Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Sister Conferences Best Papers Track

FastGR : Global Routing on CPU-GPU with Heterogeneous Task Graph Scheduler (Extended Abstract)*

Siting Liu[1,2] , **Yuan Pu**[1] , **Peiyu Liao**[1,2] , **Hongzhong Wu**[3] , **Rui Zhang**[3] , **Zhitang Chen**[4] , **Wenlong Lv**[4] , **Yibo Lin**[2] and **Bei Yu**[1]
[1] The Chinese University of Hong Kong
[2] Peking University
[3]HiSilicon Technologies Co.
[4]Huawei Noah's Ark Lab

Abstract

Running time is a key metric across the standard physical design flow stages. However, with the rapid growth in design sizes, routing runtime has become the runtime bottleneck in the physical design flow. To improve the effectiveness of the modern global router, we propose a global routing framework with GPU-accelerated routing algorithms and a heterogeneous task graph scheduler, called FastGR. Its runtime-oriented version FastGR$^{\mathrm{L}}$ achieves 2.489× speedup compared with the state-of-the-art global router. Furthermore, the GPU-accelerated L-shape pattern routing used in FastGR$^{\mathrm{L}}$ can contribute to 9.324× speedup over the sequential algorithm on CPU. Its quality-oriented version FastGR$^{\mathrm{H}}$ offers further quality improvement over FastGR$^{\mathrm{L}}$ with similar acceleration.

1 Introduction

Routing is essential in the design flow of the modern very-large-scale integration. Modern routing flow is divided into global routing and detailed routing. Global routing produces routing guidance for detailed routing by performing rough routing on a coarse grid graph [Liu *et al.*, 2020]. The efficiency and efficacy of global routing are crucial to the design closure due to its recurrent invocation and guiding role.

The literature has extensively explored shortest path searching with GPU [Djidjev *et al.*, 2015]. However, most work only explores the basic single-source shortest path algorithm. These algorithms are unsuitable for routing since we must route millions of nets while considering numerous objectives and limitations such as wirelength, number of vias, and design rules. Regarding those modern routing challenges, more appropriate GPU kernel algorithms should be designed.

In this work, we propose FastGR, a global routing framework accelerated for CPU-GPU platforms. The framework leverages a GPU-friendly pattern routing algorithm and a task graph scheduler for heterogeneous CPU-GPU systems. By utilizing the processing resources of GPUs, we can further increase the solution quality performance of our global routing framework while incurring a little runtime overhead. We develop two variants of our global routing framework: the runtime-oriented version FastGR$^{\mathrm{L}}$ and the quality-oriented version FastGR$^{\mathrm{H}}$. Experiments show that when compared to the state-of-the-art global router [Liu *et al.*, 2020], our runtime-oriented version FastGR$^{\mathrm{L}}$ can achieve 2.489× overall speedup without any quality degradation. The quality-oriented version FastGR$^{\mathrm{H}}$ [Liu *et al.*, 2022b] reduces the number of shorts by 27.855% over the runtime-oriented version FastGR$^{\mathrm{L}}$ [Liu *et al.*, 2022a] while remaining 1.970× faster than the most advanced global router [Liu *et al.*, 2020].

2 Preliminaries

2.1 Problem Formulation

A grid graph $G(V, E)$ is defined to formulate global routing problem by considering each G-cell as a vertex ($v \in V$) and drawing an edge ($e \in E$) between all the pairs of adjacent G-cells. Figure 1 illustrates the procedure of grid graph construction. We map all the pins into G-cells according to the pin position. In this sample, different colors represent different metal layers. There is a preferred routing direction (horizontal or vertical) for wire edges in each metal layer, represented as the colored solid lines. The black dotted lines mean the via edges in our grid graph. With the grid graph G construction, the global routing problem can be formulated as the minimum accumulated cost path searching problem on G for all the nets defined in VLSI designs.

2.2 Modern Global Router

Pattern routing [Kastner *et al.*, 2002] plays an important role in the modern global routing framework due to its efficiency. Two popular patterns are shown in Figure 2. We illustrate the L-shape and Z-shape pattern routing paths on 2D and 3D routing spaces. As shown in Figure 2, the L-shape pattern includes one single bend point to change the routing direction, while the Z-shape pattern routing path contains two bends.

*This work has been initially presented at the IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE) in 2022 and IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) in 2023.

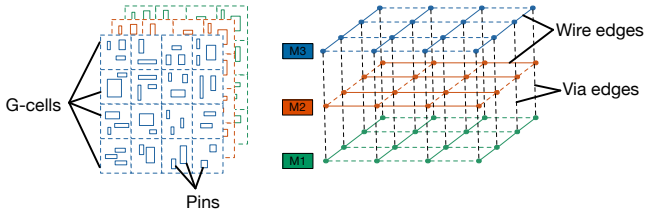

Figure 1: Grid graph construction procedure; There are 3 metal layers with 4×4 grids in each layer.

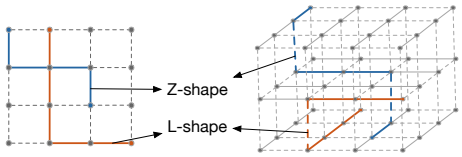

Figure 2: 2D/3D pattern routing; The red path represents one L-shape pattern routing solution, and the blue path is one of the candidate Z-shape pattern routing paths.

3 Algorithms

3.1 Overview

Figure 3 depicts the overall flow of FastGR. To begin, we present a heterogeneous task graph scheduler and use it to manage the execution order of multiple routing tasks in both portions of our global routing framework, the pattern routing stage, and the rip-up and reroute iterations. The conflicting relationship among these tasks is used to form the task graph. It is important to note that a conflict between two routing tasks indicates that they cannot be processed at the same time. Our task graph scheduler is utilized to determine the execution order of each conflicting pair of tasks.

3.2 Task Graph Scheduler

To determine the execution order of the global routing tasks, we develop a two-stage task graph scheduler. The first stage is to create a task conflict graph. The task graph scheduler is then used to determine the order of execution for each conflict edge. Following the task conflict graph generation, we extract one maximum non-conflicting set as the root task batch. All of these tasks can be divided into two groups: the root task batch and the non-root task batch. We assign the execution order to each pair of conflicting tasks as follows.

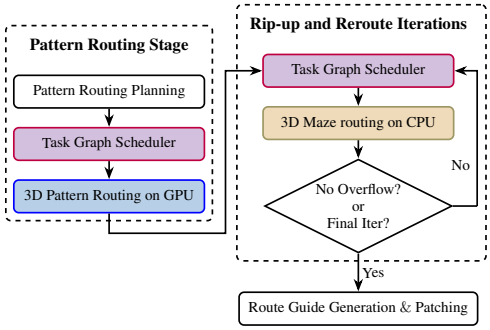

Figure 3: Overall flow of FastGR.

Notation	Description
L	The number of metal layers.
T_l	The point on l^{th} layer with the same 2D position as P_t.
$\mathcal{T}(P_s)$	The sub-tree rooted at P_s in the ordered multi-pin net.
$c(\cdots)$	The cost of a routing path for a single two-pin net.
$c_{\text{bc}}(P_s, l_s)$	The bottom children cost with P_s in the l_s layer.
$\mathbf{W}^{(i)}$	Weight matrix of part i in our computation graph flow.
$(B^{s(i)}, B^{t(i)})$	The i^{th} bend point pair of 2D patterns.
$B_l^{s(i)}$ $B_l^{t(i)}$	The bend points on l^{th} layer of 3D patterns.
$c^{*(i)}(P_s, P_t, l_t)$	The minimum cost with $(B^{s(i)}, B^{t(i)})$.
$c^*(P_s, P_t, l_t)$	The minimum cost of $\mathcal{T}(P_s)$ attached with $P_s \rightarrow T_{l_t}$.

Table 1: Notations for GPU-friendly pattern routing.

- *Only one task is in the root task batch.* The execution direction is from the root batch task to the other.
- *Both the tasks are not part of the root batch.* The execution order is from the task with a smaller task ID to the other and the task ID indicates the sorting result.

3.3 GPU-friendly 3D Pattern Routing

Figure 4 shows the programming architecture of our GPU-friendly pattern routing framework for all of these routing tasks during the pattern routing stage. Each batch in Figure 4 represents a single routing task, and each task contains several multi-pin nets. As shown in Figure 2, there are two common patterns in pattern routing approaches, where the L-shape pattern provides two candidate routing paths in 2D space and $L \times L$ candidate routing paths in 3D space. At the same time, there are two bend points in Z-shape patterns named the source bend point, and the target bend point since one connects to the source pin, and the other connects to the target pin. Without loss of generality, we discuss two bend points here and treat L-shape patterns as a special case. the bend point pair can get $M + N$ candidate paths in 2D space, where M represents the width of the bounding box of the net on G and N is the height. Section 3.3 defines the notations used in our GPU-friendly 3D pattern routing algorithms for the two-pin net $P_s \rightarrow P_t$.

We define a GPU-friendly 3D pattern routing algorithm as shown in Figure 5. The left part in Figure 5 illustrates one of the solutions of $P_s \rightarrow P_t$ with two bend points $(B^{s(i)}, B^{t(i)})$, where i means the index of the candidate bend point pair and $1 \leq i \leq M + N$. We define l_s, l_b, and l_t as the layer of the source pin, the wire connecting two bend points, and the target pin respectively.

The double-bend routing path includes three parts,

- the wire connecting the source point P_s to the source bend point $B_{l_s}^{s(i)}$;
- the vias to change routing metal layers from l_s to l_b and the wire connecting to the target bend point $B_{l_b}^{t(i)}$;
- the vias to change routing metal layers from l_b to l_t and the wire connecting to the target point T_{l_t}.

In this sample path, l_s is 1, l_b is 2 and l_t is 4. We denote this candidate double-bend pattern routing path as $\mathcal{P}\{P_s, B_{l_s}^{s(i)}, B_{l_b}^{t(i)}, T_{l_t}\}$. According to the above three parts,

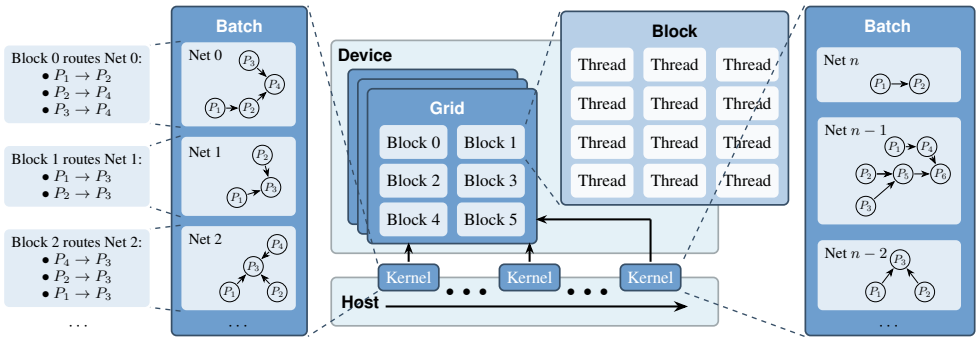

Figure 4: Programming architecture for the pattern routing stage; On the host, the kernels are invoked sequentially and each of them is used to process a single batch of nets. The nets in the same batch will be routed simultaneously on different blocks on the device.

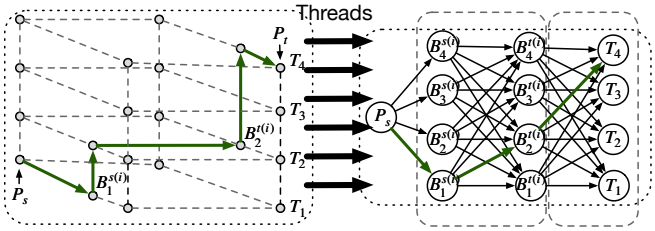

Figure 5: GPU-friendly 3D pattern routing flow for i^{th} candidate bend point pair $(B^{s(i)}, B^{t(i)})$; Each 3D pattern routing solution is denoted as $\mathcal{P}\{P_s, B_{l_s}^{s(i)}, B_{l_t}^{t(i)}, T_{l_t}\}$. The sample routing path is colored by green.

the formal formulation of the path cost is,

$$\begin{aligned} c(\mathcal{P}\{P_s, B_{l_s}^{s(i)}, B_{l_b}^{t(i)}, T_{l_t}\}) &= c_{\text{w}}(P_s, B^{s(i)}, l_s) \\ &+ c_{\text{v}}(B^{s(i)}, l_s, l_b) + c_{\text{w}}(B^{s(i)}, B^{t(i)}, l_b) \\ &+ c_{\text{v}}(B^{t(i)}, l_b, l_t) + c_{\text{w}}(B^{t(i)}, T_{l_t}, l_t). \end{aligned} \tag{1}$$

For each pair of bend points $(B^{s(i)}, B^{t(i)})$, we will generate the candidate flow i for this bend point pair. $c^{*(i)}(P_s, P_t, l_t)$ represents the minimum cost result of the two-pin net $P_s \rightarrow P_t$ in the i^{th} candidate flow with the 3D pattern routing algorithm. The calculation of $c^{*(i)}(P_s, P_t, l_t)$ with the inter-net ordering is,

$$\begin{aligned} c^{*(i)}(P_s, P_t, l_t) = \min_{0<l_s,l_b\le L} \Big\{ & c_{\text{bc}}(P_s, l_s) \\ & + c(\mathcal{P}\{P_s, B_{l_s}^{s(i)}, B_{l_b}^{t(i)}, T_{l_t}\}) \Big\}. \end{aligned} \tag{2}$$

We propose a merge step to merge the results of all $M + N$ candidate flows. We can get the final minimum cost as follows,

$$c^*(P_s, P_t, l_t) = \min_{1\le i\le M+N} c^{*(i)}(P_s, P_t, l_t). \tag{3}$$

To better utilize the GPU resources, we reformulate it as the computation graph flow using the vector/matrix addition and minimum operation. Our proposed GPU-friendly pattern routing algorithm for the i^{th} candidate bend point pair $(B^{s(i)}, B^{t(i)})$ is shown in the right part of Figure 5. The weight of the edge $P_s \rightarrow B_{l_s}^{s(i)}$ includes the bottom children cost $c_{\text{bc}}(P_s, l_s)$ and the wire cost to connect P_s and $B^{s(i)}$. The formulation of l_s^{th} of the edge weights $\vec{w}^{(1)}$ is,

$$w_{l_s}^{(1)} = c_{\text{bc}}(P_s, l_s) + c_{\text{w}}(P_s, B^{s(i)}, l_s),\ 0 < l_s \le L. \tag{4}$$

The metal layer switch procedure at the source bend point is represented by the connection between $B_{l_s}^{s(i)}$ and $B_{l_b}^{t(i)}$. Based on the connection, we formulate the entry of the edge weights matrix $\mathbf{W}^{(2)}$ at the l_s^{th} row and the l_b^{th} column as

$$w_{l_s,l_b}^{(2)} = c_{\text{v}}(B^{s(i)}, l_s, l_b) + c_{\text{w}}(B^{s(i)}, B^{t(i)}, l_b),\ 0 < l_s, l_b \le L. \tag{5}$$

We can also define the metal layer switch procedure at the target bend point as the connection between $B_{l_b}^{t(i)}$ and T_{l_t}. The entry at l_b^{th} row and the l_t^{th} column of the edge weights matrix $\mathbf{W}^{(3)}$ is defined as,

$$w_{l_b,l_t}^{(3)} = c_{\text{v}}(B^{t(i)}, l_b, l_t) + c_{\text{w}}(B^{t(i)}, T_{l_t}, l_t),\ 0 < l_b, l_t \le L. \tag{6}$$

The minimum cost $c^{*(i)}(P_s, P_t, l_t)$ of the candidate bend point pair $(B^{s(i)}, B^{t(i)})$ can be calculated as Equation (7) referring to Equation (2),

$$c^{*(i)}(P_s, P_t, l_t) = \min_{0<l_s,l_b\le L} \left\{ w_{l_s}^{(1)} + w_{l_s,l_b}^{(2)} + w_{l_b,l_t}^{(3)} \right\}. \tag{7}$$

Having all the minimum cost of $M + N$ candidate bend point pairs, we can finally get the $c^*(P_s, P_t, l_t)$ using the merge step in Equation (3) which can also be computed as the vector minimum operation on GPU.

Both L-shape and Z-shape patterns are included in the above formulations. Separate routing algorithms can be formulated by limiting bend point pair candidates.

3.4 Parallel Rip-up and Reroute Iterations

Each multi-pin net is treated as a separate routing task. Then, we apply our task graph scheduler to these routing tasks based on the conflict relationship. Finally, all these routing tasks follow the execution order determined in the ordered task graph. As a result, utilizing Taskflow [Huang *et al.*, 2019] and the ordered task graph, we can quickly maximize the parallelism of our rip-up and reroute iterations.

Bench	Overall runtime (s)					PATTERN runtime (s)					MAZE runtime (s)				
	CUGR	FastGR^{L}	SPD	FastGR^{H}	SPD	CUGR	FastGR^{L}	SPD	FastGR^{H}	SPD	CUGR	FastGR^{L}	SPD	FastGR^{H}	SPD
18t5	80.645	24.858	3.24×	45.240	1.78×	45.111	4.967	9.08×	27.774	1.62×	21.588	5.219	4.14×	2.320	9.30×
18t5m	83.49	33.457	2.50×	40.509	2.06×	7.445	3.314	2.25×	13.084	0.57×	63.32	16.117	3.93×	13.604	4.65×
18t8	260.982	97.568	2.68×	133.717	1.95×	118.117	8.983	13.15×	53.110	2.22×	108.986	52.692	2.07×	46.397	2.35×
18t8m	250.26	131.395	1.91×	141.511	1.77×	18.292	5.715	3.20×	23.690	0.77×	201.58	79.987	2.52×	77.740	2.59×
18t10	354.347	110.811	3.20×	144.570	2.45×	124.533	10.410	11.96×	38.503	3.23×	199.777	70.222	2.85×	73.873	2.70×
18t10m	339.471	165.682	2.05×	169.670	2.00×	18.485	6.788	2.72×	15.858	1.17×	291.58	106.097	2.75×	103.631	2.81×
19t7	527.955	181.445	2.91×	231.654	2.28×	223.947	15.556	14.40×	85.943	2.61×	241.787	98.780	2.45×	77.569	3.12×
19t7m	340.489	179.115	1.90×	210.831	1.62×	34.208	8.658	3.95×	45.032	0.76×	244.686	105.852	2.31×	104.355	2.35×
19t8	529.193	173.299	3.05×	240.520	2.20×	333.965	17.612	18.96×	93.800	3.56×	96.888	53.312	1.82×	42.909	2.26×
19t8m	520.547	320.228	1.63×	317.376	1.64×	51.922	11.982	4.33×	43.210	1.20×	371.748	202.516	1.84×	169.857	2.19×
19t9	842.18	250.466	3.36×	320.815	2.63×	561.337	24.481	22.93×	103.690	5.41×	129.707	74.764	1.74×	60.311	2.15×
19t9m	632.577	434.852	1.46×	498.958	1.27×	88.971	17.976	4.95×	52.054	1.71×	400.227	247.518	1.62×	283.915	1.41×
AVG			2.49×		1.97×			9.32×		2.07×			2.50×		3.16×

Table 2: Runtime speedup (SPD) results on ICCAD 2019 benchmarks. AVG means the averaged one.

Bench	Score		Wirelength		# Vias		# Shorts		
	FastGR^{L}	FastGR^{H}	FastGR^{L}	FastGR^{H}	FastGR^{L}	FastGR^{H}	FastGR^{L}	FastGR^{H}	Improved (%)
18t5	16919500	**16880800**	26988200	**26915900**	856362	**855723**	0	0	
18t5m	18774900	**18705100**	27942700	**27799600**	**802385**	809459	3188	**3135**	1.662
18t8	40881100	**40825200**	64359900	**64211700**	**2174240**	2179840	8.5	**0**	100.000
18t8m	42897000	**42728700**	64554600	**64441200**	**1948740**	1957020	5649.5	**5360**	5.124
18t10	42592100	**42575900**	66718200	**66677100**	**2308250**	2309320	0	0	
18t10m	44579100	**44275300**	71071700	**71014000**	**2059820**	2067080	1608	**1000**	37.811
19t7	71887200	**71763600**	118744000	**118495000**	**3128750**	3129020	0	0	
19t7m	67923300	**67639200**	107115000	**106577000**	**3080660**	3084060	4086.5	**4028.5**	1.419
19t8	113929000	**113844000**	181854000	**181687000**	5750370	**5750260**	0	0	
19t8m	**114362000**	114545000	177638000	**177284000**	**5612590**	5623080	**6185.5**	6822	-10.290
19t9	175523000	**175367000**	274106000	**273884000**	9603400	**9603200**	112.5	**23.5**	79.111
19t9m	173339000	**173002000**	267786000	**267304000**	**9359980**	9375940	4013	**3692**	7.999
Average									27.855

Table 3: Solution quality results after global routing. The better performance is marked as **bold**.

4 Experimental Results

4.1 Experimental Setup

The framework was developed in C++/CUDA based on CUGR [Liu *et al.*, 2020]. We conducted the experiments on a 64-bit Linux machine with Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz and 1 NVIDIA GeForce RTX 3090 GPU. ICCAD2019 benchmarks[Dolgov *et al.*, 2019] were adopted. We integrated our proposed two types of GPU-friendly pattern routing algorithms into the pattern routing stage separately to illustrate the strength of our methods. Further, the details of sorting scheme and threshold settings can be found in [Liu *et al.*, 2022b].

4.2 Acceleration

We evaluate the total runtime, pattern routing runtime, and runtime of the rip-up and reroute iterations to demonstrate the acceleration performance. Section 4 shows that our proposed FastGR^{L} can bring an overall 2.489× acceleration with 9.324× speedup on the pattern routing stage. Meanwhile, the proposed FastGR^{H} can still obtain 1.970× speed up with much more candidate routing paths, in which the pattern routing stage achieves 2.070× acceleration on average and 3.157× improvement on the maze routing stage due to the reduction of the number of nets to rip up.

4.3 Evaluation on Hybrid Shape Patterns.

Section 4 shows the comparison of the solution quality after global routing. It establishes that the global router with hybrid-shape pattern routing algorithm FastGR^{H} can beat the global router with L-shape pattern routing algorithm FastGR^{L} [Liu *et al.*, 2022a] with respect to the solution quality on most designs. Since our hybrid-shape pattern routing algorithm considers Z-shape patterns as the candidate routing paths, the number of vias increases reasonably. To further evaluate the solution performance, Dr.CU [Chen *et al.*, 2019] is applied to conduct detailed routing under the guide of the global routing solution. As for the wirelength, our FastGR framework outperforms CUGR on most designs. Furthermore, FastGR can obtain comparable detailed routing performance with CUGR in many aspects (including the number of vias, the number of shorts, and the number of spacing violations) as listed in [Liu *et al.*, 2022b].

5 Conclusion

In this paper, we propose an efficient global routing framework, FastGR, accelerated for CPU-GPU platforms. We propose two GPU-friendly pattern routing algorithms and a heterogeneous task graph scheduler. The experimental results highlight the importance of GPU-accelerated kernel algorithms and the task scheduler for inter-net ordering in routing. An adequate fuse of them can assist in reducing design cycles and improve the solution quality at the same time. In the future, we plan to extend the task graph scheduler to other critical stages and exploit the power of GPU acceleration in the VLSI flow.

References

[Chen *et al.*, 2019] Gengjie Chen, Chak-Wa Pui, Haocheng Li, and Evangeline FY Young. Dr. cu: Detailed routing by sparse grid graph and minimum-area-captured path search. *IEEE TCAD*, 39(9):1902–1915, 2019.

[Djidjev *et al.*, 2015] Hristo Djidjev, Guillaume Chapuis, Rumen Andonov, Sunil Thulasidasan, and Dominique Lavenier. All-Pairs Shortest Path algorithms for planar graph for GPU-accelerated clusters. *Journal of Parallel and Distributed Computing*, 85:91–103, 2015.

[Dolgov *et al.*, 2019] Sergei Dolgov, Alexander Volkov, Lutong Wang, and Bangqi Xu. 2019 CAD contest: LEF/DEF based global routing. In *Proc. ICCAD*, pages 1–4, 2019.

[Huang *et al.*, 2019] Tsung-Wei Huang, Chun-Xun Lin, Guannan Guo, and Martin Wong. CPP-TaskFlow: Fast task-based parallel programming using modern C++. In *Proc. IPDPS*, pages 974–983, 2019.

[Kastner *et al.*, 2002] Ryan Kastner, Elaheh Bozorgzadeh, and Majid Sarrafzadeh. Pattern routing: Use and theory for increasing predictability and avoiding coupling. *IEEE TCAD*, 21(7):777–790, 2002.

[Liu *et al.*, 2020] Jinwei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline F. Y. Young. CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model. In *Proc. DAC*, pages 1–6, 2020.

[Liu *et al.*, 2022a] Siting Liu, Peiyu Liao, Rui Zhang, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. Fastgr : Global routing on cpu-gpu with heterogeneous task graph scheduler. In *Proc. DATE*, 2022.

[Liu *et al.*, 2022b] Siting Liu, Yuan Pu, Peiyu Liao, Hongzhong Wu, Rui Zhang, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. Fastgr: Global routing on cpu-gpu with heterogeneous task graph scheduler. *IEEE TCAD*, 2022.