Superfast Full-Scale GPU-Accelerated Global Routing

Shiju Lin
The Chinese University of Hong Kong
sjlin@cse.cuhk.edu.hk

Martin D.F. Wong
The Chinese University of Hong Kong
mdfwong@cuhk.edu.hk

ABSTRACT

Global routing is an essential step in physical design. Recently there are works on accelerating global routers using GPU. However, they only focus on certain stages of global routing, and have limited overall speedup. In this paper, we present a superfast full-scale GPU-accelerated global router and introduce useful parallelization techniques for routing. Experiments show that our 3D router achieves both good quality and short runtime compared to other state-of-the-art academic global routers.

1 INTRODUCTION

Routing is an essential step in modern very large-scale integration (VLSI) design. It creates a wiring plan to connect different elements in a circuit. Due to its high complexity, it is divided into two stages, global routing and detailed routing. Global routing aims at providing useful route guides for detailed routing to shorten detailed routing time. Global routing quality can greatly affect the running time and the quality of detailed routing. The speed of global routing is also important. Routability-driven placement relies on global routing for accurate routability estimation, and faster global routing can significantly improve both the running time and the quality of routability-driven placement. Traditional global routing problem provides edge capacities and netlist information, and the objective is to generate routing paths to connect each net minimizing overflows and wirelength. Modern global routing problem, which was introduced at the 2019 CAD contest at ICCAD, targets at closing the gap between global routing and detailed routing. The LEF/DEF files for detailed routing are directly used as the input for global routing. The output is a set of connected rectangular route guides. It offers high flexibility in designing global routing models and utilizing detailed routing specific information. For example, the way to compute the capacity of an edge between two G-cells can be determined by the global router to produce good route guides for detailed routing. The global routing quality is evaluated using an academic detailed router Dr. CU[8]. A sophisticated cost function is used to measure the detailed routing quality including wirelength, via usage, non-preferred usage and various design rule violations.

Global routing typically has two stages, pattern routing and rip-up and reroute. Pattern routing generates initial routing results for all nets. FLUTE[3] is usually used for rectilinear steiner minimum tree construction in pattern routing. Obstacle-avoiding rectilinear steiner minimum trees[2, 9] are an alternative, which can better consider congestion in an early stage. The solution space of pattern routing is intentionally restricted to shorten running time by only allowing certain routing topologies such as L-shape, Z-shape and 3-bend routing. The quality of pattern routing will be inevitably affected by the constraints, but it is usually good enough for routing most nets. Some edges may have too many overflows after pattern routing, and the nets using overflowed edges will be ripped up and re-routed using maze routing in the second stage. Maze routing does not have any constraints on its routing topology and is much more flexible than pattern routing. It is able to avoid congested area and find feasible paths for most hard-to-route nets.

The routing region is a multi-layer grid graph. A popular routing technique adopted by many 2D global routers such as NCTU-GR 2.0[13], SPRoute[7] and FastRoute 4.0[14] is to compress multiple layers to a single-layer grid graph. This compression reduces the complexity and difficulty of 3D routing. After 2D routing, the 2D global routing results are mapped back to 3D routing results by layer assignment. However, compressed 2D grid graphs are less accurate than 3D grid graphs in terms of routing resources, which could limit the global routing quality. There are also 3D routers such as CUGR[11]. It has both 3D pattern routing and 3D maze routing. Although the grid graph remains 3D in CUGR, it uses another technique called two-level maze routing, coarse-grained maze routing and fine-grained maze routing, to accelerate the RRR process. The two-level idea is similar to the compression in 2D routers but the coarsening is in another dimension. However, the two approaches have one key difference. The 2D routing and layer assignment are completely separated in 2D routers. Layer assignment is conducted after all the nets are routed in 2D. In two-level maze routing, the coarse-grained routing is directly followed by the fine-grained routing when a net is being routed. Two-level routing will always generate a complete fine-grained 3D routing result for a net, retaining the 3D routing benefit of accurate routing resource allocation.

The most widely used strategy to accelerate global routing is multi-threading. A batch of nets that do not overlap in their bounding boxes can be routed simultaneously. Dr. CU[1] proposed several heuristics for generating balanced batches with high routing quality. SPRoute[7] studied fine-grained parallelism on path searching of maze routing. Compared to multi-threading with CPU, GPU has more cores and is potentially a good platform for fast global routing. Han et al.[5] developed GPU-based A* search for global routing. GAMER[10] proposed a novel parallel algorithm for finding shortest paths to accelerate maze routing on regular grid graphs. FastGR[12] focused on pattern routing and parallelized L-shape routing for 2-pin nets.

This research was partially supported by ACCESS - AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549474

Accelerating global routing via GPU has been demonstrated to be effective by prior works. In this paper, we introduce a superfast full-scale GPU-accelerated global router. The main contributions are summarized as follows.

- We propose a new simple and effective parallel L-shape pattern routing. A fast and high-quality Z-shape pattern routing is developed based on the parallel L-shape routing algorithm.
- We parallelize the fine-grained maze routing with novel techniques. We also optimize the previous parallel coarse-grained maze routing using CUDA graphs.
- Extensive experiments are conducted to show the effectiveness of our proposed methods. Experimental results show that our GPU-accelerated global routing is faster than other state-of-the-art global routers and with the best quality.

The rest of the paper is organized as follows. Section 2 introduces the problem formulation and an overview of our global router. Our parallel pattern routing and parallel maze routing are discussed in Section 3 and Section 4 respectively. Section 5 presents the experimental results. Our paper is concluded in Section 6.

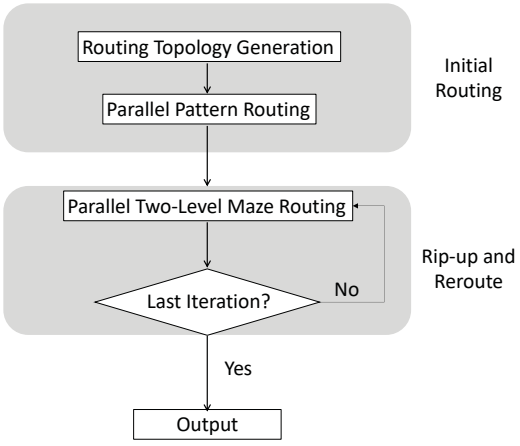

Figure 1: Overall Flow

2 PRELIMINARIES

2.1 Problem Formulation

The routing region of global routing can be represented by a set of global routing cells (G-cells). They form a multi-layer grid graph, and adjacent G-cells are connected by edges. An edge connecting G-cells on the same layer is called a wire, and an edge connecting G-cells from different layers is called a via. Each edge has a capacity, which is how many times this edge can be used. Overusing an edge will lead to overflows. A net is a collection of pins located at different G-cells, which need to be connected by edges.

Given a 3D grid graph and a set of nets, traditional global routing problem aims at finding paths for each net so that all the pins in a net are connected with minimized overflows and total wirelength. To bridge the gap between global routing and detailed routing, the 2019 ICCAD contest on global routing did not directly evaluate global routing results based on overflows and total wirelength. The new evaluation uses the global routing results as route guides for a detailed router, and the metrics are all detailed routing related.

Table 1: Comparison of LEF/DEF-Based Global Routers

Router	Dimension	Pattern Routing	Maze Routing
SPRoute 2.0[6]	2D	Multi-Threading	
CUGR[11]	3D		
GAMER[10]		Multi-Threading	Partially GPU
FastGR[12]		GPU	Multi-Threading
Ours		GPU	

2.2 Overview

The overall flow of our 3D global routing is shown in Fig. 1. Our cost scheme is adopted from [11]. The pattern routing is a GPU-accelerated L/Z-shape routing with various parallelization techniques. The 3D maze routing is divided into two levels, coarse-grained and fine-grained. The coarse-grained maze routing generates results based on a coarse grid graph, while the fine-grained maze routing uses the coarse-grained results as route guides, and searches for paths on the fine-grained grid graph within the guides. Our coarse-grained routing is developed and improved based on [10], while an efficient fine-grained routing algorithm is designed and parallelized based on the characteristics of the two-level scheme.

Table 1 shows a comparison of our work with other LEF/DEF-based academic global routers. SPRoute 2.0[6] is the only 2D global router and is faster than most 3D routers. GAMER[10] is a novel parallel maze routing algorithm integrated in CUGR. FastGR[12] introduced GPU parallelization of L-shape pattern routing. Comparing with prior works, our router is fully GPU-accelerated with novel parallelization techniques using the shortest running time and giving the best quality.

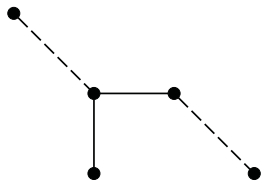

Figure 2: 2D Topology

3 PATTERN ROUTING

3.1 Background

3D pattern routing is more flexible in routing topology and can represent routing resources more accurately. In our 3D pattern routing, a 2D routing topology of a multi-pin net is first generated using FLUTE[3]. The 2D topology is an initial result with some adjustable edges. Fig. 2 shows the 2D topology of a 5-pin net. The dashed black lines represent the edges with multiple possible topologies. Such edges have two possible 2D topologies in L-shape routing. Traditional 2D pattern routing determines the exact 2D topology before layer assignment, while 3D pattern routing uses a dynamic

programming algorithm [11] to simultaneously consider layer assignment and different topologies. An optimal 3D routing topology can be generated by the dynamic programming (DP). The computations of the DP framework can be performed very efficiently, but the most time-consuming part comes from computing the minimum costs connecting two points using 3D L/Z-shape routing.

A parallel L-shape routing method was introduced by FastGR[12]. However, its formulation involves complicated vector and matrix operations. We propose a different parallelization method for L-shape routing, which is simple, fast and easy to understand. More importantly, it does not consume much computing resource, which allows us to extend it to an efficient parallel algorithm for the compute-intensive Z-shape routing.

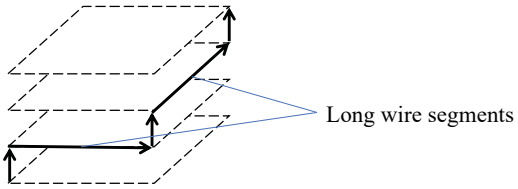

Figure 3: L-Shape Routing

3.2 Parallel L-Shape Routing

Fig. 3 shows an example of 3D L-shape routing. If the 3D grid graph has L layers, there are $2 \times L \times L$ combinations of L-shape routing. The most time-consuming step in L-shape routing is computing the total cost of a long wire segment. A long wire segment refers to a set of consecutive wires as illustrated in Fig. 3. An efficient way to calculate the total cost of a long wire segment is to use prefix sum. Let c_i be the cost of the i-th wire, and $s_i = \sum_{j=1}^{i} c_j$ be the prefix sum. The total cost of a long wire segment consisting of the l-th to the r-th wires can be efficiently computed in constant time by $s_r - s_{l-1}$. The prefix sum calculation can be parallelized using the method described in [10], which can calculate the prefix sum of a row of n elements in $O(\log n)$ time. Morever, different rows or columns of wires do not affect each other, and they can be computed simultaneously. Therefore, it takes $O(\log n)$ time to pre-compute the prefix sum and $O(1)$ for each query of the long wire segment cost.

Our L-shape routing for a single 2-pin connection can be divided into 5 steps as shown by the 5 arrows in Fig. 3. The first step is computing the cost from the starting point to a position on different layers. The second step is computing the cost of a long wire on different layers. The third step is calculating the minimum cost at different layers of the bend points by using vias. The last two steps have the same computational process as the first two steps. Every step can be done sequentially in $O(L)$ time, where L is the number of layers. Note that the third step is more complicated than the first and last steps. In the third step, we need to consider all possibilities of via connection starting from any layer and ending at any other layer, while the starting layer in the first step and the ending layer in the last step are fixed. Although the third step has L^2 pairs of starting and ending layers, it can still be finished in linear time using Algorithm 1, since it is just needed to record the best cost for each ending layer. Note that parallelizing the third step will not be helpful, because atomic operations are needed as each layer is being updated by other layers simultaneously.

Algorithm 1 Via Update

Input:
d_i : minimum cost at layer i of current point
$via(a, b)$: via cost between layer a and layer b of current point
Output: updated d_i

for $i \leftarrow 2$ to L **do**
    $d_i \leftarrow \min\{d_i, d_{i-1} + via(i-1, i)\}$
end for
for $i \leftarrow L-1$ downto 1 **do**
    $d_i \leftarrow \min\{d_i, d_{i+1} + via(i, i+1)\}$
end for

Besides the parallelization of the prefix sum computation for long wire costs to speed up the above simple L-shape scheme, parallelism of our L-shape routing also comes from routing a batch of non-overlapping nets concurrently. In pattern routing, the number of nets is huge and many nets have small bounding boxes. Therefore, the batch size is often large and GPU can be well utilized by the net-level parallelism. The prefix sum needs to be pre-computed before routing each batch of nets. Larger batch sizes imply fewer batches, which means fewer prefix sum computations. This parallel L-shape routing turns out to be very fast in practice.

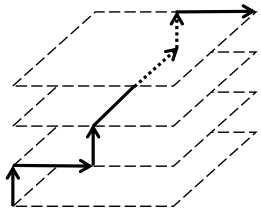

Figure 4: Z-Shape Routing

3.3 Parallel Z-Shape Routing

Fig. 4 shows an example of Z-shape routing which is more complex but flexible than L-shape routing. If the size of the bounding box of a 2-pin net is $n \times m$ and the number of layers is L, Z-shape routing has $L^3 \times (n+m)$ possible paths, while L-shape routing has only $2 \times L^2$ possible paths. The most significant difference between L-shape and Z-shape routing is that Z-shape routing has more choices for the bending points. Our major idea of parallelization for Z-shape routing is to consider all the bending point candidates simultaneously.

Fig. 5 shows different steps of a parallel Z-shape routing example. The process is sequentially divided into 7 steps from (a) to (g). Four threads are used in this example and are indicated by different colors. Each thread corresponds to a possible bending point, and is responsible for the operations related to that point. The technique of pre-computing prefix sum for fast long wire cost queries is also employed here, so each arrow in Fig. 5 needs $O(1)$ time to finish,

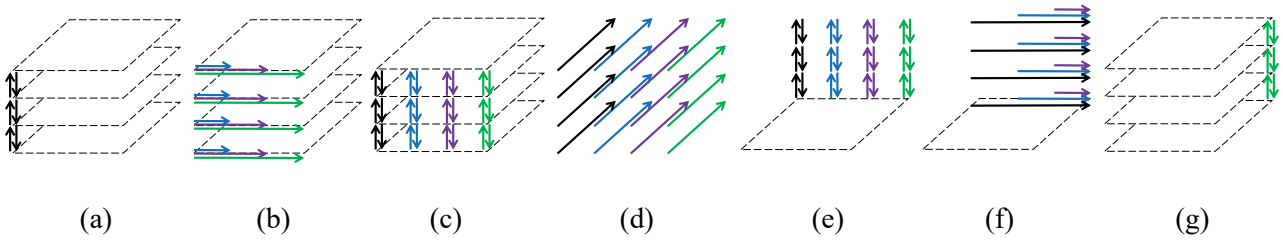

Figure 5: Parallel Z-Shape Routing

and the total running time is $O(L)$. Net-level parallelism is also adopted in a similar fashion as in parallel L-shape routing. Each net in a batch of non-overlapping nets occupies a block in CUDA, and multiple threads are allocated for every block. Ideally, the number of threads assigned to a net should be equal to the number of bending point candidates of the net to fully utilize the parallelism. However, CUDA requires all the blocks that are launched together to have the same number of threads. It will incur huge overhead to launch a block (kernel) for each net in a batch separately. The only solution is to choose a fixed number of threads for all nets. Our implementation chooses 32 as the unified number of threads for the nets, because most nets are small and the warp size in CUDA is 32. If there are T candidate bending points, each thread will be responsible for $\lfloor T/32 \rfloor$ or $\lceil T/32 \rceil$ candidates.

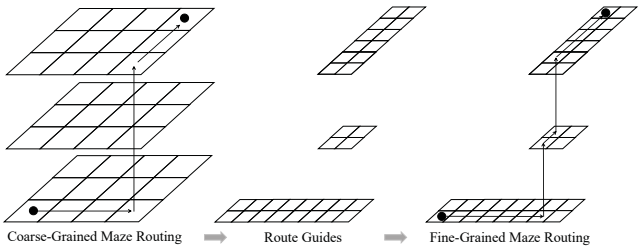

Figure 6: Two-Level Maze Routing

4 MAZE ROUTING

4.1 Two-Level Maze Routing

Maze routing is flexible but time-consuming. Two-level maze routing is a useful technique to reduce running time. Fig. 6 shows an example of two-level routing on a 3-layer 8×6 grid graph. 2×2 G-cells are merged to a coarse-grained G-cell. Coarse-grained maze routing is first applied to find a path on the coarse grid graph. Then, all the coarse-grained G-cells on the path are used as route guides. Only G-cells within the route guides will be considered in the fine-grained routing.

GAMER[10] proposed a novel parallel algorithm to find the shortest path for the coarse-grained maze routing problem. However, it cannot be applied in fine-grained maze routing in which the routing region is irregular. Note that if the regular routing region is used in fine-grained routing, the routing graph will be huge and the efficiency will drop drastically.

4.2 Parallel Fine-Grained Maze Routing

Maze routing of multi-pin nets iteratively connects the current route to the closest unconnected pin. This allows us to decompose multi-pin nets to 2-pin nets by their coarse-grained routing results. A routing path connecting two pins consists of wire segments and via segments. A wire/via segment is a maximal collection of consecutive wires/vias. A routing path can be divided into interleaved wire and via segments. The segments are routed one by one sequentially, but routing within a segment is parallelized.

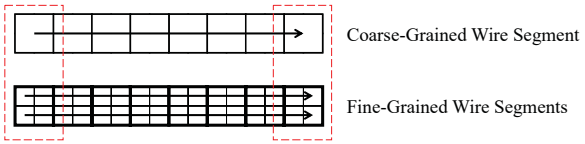

Figure 7: Wire Segments

4.2.1 Wire Segments. Each layer has a preferred routing direction that all routing paths need to follow, so a coarse-grained wire segment corresponds to some independent fine-grained wire segments. An example is shown in Fig. 7, where 2×2 G-cells form one coarse-grained G-cell. Moreover, only the first and the last coarse-grained G-cells contain useful information. The first G-cell contains the cost information from the previous segment and the last G-cell contains the computed cost information for the next segment. These G-cells are indicated by the red dashed boxes in Fig. 7. The intermediate G-cells can be ignored by directly calculating the cost from the first to the last G-cells. The accumulated cost of passing through the intermediate G-cells can be obtained efficiently by pre-computing the prefix sum of the wire cost. Different fine-grained wire segments from the same coarse-grained wire segment can be routed simultaneously.

4.2.2 Via Segments. Handling via segments is more complicated than handling wire segments. Both the fine-grained vias and wires within the G-cells of the coarse-grained via segment should be considered. An example is shown in Fig. 8. In this example, one coarse-grained via segment corresponds to 4 fine-grained via segments and 6 fine-grained wire segments. The fine-grained vias and wires can be used to create complex paths. To parallelize fine-grained routing of a coarse-grained via segment, an approach based on the sweep operation in [10] is used. The vias and wires are considered separately and alternately as shown on the right sub-figure of Fig. 8. They are referred to as wire sweeps or via sweeps since

only wires or vias are used. Alternating wire and via sweeps can correctly create all possible paths [10], and the number of iterations is usually small for minimizing via usage. Different wire segments are independent in wire sweeps, and can be routed simultaneously. The same applies to via segments in via sweeps. If a coarse-grained G-cell consists of $c \times c$ fine-grained G-cells and a coarse-grained via segment spans L layers, $c \times L$ and $c \times c$ threads are needed for wire and via sweeps respectively.

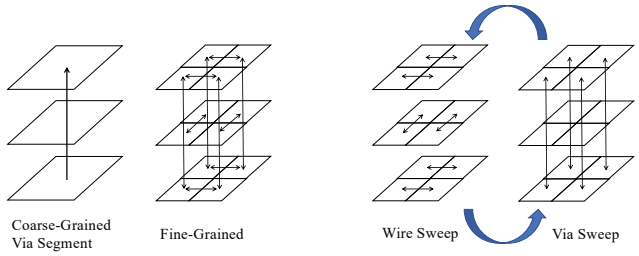

Figure 8: Via Segments

4.3 CUDA Graph Optimization

The approach in [10] is used in our coarse-grained maze routing, which is a highly parallel shortest path finding algorithm by alternating between wire and via sweeps where a wire/via sweep is to update the shortest distance with wires/vias only. It is similar to how via segments are handled, which was discussed in Section 4.2.2. However, there are different levels of parallelism. If a grid graph has $L \times n \times n$ G-cells and each coarse-grained G-cell consists of $c \times c$ fine-grained G-cells, handling a via segment (Section 4.2.2) needs $\max\{c \times L, c \times c\}$ threads while the coarse-grained routing needs $L \times \lceil n/c \rceil^2$ threads. The latter one is so large that a kernel with multiple blocks of threads is needed for one sweep operation. Although high parallelism can be achieved, the overhead of kernel calls is very significant. To resolve this issue, we resort to CUDA graph optimization.

CUDA graphs can pack multiple kernels, defined as a dependency graph, into one CUDA graph instance. The overhead of launching such an instance is much less than that of launching the kernels individually. However, building a CUDA graph instance also has overhead. A CUDA graph instance should be executed many times in order to benefit from this new feature. If an alternating sweep operation have T (5 in our implementation) iterations, routing two pins needs $2 \cdot T$ kernels for wire and via sweeps. Routing a net of $P + 1$ pins needs $P \cdot 2T$ kernel calls in total. Note that P is different for different nets, which makes it hard to construct CUDA graph instances that can be used by many nets.

There are many approaches with different trade-offs between the overheads of CUDA graph instantiation and CUDA graph execution. A straightforward method is to construct a CUDA graph instance with $2T$ kernels, and the instance will be launched P times for a net of $P + 1$ pins. Another extreme approach is to construct CUDA graph instances for routing nets of $2^i + 1$ pins (i.e., $2^i \cdot 2T$ kernels), where i ranges from 1 to N and 2^N is equal to the maximum P among all the nets. A net of $P + 1$ pins can be routed by launching no more than $\lceil \log_2 P \rceil$ CUDA graph instances. For example, a net of 27 pins can be routed with two instances of $2^4 \cdot 2T$ and $2^3 \cdot 2T$ kernels because $26 = 2^4 + 2^3$. In general, a 1 in the i-th bit of the binary representation of P means that the instance of $2^i \cdot 2T$ kernels should be launched. This approach needs $\log_2 \max\{P\}$ CUDA graph instances and $\lceil \log_2 P \rceil$ instance executions per net, while the previous simple method needs 1 CUDA graph instance with P instance executions per net. The latter approach is more balanced between the overheads of building and executing CUDA graph instances. In practice, N is flexible and can be set to a smaller number, because only a small portion of nets have many pins and it does not make a big difference in running time for them to execute more CUDA graph instances.

5 EXPERIMENTAL RESULTS

5.1 Experiment Setup

All the experiments are conducted on a 64-bit Linux workstation with Intel Xeon Silver 4114 CPUs (2.20 GHz, 20 cores) and 256 GB memory. One NVIDIA GeForce RTX 3090 graphics card is used. The benchmarks are from the 2019 CAD contest at ICCAD[4]. The global routing results are evaluated by the official evaluator[1] of the contest with Innovus v18.12-s106.

5.2 Global Routing Results

Table 2 shows the global routing results obtained by running the open-source code or provided executables on our machine and using the official evaluator. Eight threads are used for all the routers, which is consistent with what is used in their original papers. The run time is in seconds. Smaller scores are better. Our global router has the shortest running time in every benchmark and has the best average routing quality. Compared to a multi-threaded 3D global router CUGR, our global routing is 13.3× faster and has 0.2% better quality. Compared to a multi-threaded 2D global router SPRoute 2.0, our router is 2× faster and has 0.3% better quality.

5.3 Comparison of L-Shape/Z-Shape Routing

The advantage of Z-shape routing over L-shape routing is not only better routing quality, but also less overall global routing time. This is because Z-shape routing is more flexible and can avoid congestion better than L-shape routing, so fewer nets need to be ripped up and rerouted using time-consuming maze routing. This is demonstrated by the experimental results shown in Table 3. L-shape routing on average is over 10% faster than Z-shape routing, but the average difference of pattern routing all the nets is only 0.366 seconds. However, L-shape routing results in 37.9% more nets being ripped up compared to Z-shape routing. This difference makes the maze routing after L-shape routing 3.5 seconds slower than that after Z-shape routing. Global routing with Z-shape routing also has better routing quality as we expected. It is 0.2% better than global routing with L-shape routing.

[1]Note that we strictly follow the official evaluation metrics here for all the global routers. The results shown in this paper may be different from those shown in some previous papers, where different evaluation metrics are used.

Table 2: Global Routing Results

Benchmark	CUGR[11]		FastGR[12]		GAMER[10]		SPRoute 2.0[6]		Ours	
	Time	Score	Time	Score	Time	Score	Time	Score	Time	Score
ispd18_test5	73.1	16104127	34.5	16144948	48.0	16099658	9.5	16266324	5.6	16157416
ispd18_test8	282.9	37937622	123.8	37899025	108.0	37936493	26.0	38303840	15.4	38123108
ispd18_test10	373.2	40593601	151.5	40674135	126.0	40523120	33.0	42865920	17.5	41257617
ispd19_test7	652.4	88481579	225.3	88502557	179.0	88478185	46.6	87406430	30.6	88686088
ispd19_test8	431.1	128287651	222.2	128446998	253.0	128411260	65.5	127723613	37.0	127403748
ispd19_test9	620.9	201500802	321.1	201490357	355.0	201161603	101.3	199900738	59.2	199574710
ispd18_test5_metal5	93.5	16206355	46.5	16224663	55.0	16206853	11.1	16101826	7.8	16559899
ispd18_test8_metal5	289.9	37313105	181.2	37240290	149.0	37262694	45.9	39604236	24.1	37738787
ispd18_test10_metal5	399.1	46068410	212.1	48513161	192.0	45685561	97.2	47267127	22.0	47909903
ispd19_test7_metal5	434.6	82368279	232.4	82243597	164.0	82349960	51.5	81104527	32.8	81344536
ispd19_test8_metal5	670.2	126219722	472.5	126674690	265.0	126175581	128.3	126389940	48.4	126366645
ispd19_test9_metal5	732.9	197686583	652.9	197866328	367.0	197778170	128.3	197033254	80.1	195588171
Average	421.2	84897320	239.7	85160062	188.4	84839095	62.0	84997315	31.7	84725886
Ratio	13.3	1.002	7.6	1.005	5.9	1.001	2.0	1.003	**1.0**	**1.000**

Table 3: Comparison of L-shape and Z-shape Routing

Benchmark	L-Shape				Z-Shape			
	Pattern Route	Maze Route		Score	Pattern Route	Maze Route		Score
	Time (s)	#Nets	Time (s)		Time (s)	#Nets	Time (s)	
ispd18_test5	0.869	858	1.335	16239945	0.927	589	1.157	16157416
ispd18_test8	1.701	2188	5.157	38122030	1.812	1631	4.536	38123108
ispd18_test10	2.346	5430	5.962	41182093	2.655	4988	5.911	41257617
ispd19_test7	4.115	5386	13.527	88633320	4.494	3033	9.259	88686088
ispd19_test8	5.044	2811	9.443	127736730	5.572	1482	6.928	127403748
ispd19_test9	11.701	3928	16.243	199479861	12.508	1906	9.932	199574710
ispd18_test5_metal5	0.415	9935	4.805	16782092	0.544	6985	3.971	16559899
ispd18_test8_metal5	0.773	31335	16.679	37801441	0.997	24822	14.441	37738787
ispd18_test10_metal5	1.115	34993	13.296	48127503	1.284	27859	12.052	47909903
ispd19_test7_metal5	1.524	22610	19.287	81344646	2.005	18678	16.897	81344536
ispd19_test8_metal5	2.271	33180	30.126	126580968	2.800	22996	23.762	126366645
ispd19_test9_metal5	5.387	49409	55.044	196489341	6.054	31523	40.024	195588171
Average	3.105	16839	15.909	84876664	3.471	12208	12.406	84725886
Ratio	**0.895**	1.379	1.282	1.002	1.000	**1.000**	**1.000**	**1.000**

5.4 CUDA Graph Optimization

CUDA graph optimization was introduced in Section 4.3. In this subsection, we study the trade-off between CUDA graph instantiation and CUDA graph execution by setting different numbers of instantiated CUDA graphs (variable N in Section 4.3). Note that besides the kernels mentioned in Section 4.3, one more kernel is added for fine-grained routing. Since a batch of non-overlapping nets can be routed simultaneously, we need to have B sets of CUDA graphs to accommodate all the nets in a batch, where B is the largest size of a batch. Table 4 shows the GPU running time on maze routing with and without CUDA graphs. $N = 0$ is the results without CUDA graph optimization. Using CUDA graphs brings 33% reduction in GPU time. Larger N implies more instantiated CUDA graphs and fewer CUDA graph executions. In Table 4, the instantiation time increases and the execution time decreases as N becomes larger. When it comes to the total time, $N = 1$ and $N = 2$ are the best. According to the experimental results, increasing N does not further reduce the GPU running time in our routing framework. The final CUDA graph optimization in our implementation is to instantiate one CUDA graph for routing 2 pins, and launch the graph P times to route a net of $P + 1$ pins. Net-level parallelism is also used, so we have B CUDA graphs with the same functionality (routing 2 pins) but different parameters (e.g., memory locations of the nets).

Table 4: GPU Time on Maze Routing with Different CUDA Graph Optimizations

Benchmark	Instantiation Time (s)			Execution Time (s)			Total Time (s)			
	N=1	N=2	N=3	N=1	N=2	N=3	N=0	N=1	N=2	N=3
ispd18_test5	0.05	0.13	0.36	0.77	0.70	0.51	1.00	0.82	0.83	0.87
ispd18_test8	0.05	0.17	0.41	2.79	2.68	2.47	3.25	2.84	2.85	2.88
ispd18_test10	0.06	0.21	0.45	2.84	2.66	2.48	4.03	2.90	2.87	2.93
ispd19_test7	0.06	0.14	0.43	5.53	5.45	5.26	6.37	5.59	5.59	5.69
ispd19_test8	0.06	0.15	0.34	4.85	4.77	4.78	5.31	4.91	4.92	5.12
ispd19_test9	0.05	0.16	0.33	7.18	7.12	6.99	7.67	7.23	7.28	7.32
ispd18_test5_metal5	0.04	0.13	0.35	2.92	2.84	2.67	5.65	2.96	2.97	3.02
ispd18_test8_metal5	0.06	0.15	0.40	10.19	10.04	9.84	17.69	10.25	10.19	10.24
ispd18_test10_metal5	0.06	0.18	0.47	7.66	7.58	7.40	17.25	7.72	7.76	7.87
ispd19_test7_metal5	0.05	0.21	0.45	11.12	10.95	10.79	14.45	11.17	11.16	11.24
ispd19_test8_metal5	0.08	0.19	0.47	16.60	16.45	16.20	20.67	16.68	16.64	16.67
ispd19_test9_metal5	0.05	0.19	0.48	29.40	29.23	28.85	33.30	29.45	29.42	29.33
Average	0.06	0.17	0.41	8.49	8.37	8.19	11.39	8.54	8.54	8.60
Ratio							1.33	**1.00**	**1.00**	1.01

Table 5: Comparison of GPU-Accelerated Pattern Routing

Benchmark	Pattern Routing Time (s)		
	FastGR[12] (L-Shape)	Ours	
		L-Shape	Z-Shape
ispd18_test5	5.96	0.87	0.93
ispd18_test8	11.05	1.70	1.81
ispd18_test10	12.91	2.35	2.65
ispd19_test7	17.97	4.12	4.49
ispd19_test8	23.25	5.04	5.57
ispd19_test9	29.78	11.70	12.51
ispd18_test5_metal5	4.01	0.42	0.54
ispd18_test8_metal5	7.07	0.77	1.00
ispd18_test10_metal5	8.40	1.12	1.28
ispd19_test7_metal5	11.69	1.52	2.00
ispd19_test8_metal5	15.13	2.27	2.80
ispd19_test9_metal5	20.00	5.39	6.05
Average	13.94	3.11	3.47
Ratio	4.49	**1.00**	1.12

Table 6: Comparison of GPU-Accelerated Maze Routing

Benchmark	GAMER[10]	Ours
ispd18_test5	36.303	1.157
ispd18_test8	67.244	4.536
ispd18_test10	72.963	5.911
ispd19_test7	123.098	9.259
ispd19_test8	169.371	6.928
ispd19_test9	252.057	9.932
ispd18_test5_metal5	14.245	3.971
ispd18_test8_metal5	29.497	14.441
ispd18_test10_metal5	30.788	12.052
ispd19_test7_metal5	62.459	16.897
ispd19_test8_metal5	93.082	23.762
ispd19_test9_metal5	144.556	40.024
Average	91.305	12.406
Ratio	7.360	**1.000**

5.5 Comparison of GPU-Accelerated Routing

To better demonstrate the efficiency of our pattern routing and maze routing, we compare our pattern routing with the GPU-accelerated pattern routing in FastGR, and our maze routing with the GPU-accelerated maze routing in GAMER.

The pattern routing comparison are shown in Table 5. Our Z-shape pattern routing is over 4× faster than the L-shape pattern routing in FastGR. Table 6 shows the maze routing comparison. Note that the running time consists of both the coarse-grained routing time and the fine-grained routing time. Our maze routing is over 7× faster than GAMER.

6 CONCLUSION

In this paper, we propose a superfast full-scale GPU-accelerated global router. In pattern routing, we propose a new simple and fast parallel L-shape routing, and extend it to an efficient and high-quality Z-shape routing. In maze routing, we divide the fine-grained routing into interleaving wire and via segments, and solve them using different parallel methods. CUDA graphs are introduced to reduce overhead in maze routing. Experimental results show that our global routing achieves the shortest running time and has the best routing quality compared to other state-of-the-art LEF/DEF-based global routers.

7 ACKNOWLEDGEMENTS

We would like to thank Prof. Evangeline F.Y. Young for the discussions and advice.

REFERENCES

[1] Gengjie Chen, Chak-Wa Pui, Haocheng Li, and Evangeline F. Y. Young. 2020. Dr. CU: Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 9 (2020), 1902–1915. https://doi.org/10.1109/TCAD.2019.2927542

[2] Wing-Kai Chow, Liang Li, Evangeline F.Y. Young, and Chiu-Wing Sham. 2014. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach. *Integration* 47, 1 (2014), 105–114. https://doi.org/10.1016/j.vlsi.2013.08.001

[3] Chris Chu and Yiu-Chung Wong. 2008. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 1 (2008), 70–83. https://doi.org/10.1109/TCAD.2007.907068

[4] Sergei Dolgov, Alexander Volkov, Lutong Wang, and Bangqi Xu. 2019. 2019 CAD Contest: LEF/DEF Based Global Routing. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–4. https://doi.org/10.1109/ICCAD45719.2019.8942107

[5] Yiding Han, Koushik Chakraborty, and Sanghamitra Roy. 2013. A global router on GPU architecture. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 78–84. https://doi.org/10.1109/ICCD.2013.6657028

[6] Jiayuan He, Udit Agarwal, Yihang Yang, Rajit Manohar, and Keshav Pingali. 2022. SPRoute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 586–591. https://doi.org/10.1109/ASP-DAC52403.2022.9712557

[7] Jiayuan He, Martin Burtscher, Rajit Manohar, and Keshav Pingali. 2019. SPRoute: A Scalable Parallel Negotiation-based Global Router. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942105

[8] Haocheng Li, Gengjie Chen, Bentian Jiang, Jingsong Chen, and Evangeline F. Y. Young. 2019. Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–7. https://doi.org/10.1109/ICCAD45719.2019.8942074

[9] Liang Li, Zaichen Qian, and Evangeline F. Y. Young. 2009. Generation of optimal obstacle-avoiding rectilinear Steiner minimum tree. In *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*. 21–25.

[10] Shiju Lin, Jinwei Liu, and Martin D.F. Wong. 2021. GAMER: GPU Accelerated Maze Routing. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD51958.2021.9643563

[11] Jinwei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline F. Y. Young. 2020. CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218646

[12] Siting Liu, Peiyu Liao, Rui Zhang, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. 2022. FastGR: Global Routing on CPU-GPU with Heterogeneous Task Graph Scheduler. In *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*.

[13] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao. 2013. NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing. *IEEE Transactions on computer-aided design of integrated circuits and systems* 32, 5 (2013), 709–722.

[14] Yue Xu, Yanheng Zhang, and Chris Chu. 2009. FastRoute 4.0: Global router with efficient via minimization. In *2009 Asia and South Pacific Design Automation Conference*. 576–581. https://doi.org/10.1109/ASPDAC.2009.4796542