Efficient Design Rule Checking with GPU Acceleration

Wei Zhong
DUT

Zhenhua Feng
DUT

Zhuolun He
CUHK

Weimin Wang
DUT

Yuzhe Ma
HKUST(GZ)

Bei Yu
CUHK

***Abstract*—Design rule checking (DRC) is an essential part of the chip design flow, which ensures that manufacturing requirements are conformed to avoid a chip failure. With the rapid increase of design scales, DRC has been suffering from runtime overhead. To overcome this challenge, we propose to accelerate DRC algorithms by harnessing the power of graphics processing units (GPUs). Specifically, we first explore an efficient data transfer approach for geometry information of a layout. Then we investigate GPU-based scanline algorithms to accommodate both intra-polygon checking and intre-polygon checking based on the characteristics of the design rules. Experimental results show that the proposed GPU-accelerated method can substantially outperform a multi-threaded DRC algorithm using CPU. Compared with the baseline with 24 threads, we can achieve an average speedup of $36\times$ and $201\times$ for spacing rule checks and enclosing rule checks on a metal layer, respectively.**

I. Introduction

Design rule checking (DRC) is the process to verify that a design layout conforms to a set of predefined design rules, which are established to ensure high manufacturability. These rules define geometric constraints to meet the physical limitations of the lithography and manufacturing process. With the rapid increase in design size, the design rule checking process becomes inevitably complicated and time-consuming. Many existing DRC algorithms focus on improving CPU efficiency, while the resulted speedup can be limited due to thread overhead, limited bandwidth, and CPU cache size. As the complexity of integrated circuits continues to climb, CPU-based multi-core solutions may not accommodate the large input layouts.

In this paper, we propose a GPU-accelerated framework to conduct DRC flow efficiently. Firstly, we customize an efficient method to transfer the required data from the host (CPU) side to the device (GPU) side, which addresses the bottleneck of data transfer resulted from the heterogeneity of the framework. Then we propose a GPU-based scanline algorithm, which covers detection within a single polygon as well as between polygons, in which the scanning and detection steps are parallelized. We use designs from the OpenROAD [1] project to test our algorithm and use `KLayout` [2] as our baseline to test the effectiveness of our approach.

II. Proposed Method

Fig. 1 illustrates the overview of our GPU-accelerated design rule checking flow. The blue block and the white block denote the computation on CPU and GPU, respectively. We start off by clipping the input layout into multiple regions. The interconnected polygons in the region should be merged before being transferred to the GPU. Then data transfer is conducted with the an efficient transfer method. Note that design rules can be defined in a single polygon, or between polygons on the same layer, or between polygons on different layers. Consequently, we develop different scanline strategies to find candidate edges pairs with potential violation according to different design rules. Finally, based on the candidate edges pairs, we then check them to obtain the truly conflicting edges.

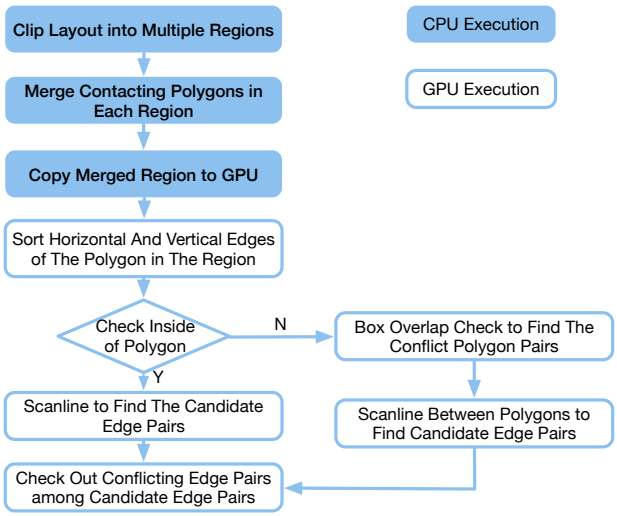

Fig. 1 Overview of our GPU-accelerate DRC.

A. Data Transfer and Preprocessing

If we launch data transfer in a polygon-based manner, the overhead would be roughly proportional to the number of polygons, which is not friendly to the layers containing small but numerous patterns. Ideally, we hope data transfer time to be linear in the size of useful geometry contents (i.e., number of edges in this case). To solve this problem, we resort to an efficient approach to transfer geometry data of a layout from CPU to GPU by switching the perspective from polygons to edges. Firstly, we extract all the edges of the polygons in a layer and attach an attribute to each edge to indicate which original polygon it belongs to. In this way, we effectively pack the original multiple polygons into a single collection of edges (i.e., a large 'dummy polygon'). Then we merge these edges into a matrix. After passing the merged edge matrix into the GPU, we reassemble the polygons based on the properties of each edge in the package.

Here, each polygon is represented as a set of consecutive connected edges, which is difficult to retrieve the position of the edge in the polygon directly from the edge information. However, we need to determine where a scanline starts and what direction the scanline sweeps through based on the position of the edges in a polygon. We classify polygon edges into

978-3-9819263-7-8/DATE23/© 2023 EDAA

horizontal ones and vertical ones, and sort the edges accordingly for further process.

B. GPU-based Scanline

In our GPU-accelerate scanline algorithm, according to the characteristics of design rule checking, we divide the checking into two scenarios: intra-polygon checking (*e.g.* width and space) and inter-polygon checking (*e.g.* space and enclosing). In the inter-polygon checking, we propose an overlap checking for coarse-grained filtering and an improved scanline algorithm for violation checking.

Scanline Inside Polygon. In the intra-polygon checking, we process horizontal and vertical edges separately. Compared with sequential scanline algorithm, we enable multiple scanlines in parallel. Specifically, for the vertical edges of a polygon, our algorithm scans from left to right starting from each edge simultaneously until the distance between the scanned edge and the starting edge is greater than our preset threshold. For the horizontal edges of a polygon, the scanning direction is from bottom to top. Since we have sorted the horizontal and vertical edges, there will be no duplicate records.

Overlap Checking. For the detection between polygons, we must know which two polygons are possible to have violations. Since polygons are generally very irregular, it is very difficult to directly detect whether it violates the rules with other polygons. Instead, we first represent each polygon with its bounding box. Then, if the box in which one polygon is located overlaps with the box extended by another polygon according to the threshold, they will form a conflict polygon pair and be recorded.

Scanline Between Polygons. In the inter-polygon checking, we allocate GPU threads for each conflicting polygon pair. In each thread we use the boundary range of the simple polygon to control the check range in the complex one. By scanning the obtained large polygon range we can get the candidate conflicting edge pairs that potentially introduce violations.

C. Violation Identification

After the scanline operation, we get a large set of candidate edges. However, there are many non-conflicting edges that need to be trimmed in the candidate edge pairs. We propose a detection strategy using GPU based on the characteristics of the candidate edge pairs. We assign a GPU thread to each candidate edge pair. Then in each kernel we will use the distance, projection, angle and other positional relationships of these two edges to check if they indeed violate the design rules, which involves simple operation, *e.g.* judgment, comparison, etc.

III. Experimental Results

Experimental Setup. We conduct various experiments on a 64-bit Ubuntu Linux machine with TITAN RTX GPU and 3.5GHz Intel Core i9-10920X CPU. The compilers include CUDA NVCC 10.2 and GNU GCC 5.4.0. We use 4096 threads for all kernel configurations and 1 CPU core for all host operations. Our baseline is `KLayout`0.26.6 [2]. The layout benchmarks in the experiments are generated by OpenROAD project [1] with default settings. In order to ensure fairness, the parameters and checking content set by our algorithm are exactly the same as those in `KLayout`.

TABLE I Enclosing check in `Metal1`

Design	gcd	aes	bp_be	bp
8 CPU threads	33.522	13194.039	58477.239	90250.85
16 CPU threads	34.212	13074.176	51671.131	85792.708
24 CPU threads	34.52	13072.36	49047.536	74497.754
Ours	**0.343**	**27.932**	**257.056**	**409.381**
Speedup	**100.641**×	**468.01**×	**190.80**×	**181.98**×
Average	**201.1**×			

TABLE II Enclosing check in `Metal2`

Design	gcd	aes	bp_be	bp
8 CPU threads	5.547	1977.047	2859.979	3332.67
16 CPU threads	5.732	1997.85	2435.594	2321.697
24 CPU threads	5.552	1976.503	2320.845	2298.961
Ours	**0.291**	**30.493**	**132.717**	**250.022**
Speedup	**19.08**×	**64.82**×	**17.49**×	**11.21**×
Average	**22.19**×			

TABLE III Space check in `Metal1`

Design	gcd	aes	bp_be	bp
8 CPU threads	10.99	376.244	5950.007	14865.705
16 CPU threads	11.131	3692.87	4540.09	8833.2
24 CPU threads	10.989	3690.08	4226.62	7565.84
Ours	**0.316**	**19.091**	**250.799**	**471.71**
Speedup	**34.78**×	**193.29**×	**16.85**×	**16.04**×
Average	**36.71**×			

TABLE IV Space check in `Metal2`

Design	gcd	aes	bp_be	bp
8 CPU threads	6.378	2732.5534	3870.166	5015.233
16 CPU threads	6.168	2703.47	3365.211	3767.176
24 CPU threads	6.174	2666.539	3114.918	3621.914
Ours	**0.399**	**28.591**	**121.279**	**238.966**
Speedup	**15.47**×	**93.26**×	**25.68**×	**15.16**×
Average	**27.38**×			

Runtime Performance. Our algorithm can significantly accelerate DRC. We compare the runtime of our algorithm with the runtime of the DRC in `KLayout`, where various levels of parallelism in `KLayout` are investigated. The DRC rules we use are extracted according to FreePDK45 [3]. It can be observed that our algorithm achieves significant speedup over CPU-based DRC (`KLayout`version). The speedup in the TABLE I to TABLE IV are calculated by comparing with the runtime using 24 CPU threads. TABLE I and TABLE III show the enclosing check and space check results of `Metal1` and `Via2` layers. TABLE II and TABLE IV show the enclosing check and space check results of `Metal2` and `Via2` layers. We can achieve 201× speedup on the enclosing check of the `Metal1` layer and 22× speedup on the enclosing check of the `Metal2` layer compared with the baseline. For the space check of the `Metal1` layer, we can improve the speed by an average of 36× compared to the baseline. And for the space check of the `Metal2` layer, we can improve the efficiency by an average of 27× compared to the baseline.

References

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proc. DAC*, 2019, pp. 1–4.

[2] "KLayout," https://klayout.de/.

[3] "FreePDK45," https://www.eda.ncsu.edu/.