi Optimization Inc., "Gurobi optimizer reference manual," http://www.gurobi.com, 2022.

[17] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, "Taskflow: A lightweight parallel and heterogeneous task graph computing system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1303–1320, 2022.

[18] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal, "Routability-driven FPGA placement contest," in *ACM International Symposium on Physical Design (ISPD)*, 2016, pp. 139–143.

[19] Y. Meng, W. Li, Y. Lin, and D. Z. Pan, "elfplace: Electrostatics-based placement for large-scale heterogeneous fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.